ISSN 1751-8644
doi: 0000000000
www.ietdl.org

A Survey on Automated Symbolic Verification and its Application for Synthesizing Cyber-Physical Systems

Lucas C. Cordeiro[1]* *Eddie B. de Lima Filho*[2] *Iury V. Bessa*[3]

[1] *School of Computer Science, University of Manchester, Manchester, UK*
[2] *TPV Technology, Manaus, Brazil*
[3] *Department of Electricity, Federal University of Amazonas, Manaus, Brazil*
* *E-mail: lucas.cordeiro@manchester.ac.uk*

Abstract: Dependency on correct operation of embedded systems is rapidly growing, mainly due to their wide range of applications, such as micro-grids, automotive device control, health care, surveillance, mobile devices, telecommunications, internet of things, and consumer electronics. Their structures are becoming more and more complex and currently require multi-core processors with scalable shared memory, signal-processing pipelines, and sophisticated software modules, in order to meet increasing computational power, flexibility demands, and also to adapt to new scenarios and behaviors. Additionally, interaction with real-world entities and addition of modern communication capabilities further enhance the mentioned features and give rise to a new system class: *embedded and cyber-physical systems* (ECPS). As a consequence, reliability of ECPS becomes a key issue during system development, which must be carefully addressed and assured. Generally, state-of-the-art verification methodologies for ECPS generate test vectors (with constraints) and use assertion-based verification and high-level processor models, during simulation; however, new challenges arose, such as need for meeting time and energy constraints, handling concurrent software, dealing with platform restrictions, evaluating implementation-structure choices, validating operation logic, ensuring correct system behavior together with physical plants, providing compliance with target systems, incorporating knowledge about new problems and conditions, and supporting new software architectures and legacy designs, which now have to be tackled. This survey deals with the mentioned issues, reviews related literature, and discusses recent advances on symbolic model checking techniques and their applications to control synthesis. Additionally, challenges, problems, and recent advances to ensure correctness and timeliness, regarding ECPS, are discussed. Reliability issues, when developing ECPS, are then considered, as a prominent verification and synthesis application for achieving *correct-by-construction* systems.

1 Introduction

Generally, embedded computer systems perform dedicated functions with high degree of reliability, according to their original design and requirements (*e.g.*, real-time) [1]. They are ubiquitous in modern day information systems and are also becoming increasingly important in our society, due to their use to process, monitor, control, and even replace every human activity, which include the ones related to factories, power plants, mobile devices, traffic control, vehicles, and home duties [2]. Besides, they are also 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.

Interaction between embedded software and physical processes created a different class of systems, which are complex, highly integrated, and present a mixture of continuous and discrete dynamics, named as *hybrid systems* (HSs). Indeed, the embedded-system domain allied to the most recent communication revolution caused an intense integration between different and spread physical processes, which are called embedded and cyber-physical systems (ECPS) [3, 4]. In particular, the presence of modern communication technologies (*e.g.*, internet of things - IoT) causes a revolution in terms of flexibility, scalability, and complexity of those systems and adds a novel class of problems to challenges found in the embedded systems domain. For example, micro-grids, *i.e.*, small-scale electricity systems that gather different sources of distributed generation (DG) and loads, are emerging ECPS, where reliability and carbon emission reduction are of paramount importance [2]. One may also notice that each DG element and the majority of current loads already employ embedded software and present several safety-critical requirements; however, the integration of both constitute ECPS, where additional challenges related to synchronization, stability, control, communication, and reliability of entire micro-grids arise.

Thus, ECPS demand short development cycles and high-level of reliability and robustness [3, 5], besides presenting challenges that include but are not limited to the ones already imposed by embedded systems. As a consequence of their popularization, human life has also become more and more dependent on services provided by this type of system and, in particular, their success is strictly related to both service relevance and quality/reliability.

Fig. 1 illustrates subcomponents of general ECPS, inside the biggest blue box, which typically consist of human-machine interface (*e.g.*, keyboard and display), processing unit (*e.g.*, real-time computer system), and instrumentation interface (*e.g.*, sensor, network, and actuator) that can be connected to some physical plant [1]. Fig. 1 also shows some ECPS examples, including mass-production devices, multi-core processors embedded into consumer electronic devices, and safety-critical systems. Indeed, many current ECPS, such as unmanned aerial vehicles (UAVs) [6] and medical monitoring systems [7], become interesting solutions only if they can reliably perform their target tasks. For instance, UAVs are a trend on military missions and civil applications, since they can easily achieve places that cannot be accessed by humans, *i.e.*, without on-board pilots and also with different degrees of autonomy; however, an incorrect plan execution may cost civilian lives, which is unacceptable. In addition, wrong disease diagnosis or condition-evaluation reports have the potential to compromise patients' health and even many aspects of their lives, with serious consequences. On

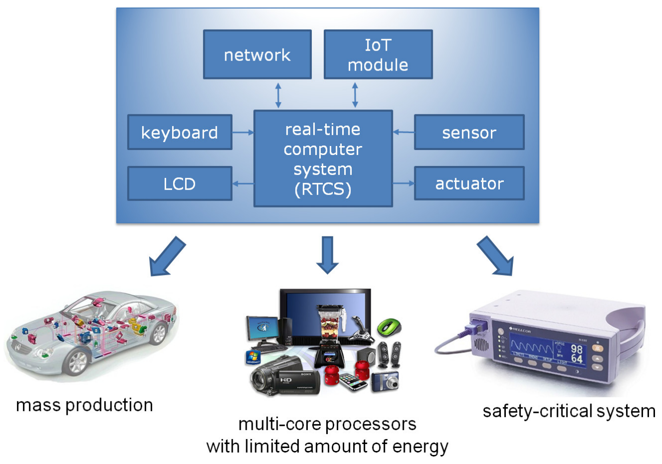

network
IoT module
keyboard
real-time computer system (RTCS)
sensor
LCD
actuator
mass production
multi-core processors with limited amount of energy
safety-critical system

Fig. 1: Subcomponents of general ECPS and some examples.

the one hand, portable ECPS are capable of monitoring and identifying conditions, which is very difficult for human specialists, mainly when their contact with patients happens only in medical clinics. On the other hand, wrong or incomplete diagnostic data may delay necessary treatments or deteriorate patients' health.

Besides, when physical interaction with the real-world is needed, which happens in ECPS, 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 [8], which then raises deeper concerns related to 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. Specifically, such representations can be used to evaluate and even synthesize a given particular system, by ensuring that all necessary functions are correctly implemented and an appropriate behavior is exhibited, *i.e.*, a system is indeed correct by its method of construction [9].

For instance, one may argue that a Linux device driver [10] was verified and correctly uses the related kernel services (*e.g.,* exported symbols and scheduling); however, not much can be said about correct access to the associated hardware and its use, *i.e.*, if *read* and *write* operations are performed according to what is specified in a device's data-sheet and setup periods are respected, for instance. In summary, it is important to ensure that the developed driver is compliant with the underlying hardware, which goes further than code correctness and also tackles interaction with surrounding entities.

Regarding behavior correctness, one important observation is that it is becoming more difficult, since fixed mathematical models are being replaced by machine learning approaches (ML) [11], where future operation may change based on data acquired during previous actions, due to model updating. Indeed, ML algorithms have a dual role on software verification development: model evolution and verification enhancement. Given the dissemination of such algorithms, several ECPS have embedded ML software and, as a consequence, system-correctness verification procedures should also adapt somehow and incorporate new checks (which may also rely on ML), when analyzing target models, or fed with acquired data, as recently reported in literature [12, 13]. In addition, model checking mechanisms have also incorporated ML algorithms, in order to increase efficiency of verification [14–16] and synthesis [17] procedures.

A number of distinctive characteristics might influence ECPS verification and synthesis processes, 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, 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 underlying hardware platforms (*e.g.*, real-time, memory allocation, interrupts, interfacing, and concurrency) [18] or even new structures provided with the goal of ensuring more computational capacity [19] must be considered during verification. Such observations expand the addressed issues and even complement what was previously mentioned.

As one may notice from previous paragraphs, verifying and safely synthesizing ECPS are challenging tasks, which involves many aspects of target applications. In addition, there is an increasing demand for correct-by-design ECPS, which prevent financial losses, given the associated high reliability and correct behavior, and provide systems best fitted to a given scenario. Indeed, while formal synthesis aims to obtain such systems, it cannot achieve that goal without robust formal verification techniques, which may even be integrated in an automated framework for system creation. Finally, it is worth noticing that synthesis and verification are linked through system (behavior) models, which present restrictions and include parameters capable of influencing a given system's operation. As pointed out by Zheng and Julien [20], mission-critical ECPS applications often rely on simulation-based testing; however, at the same time, simulation-centric tools present limitation in correctness assessment. Indeed, problems may even arise from hardware limitations not considered in simulations, which could be tackled through comprehensive systems models, and, in medical-device development, where correctness is usually pursued by code inspection, static analysis, and module and system integration testing, patients are not often considered [20]. Regarding the latter, a system model including both ECPS and patient, in association with formal techniques, would have the potential to show unforeseen problems related to their interaction, *i.e.*, models are often underrepresented in verification stages. Nonetheless, scalability to real-size systems and completeness are still concerns, which may be fulfilled with statistical [21] and induction [22] approaches, respectively.

Therefore, modeling, verification and synthesis of ECPS are interdependent problems. In particular, modeling of ECPS and properties is a sine qua non condition for ECPS verification, since the verification algorithms are performed based on these models that must trade off the accuracy and simplicity for compatibility with these (software) procedures. In addition, formal synthesis consists in designing systems that are correct according to the evaluation by the formal verification algorithms. Therefore, formal synthesis depend on formal verification and both are based on sound models of systems and properties. It is worth mentioning that adequate ECPS modeling for formal verification and synthesis, should take into account communication delays, network availability, faults, and cyber attacks.

This article represents a recent survey study about verification methods and their application to ECPS synthesis, while addressing both largely employed schemes and the ones with potential to improve current results or provide better fits to real and high-complex ECPS. Later on, we then focus on symbolic methods, specifically bounded model checking (BMC), which represents the authors' vision for future ECPS verification and synthesis schemes. In particular, in order to perform that, we have adopted an automated search mechanism via popular digital libraries, with the goal of finding the most relevant studies regarding ECPS. Here, we have used Google Scholar[*], Web of Science[†] and Scopus[‡] digital libraries, which represent the largest databases for scientific publications. Our main goals for this survey article can be described as follows:

- Briefly describe part of the established research about ECPS modeling, verification, and synthesis;
- Provide an overview about technical issues and recent advances in symbolic verification, together with synthesis applications of the latter that are currently employed for ECPS, while focusing on verification techniques based on Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT);

[*] `https://scholar.google.com/`
[†] `https://login.webofknowledge.com/`
[‡] `https://www.scopus.com/`

• Present open challenges on ECPS verification and synthesis and also clarify current trends, which is mainly driven by ECPS, IoT, and ML.

This survey is organized as follows. We present a brief discussion about models for ECPS, in Section 2, and then provide an overview on ECPS verification, in Section 3, while taking into account aggregate effects and specific domains. Later, in Section 4, we discuss specific topics in formal verification and promising approaches for verifying ECPS, which include BMC, k-induction verification based on invariant inference, property directed reachability (or IC3), abstraction, counterexample guided inductive synthesis, and system model incorporation. The state-of-the-art on formal synthesis for ECPS is discussed in Section 5. In Section 6, the main challenges regarding verification and synthesis of ECPS are described, while Section 7 tackles those problems that are (partially) open in current published research and later describes current achievements and future trends in ECPS verification and synthesis. Section 8 addresses new applications, which take into account the new trends (*e.g.*, ML) mentioned here. Finally, we describe the main limitations of this survey, in Section 9, and then conclude and present future work, in Section 10.

2 Modeling Embedded and Cyber-Physical Systems as Hybrid Systems for Verification and Synthesis

2.1 ECPS Concept and Definition

Automated formal verification has been employed to ensure correctness and reliability of ECPS, during the last decade, as well as provide controller synthesis [23–29] for ECPS and HSs [21, 30–35]. ECPS reliability is related to its ability to maintain correct execution of specific functions, which depends on correctness of embedded software and hardware compatibility. Correctness is asserted with respect to a given specification, which may be related to safety or liveness [36, 37]. A safety specification concerns a set of undesirable (reachable) states that must be avoided or simply ensures that "something bad never happens", while a liveness one refers to a desired (reachable) state that will eventually be achieved or simply ensures that "something good will eventually happen". In both cases, a model is needed for ECPS, in order to ensure that no error state is reachable or that a desirable one is eventually reached. Therefore, an algorithmic (decision) procedure can be devised from a formal model, in order to decide about its correctness and reliability; this decision procedure output is simply "yes", if a specification holds, or, otherwise, "no" [38]. A general ECPS model is then defined as follows.

Definition 1. *ECPS S are tuples $S = (X, X_0, U, Y, r, v)$, where X is a set of reachable states, X_0 is a set of initial states, U is a set of inputs, Y is a set of outputs, r is a transition function, such that $r : X \times U \to X$, and v is an output mapping function, such that $v : X \times U \to Y$ [39–41].*

2.2 Formal Modeling for ECPS Verification and Synthesis

This general ECPS model can be used to synthesize ECPS; in particular, synthesizing reliable ECPS is related to designing controllers that are able to handle complex system dynamics. One may notice that the so-called ECPS can also be considered HSs, since both types present a mixture of continuous and discrete dynamics. Thus, both formal verification and control synthesis for ECPS are based on hybrid control system theory, which is still under development, since control theories for continuous and discrete systems were developed independently [42]. As a consequence, formal ECPS synthesis includes but is not limited to a control synthesis problem and must consider both formal verification of its correctness and reliability. Besides, formal verification and control synthesis for ECPS is usually based on a model denoted as hybrid automata (HA), which follows Definition 1.

The overall automated formal-synthesis process for ECPS involves three main steps: *modeling*, which is related to obtaining a sound mathematical representation (usually HA) to capture all relevant aspects of continuous and discrete dynamics and their interaction, as well as formulate properties and invariants that describe a desirable behavior; *verification*, where ECPS are checked against a trusted mathematical model and compliance with a desired behavior is verified; and *synthesis*, where a controller is (automatically) produced or invariant control laws that allow ECPS to meet requirements and behave as desired are found, which can then be confirmed by a verification oracle.

Generally speaking, formal models are largely employed in science and engineering areas, in order to analyze, predict, and modify the behavior of real-world systems. Specifically, deterministic and stochastic models are the basis of many recent engineering advances: the former is uniquely determined by parameter values and previous states, while the latter presents inherent randomness, which is described by probability distributions, *i.e.*, random variables whose behavior is described by probability density functions [43]. Nonetheless, one can argue that such models are not enough to represent ECPS, since interactions among different entities (software and hardware components) present several events that could not be deterministically or stochastically modeled, *i.e.*, they should be better represented by non-deterministic approaches [5]. Indeed, that is even more evident when unknown properties are addressed, such as internet traffic that is dependent on human behavior. Indeed, probability relies on likelihood and non-determinism on possibility.

On the one hand, early representations of hybrid systems, *i.e.*, systems that mix continuous and discrete dynamics, consisted in instantiating discrete components and discretized continuous dynamics, by using discrete models and considering non-modeled effects as uncertainties or perturbations exogenous to them, which can even be negligible. On the other hand, that particular approach is unsuitable for complex ECPS, which present intense non-determinism, where uncertainties play a central role and might invalidate an entire system behavior. Nonetheless, state-of-the-art verification and synthesis methodologies for ECPS take advantage of the entire modeling framework for hybrid systems; in particular, hybrid automata and their variants. Indeed, ECPS have been modeled through different structures that present different degrees of non-determinism and balance between continuous and discrete dynamics, such as timed automata [44], hybrid automata [45, 46], and stochastic hybrid automata [47, 48].

Another useful formalism for ECPS is timed automata, which targets real-time systems and employs finite automata with clocks, *i.e.*, real variables that follow continuous trajectories (behaviors) with constant slope. Hybrid automata bridges digital processing and analog entities and consequently generalizes timed automata, by also employing finite automata with real variables, but whose trajectories follow more general dynamical laws [46]. Timed automata and hybrid automata are deterministic models for hybrid systems that built the basis of HS representations, which were subdivided into different classes of hybrid automata, depending on mapping functions (r and v), *e.g.*, linear hybrid automata [49], rectangular hybrid automata [46, 50], nonlinear hybrid automata [51, 52], integration graphs [53], and linear stochastic hybrid automata [47]. Recently, Rungger and Tabuada [39] and Tabuada *et al.* [54] proposed a general and wide model for ECPS, which is based on hybrid automata, by providing the concept of robustness and important tools for robust stability analysis. In addition to HA- and TA-based models, there are a number of other ECPS modeling initiatives based on (exact or approximate) bi-simulation [41] and simulation [55], as well as interval analysis [56].

Another approach to ECPS modeling tackles quantized control systems and finite-word length (FWL) effects on the performance of those systems. Differently from bi-simulation, quantized models are built from a countable subset of the (quantized) input space [57]. In particular, Keel and Bhattacharyya showed that even robust and optimal controllers may still present an undesired property [58, 59]:

fragility or sensitivity to implementation aspects (*e.g.*, FWL representations of digital controllers). Therefore, various studies tried to connect sampled-data theory to hybrid-system models [60–63].

In addition to ECPS-dynamics modeling, it is often interesting to establish invariants on ECPS's state-spaces, which can be used for accelerating formal verification and synthesis processes regarding those systems. Such invariants might be synthesized, in order to point out state-space boundaries [64], and could determine ECPS stability [65] or reachability [56, 66]. Finally, it is worth mentioning that there are still some recent modeling approaches that are mixtures of stochastic and non-deterministic models for ECPS, *e.g.*, stochastic non-deterministic automata [67] and piecewise-deterministic Markov chains [47].

2.3 Illustrative Example of ECPS Model

In order to illustrate the general modeling process of ECPS, by means of HAs, a three-tanks system, known as Sim3Tanks [68], is considered here. Such a three-tanks system consists of three cylindrical tanks connected by four pipes, which allow fluid exchange between the lateral tanks (tanks 1 and 2) and the central one (tank 3), in both directions, as illustrated in Fig. 2, while Fig. 3 shows its corresponding HA. The upper pipes and valves that connect tanks 1 and 2 to tank 3 are located at the same height h_0 and are called transmission pipes and valves, respectively. The lower pipes and valves that connect tanks 1 and 2 to tank 3, in turn, are aligned with their base and are called connection pipes and valves, respectively. At the bottom of each tank, there are output pipes and valves; the dashed arrows illustrated in Fig. 2 indicate the reference direction of each flow.

The state-space X of Sim3Tanks is defined by the state vector $x = [h_1 \ h_2 \ h_3]^T$ and the initial state space X_0 is assumed to be levels lower than the height of the transmission valves, *i.e.*, $h_1, h_2, h_3 < h_0$. The flow rates from pumps P_1 and P_2 (*i.e.*, Q_{P_1} and Q_{P_2}) are inputs that define the input space U, with $u = [Q_{\mathrm{P}_1} \ Q_{\mathrm{P}_2}]^T$. The flows through transmission, connection, and output pipes can be determined, respectively, by

$$Q_v = K_v \beta \operatorname{sgn}(\Delta h_{v,k}) \sqrt{|\Delta h_{v,k}|}, \quad \text{with} \quad v = a, b, \tag{1}$$

$$Q_{i3} = K_{i3} \beta \operatorname{sgn}(h_i - h_3) \sqrt{|h_i - h_3|}, \quad \text{with} \quad i = 1, 2, \tag{2}$$

and

$$Q_j = K_j \beta \sqrt{h_j}, \quad \text{with} \quad j = 1, 2, 3, \tag{3}$$

where

$$\Delta h_{a,1} = \Delta h_{b,1} = \Delta h_{a,3} = \Delta h_{b,5} = 0, \tag{4}$$

$$\Delta h_{a,2} = \Delta h_{b,2} = \Delta h_{a,4} = \Delta h_{b,6} = h_0 - h_3, \tag{5}$$

$$\Delta h_{a,5} = \Delta h_{a,7} = h_1 - h_0, \tag{6}$$

$$\Delta h_{a,6} = \Delta h_{a,8} = h_1 - h_3, \tag{7}$$

$$\Delta h_{b,3} = \Delta h_{b,7} = h_2 - h_0, \tag{8}$$

and

$$\Delta h_{b,4} = \Delta h_{b,8} = h_2 - h_3. \tag{9}$$

K_v, K_{i3}, and K_j correspond to the state of the flow control valves ($K_v, K_{i3}, K_j \in [0, 1]$) and $\beta = \mu S \sqrt{2g}$, where μ is the flow

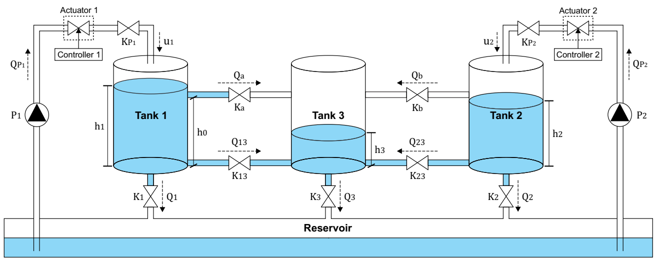


Fig. 2: Sim3Tanks

correction term, S is the pipe cross-sectional area, and g is the gravity acceleration constant. The transition function $r : X \times U \to X$ is a mix of continuous-time transitions, represented by ordinary differential equations of type $\dot{x} = r_k(x, u)$, for $k = 1, \ldots, 8$, which are defined for each control mode i and discrete-time transitions relations related to actions, such as "level up" and "level down". In particular, the continuous-time transitions are described as

$$\begin{aligned} \dot{x} = {} & r_k(x, u) \\ = {} & \begin{bmatrix} \frac{1}{S_c} K_{\mathrm{P}_1} Q_{\mathrm{P}_1} - \frac{1}{S_c}(Q_a + Q_{13} + Q_1) \\ \frac{1}{S_c} K_{\mathrm{P}_2} Q_{\mathrm{P}_2} - \frac{1}{S_c}(Q_b + Q_{23} + Q_2) \\ \frac{1}{S_c}(Q_{13} + Q_{23} + Q_a + Q_b) - \frac{1}{S_c} Q_3 \end{bmatrix}. \end{aligned} \tag{10}$$

The output function $v : X \times U \to Y$ is related to the sensors available in Sim3Tanks. At the top of each tank, there is an ultrasonic sensor responsible for measuring the liquid level inside it and, in all pipes, there is a flow control valve (proportional or on/off) and a Hall effect flow sensor. As a result, this system has a total of twelve sensors, *i.e.*, three level sensors (one per tank) and nine flow sensors (one per valve). Thus, its output function is defined, for all control modes, as

$$y = v(x, u) = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ 0 \\ 0 \\ Q_a \\ Q_b \\ Q_{13} \\ Q_{23} \\ Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_{\mathrm{P}_1} & 0 \\ 0 & K_{\mathrm{P}_2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u. \tag{11}$$

One may suppose that there is a safety condition that forbids the liquid level of some tank to be greater than a maximum level $\bar{h}$, in order to avoid spill or structure damage. By assuming that the controlled variable of Sim3Tanks is the output flow rate Q_3, with $K_1 = K_2 = 0$, it is desirable that this particular variable tracks a reference signal z_{ref}. The first condition is a safety specification, *i.e.*, some state ($h_1 > \bar{h}$ or $h_2 > \bar{h}$ or $h_3 > \bar{h}$) that must be unreachable, and the second one is a liveness specification, where the desirable state ($Q_3 = z_{\mathrm{ref}}$) is eventually reached. Such properties can then be modeled through the computation tree logic (CTL) formulae ϕ_{safety} and ϕ_{liveness}, respectively, as follows:

$$\phi_{\mathrm{safety}} \models \mathbf{AG} \neg(\phi_1 \vee \phi_2 \vee \phi_3) \tag{12}$$

and

$$\phi_{\mathrm{liveness}} \models \mathbf{AF}\ \mathbf{AG}\ \phi_4, \tag{13}$$

where propositional variables $\phi_1, \phi_2, \phi_3, and\ \phi_4$ can be represented as

$$\phi_1 \leftrightarrow h_1 > \bar{h},$$

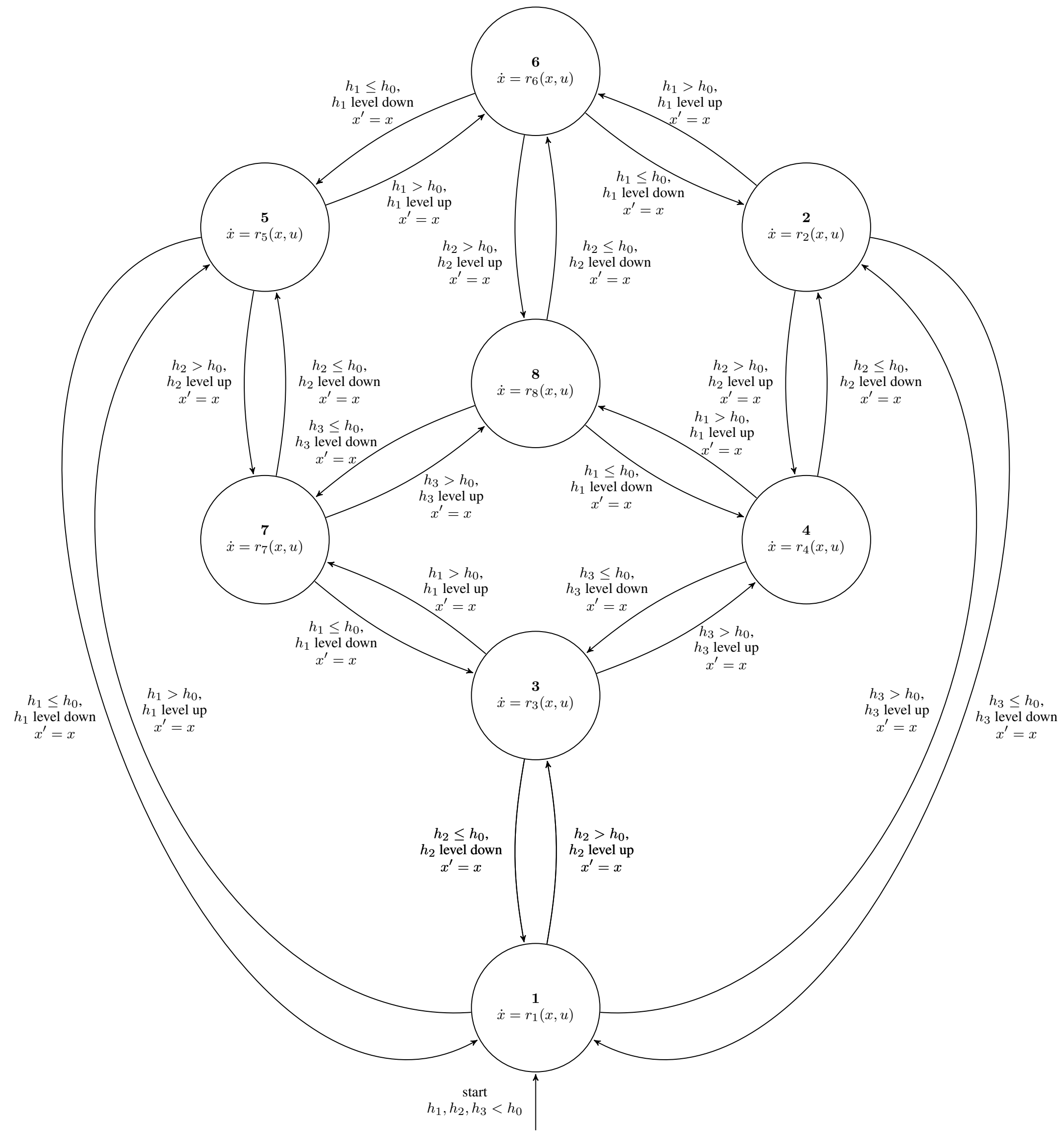
$$\phi_2 \leftrightarrow h_2 > \bar{h},$$

$$\phi_3 \leftrightarrow h_3 > \bar{h},$$

and

$$\phi_4 \leftrightarrow Q_3 = z_{\mathrm{ref}}.$$

In summary, ϕ_{safety} means that, along all paths, $\neg(\phi_1 \vee \phi_2 \vee \phi_3)$ must hold globally, while ϕ_{liveness} means that, along all paths and at some moment in the future, ϕ_4 becomes true and remains that way thereafter. Properties ϕ_{safety} and ϕ_{liveness} could also be



6 $\dot{x} = r_6(x,u)$
5 $\dot{x} = r_5(x,u)$
2 $\dot{x} = r_2(x,u)$
8 $\dot{x} = r_8(x,u)$
7 $\dot{x} = r_7(x,u)$
4 $\dot{x} = r_4(x,u)$
3 $\dot{x} = r_3(x,u)$
1 $\dot{x} = r_1(x,u)$
$h_1 \leq h_0$, h_1 level down $x' = x$
$h_1 > h_0$, h_1 level up $x' = x$
$h_2 > h_0$, h_2 level up $x' = x$
$h_2 \leq h_0$, h_2 level down $x' = x$
$h_3 \leq h_0$, h_3 level down $x' = x$
$h_3 > h_0$, h_3 level up $x' = x$
start $h_1, h_2, h_3 < h_0$

Fig. 3: Hybrid automata modeling of Sim3Tanks.

modeled through other temporal logic, such as linear-time temporal logic (LTL) as follows:

$$\phi_{\text{safety}} \models \mathbf{G}\neg(\phi_1 \vee \phi_2 \vee \phi_3) \qquad (14)$$

and

$$\phi_{\text{liveness}} \models \mathbf{G}\,\mathbf{F}\,\phi_4. \qquad (15)$$

Based on this kind of approach and supposing the system behavior of Sim3Tanks is captured in a model and taken into account, this system could then be checked, through a suitable methodology.

3 Symbolic Verification of Cyber-Physical Systems

3.1 Existing Symbolic Verification Methodologies for Cyber-Physical Systems

ECPS are complex entities composed by computing systems that interact with each other and their surrounding environment, through data exchange; the latter usually being converted into different domains (analog and digital). As a consequence, there is a compound behavior, which is not exactly a composition of individual elements, but instead a distinct collection of effects arising from the mentioned arrangements and interactions. Aggregate effects influence system actions, resource allocation, and fragility aspects, which then require methodologies for mitigating or taking advantage of them, thus depending on specific scenarios. In that respect, formal verification is an interesting approach, where a system can be analyzed for determining whether it meets design requirements.

Aggregate effects can be observed in a wide variety of ECPS, such as medical devices, autonomous vehicles, and smart grids [69]. For instance, controllers that do not take into account aggregate effects have the potential to cause serious problems, such as automatic medical infusion systems that do not consider continuous drug administration to human bodies [69], which present modified behavior caused by chemical compounds; and practical implementations of digital controllers for UAV attitude systems, if those entities do not take into account FWL effects, which then causes UAV crash and malfunctioning, during missions [70]. Such an effect could also be noticed in system Sim3Tanks presented in the preceding section, that being in the implementation of a controller for Q_3, which can indeed be affected by FWL effects, or fluid density and viscosity modifications, which are able to influence pump performance and flow rates.

Based on what was exposed in the last paragraphs, it may be not enough to verify ECPS implementations, from the point of view of a designed hardware or software model, in such a way that composing elements, being those constructive gates or instructions in a given programming language, are correct, but instead assure that their behaviors were taken into account, even when influenced by noise or exogenous conditions. In addition, aggregate behaviors should also be considered, in such a way that interaction effects are not disregarded, which may result in critical problems. Nonetheless, behavior models may be difficult to develop, given their stochastic nature, and model checking may become very complex to perform. Finally, safety in ECPS may not be achievable, in some scenarios, due to limitations of practical components, which is related to error conditions and signal delays.

ECPS correctness can be verified through symbolic model checking [21], which ensures if a property holds for all reachable states. SMV [71] and NuSMV [72] are two representative symbolic verifiers based on binary decision diagrams (BDDs); actually, the latter is a re-implementation and extension of SMV, which also supports BMC using SAT solvers. In NuSMV, system properties can be expressed in a wide range of temporal properties, such as CTL, as done for system Sim3Tanks in Section 2.3, real-time CTL, LTL, and property specification language (PSL). SMV and NuSMV also construct counterexamples, whether a property violation is found; however, given that BDDs based verifiers can be unfeasible regarding system resources, BMC using SAT or SMT solvers may be considered a viable option. For instance, Chaves *et al.* [70] explored that behavior-modeling approach and developed models for digital controllers implemented with fixed-point arithmetic, where effects, such as overflow and limit-cycle oscillations, were checked, which resulted in correct operation or guidelines for system redesign. As already mentioned, Sim3Tanks could benefit from such an approach, regarding its target controllers, given that their practical implementations are usually developed in a representation format and then performed in another one, which is not taken into account in a compound structure. In summary, the mentioned effects were transformed into properties to be checked by a BMC tool, whose operation might result in a counterexample. Another viable solution is non-linear hybrid automata in SMT-based verification [73], where quantifier-free theories are reduced, at the cost of additional variables.

Another interesting approach for ECPS is deductive verification via theorem proving, which has the potential to be faster, given that it takes into account specific inference rules w.r.t. system constraints for efficient state-space exploration [74]. Platzer [75] tackles some interesting topics regarding theorem proving applied to ECPS, from which we can highlight multi-dynamical systems, which represent complex systems as combinations of multiple elementary dynamical features. Given that ECPS rely on decisions evaluated by computer control, they are discrete; however, since they interact with physical processes, they are also continuous, while dealing with scenarios characterized by uncertainty. As a consequence, ECPS verification involves many dynamical aspects, and one example exploiting them is KeYmaera [76], which uses a combination of automated theorem proving, real quantifier elimination, and symbolic computations for ECPS verification. In addition, Sanwal and Hasan [77] proposed high-order-logic theorem proving for ECPS continuous-aspects analysis. Indeed, continuous parts of ECPS can be modeled, while theorem proving ensures correctness.It is worth noticing that such an approach would be very suitable to Sim3Tanks (see Section 2.3), which could be used for modeling interaction with the fluid in its reservoir.

Simulation is another tool for extracting behavior models regarding ECPS, although they do not consider interaction with continuous systems. One notable example is the wireless cyber-physical simulator (WCPS) [78], which is based on a federate architecture combining realistic simulations of both wireless sensor networks and structures, with focus on civil structural analysis. Canadas *et al.* [79], in turn, used time automata for modeling ECPS to be used during simulation, where global models are systematically created for plant and controller; here, software tools can then be used for validation, such as UPPAAL [80]. Junjie *et al.* [81] proposed a unified approach for ECPS modeling, with modelica, as a basis for system verification. Indeed, the use of a unified modeling language for ECPS is very appealing; however, behavior model of whole systems should be carefully performed, while considering environmental influences and aggregate effects. Finally, simulation approaches depend heavily on models, which can be done through automatic abstraction. Thacker *et al.* [82] worked on a methodology based on automatic abstraction, where models are described through a language inspired by assembly code that unifies hardware, software, and environmental aspects, in a single formalism.

Symbolic execution is considered a high-coverage testing method, given that it ensures addressing of reachable paths, with properties described as assertions. In summary, variables are represented as symbols that are modified during program execution, while they are evaluated with assertions. Ishigooka *et al.* [83] proposed a framework with model generator, which provides models and checks them through symbolic execution. A careful definition regarding properties to be checked is required, by taking into account interaction between actuator and plant. Radojicic *et al.* [84], in turn, take into account uncertainty, through affine arithmetic decision diagrams, during symbolic execution of ECPS. In addition, computation with uncertainty is separated from simulation, which makes it possible to integrate symbolic execution into pre-existing frameworks. Finally, Majumdar *et al.* [85] introduced a closed-loop symbolic execution tool for ECPS, named as closed-loop symbolic execution (CLSE). Its inputs are a differential equation for plant and a software implementation for controller, whose combination is executed up to a horizon time. Due to those aspects, this one would be a good option for verifying Sim3Tanks, after some few modifications, given that its continuous-time transitions are already described by differential equations. Nonetheless, those authors noticed scalability problems, when dealing with typical control systems, which could be solved through combination with abstract interpretation.

We could not conclude without mentioning compositional verification, which consists in extracting aspects that are relevant to a given system, by analyzing its individual components. In such a context, Borda *et al.* [86] proposed a language for modeling self-adaptive ECPS and a technique for composition verification of those systems, where specific aspects affecting a requirement are identified and adaptation procedures are devised for them, in order to preserve behavior, in face of environment changes. In summary, such adaptation procedures can be confined to specific system components, where formal verification can check individual behavior and ensures requirement satisfaction, in a restricted state space (of a component).

Finally, as stated at the beginning of this section, aggregate behaviors are of paramount importance, in such a way that unforeseen conditions, due to interaction of internal components, are not disregarded, which can cause serious damages to ECPS. For instance, Brings [87] proposed a methodology based on automated generation of models representing the different possible system-networks a cyber-physical system may interact with, in order to identify relevant behavior aspects and provide automated verification. Each aggregate model is checked against a system specification, with the goal of providing proper operation for aggregate behaviors.

3.2 *Verification for Specific Domains*

In the previous section, we tackled ECPS verification focusing on techniques, with a broad summary regarding the most important results. Nonetheless, specific domains present distinct challenges and employ special algorithms, with approaches tuned for unique goals. For instance, Kang *et al.* [88] proposed a methodology for performing verification and validation of automotive ECPS, regarding non-functional properties of autonomous vehicles, through Simulink design verifier and constraints translated into UPPAAL-SMC [89] models, during design phases. In addition, it is worth noticing that the proposed methodology supports continuous dynamic behavior models for plants, in Simulink.

Medical ECPS are another interesting domain, which constitute critical systems whose main goal is to keep patients' safety. As a consequence, verification of those ECPS are strictly necessary, in order to provide reliable systems. In that spirit, Silva *et al.* [90] proposed a model-based approach focusing on re-usability and productivity, which allows users build aggregate models, based on patients (clinical databases) and medical devices (technical specifications), and simulate them, in order to identify undesirable behavior.

There is also an interesting approach towards micro-grid ECPS [91], which is based on a test platform. It has three major components for validation: real-time model of a distribution feeder, network simulator in-the-loop model, and physical hardware. As an experimental validation of the proposed strategy, a threshold at which a micro-grid controller could break down was presented. Finally, cyber-security aspects could also be incorporated, in addition to large power systems, with the goal of preserving system's availability, integrity, and confidentiality.

Finally, Klein *et al.* [92] presented formal verification approaches for ensuring safety and security, in UAVs. In particular, they described an attack by rogue camera software and a virus delivered through a compromised universal-serial-bus stick, for the Boeing Unmanned Little Bird H-6U. Those authors emphasized the need for formal methods regarding vulnerability analysis (software connected with hardware), remote accessibility (device authentication and access control), and patch management (vendors might be long gone).

4 Discussion Regarding Formal Verification of Cyber-Physical Systems and Future Trends

The ECPS-verification problem can be tackled through formal methods as based on decidability, which is stated according to Definition 2.

Definition 2. *Given a class of ECPS Ω and a set Φ of desired properties, a class of verification problems γ is called decidable if an algorithmic procedure can decide whether there exists a finite number of steps $\mathcal{O} \in \Omega$ that satisfies every property $\mathcal{L} \in \Phi$ [40].*

Decidability can be regarded as a key issue in ECPS verification, since there are uncountable states and complex dynamics to be considered. In particular, ECPS safety verification is undecidable, except in very specific cases, as in systems that can be represented by timed and rectangular automata [93]. An alternative to ensure that ECPS trajectories will not pass through unsafe states is the use of barrier certificates [94, 95]: one example is stability, which ensures state convergence to an equilibrium point and consequently means that there is a region where any trajectory remains inside it. Stability can be ensured based on Lyapunov theory, for a wide class of model abstractions [27], or linear system stability theory, for discrete simulations [96].

Generally, properties that must hold for ECPS are related to safety/reachability and liveness, and more recently, security requirements such as availability (services are accessible if requested by authorized users), integrity (data completeness and accuracy are preserved), and confidentiality (only authorized users can get access to a system's data). The earliest studies on HSs and ECPS proposed approaches based on computational tree logic and linear temporal logic [45], in order to specify properties and verify those systems. A glimpse regarding that, in a unified manner, can be noticed in Section 2.3, through system Sim3Tanks. In the last decade, signal temporal logic [97, 98] was proposed and have been used since then for this purpose, thus providing more suitable operators for dynamic systems and signal processing. Therefore, model-checking procedures have been employed to verify those properties, even under undecidability conditions via approximations, *e.g.*, bounding the number of steps from initial states, which leads to BMC [99].

Along with CPS modeling and verification techniques purely based on hybrid automata, one can also highlight logic-based approaches [100], which may still use hybrid automata, but which are not based on a algebraic description of states and transitions. Some examples of logic-based verification are theorem proving techniques, *e.g.*, deductive verification (*i.e.*, interactive theorem proving) [101] and automatic theorem proving [74], and reduction-based techniques, among which BMC stands out [102].

Recently, some studies have addressed ECPS formal verification considering discretization, switching, and quantization effects. Dugirala and Viswanathan [103] verified embedded control software considering real-time issues, *e.g.*, delayed responses and possible loss of real-time deadlines. Anta *et al.* [104] presented a tool called Costan, which finds errors in mathematical-model implementations and verifies whether those are tolerated, considering quantization effects and fixed-point implementation, while focusing on quantization error effects over system stability. Similarly, Ismail *et al.* [105] presented DSVerifier, a BMC tool for digital systems that employs SMT solvers as back-ends, in order to provide support for digital system design and verification considering FWL effects. In particular, DSVerifier checks if a designed digital controller and/or closed-loop ECPS presents a desired performance, when it is implemented with a given FWL format [96, 106].

Finally, it is worth mentioning that there are other alternatives for ECPS modeling and analysis, which are not based on verification, but instead on simulators [81, 107, 108], as already exposed. Although such approaches generally fail in ensuring that some properties hold, they present good scalability, which is ideal for large scale and complex systems that are hardly modeled through automata- or logic-based approaches. As pointed out in Section 3, Modelica [81] and Simulink/MATLAB [108] are important examples of widely used simulators for ECPS. In order to achieve a balanced trade-off between ECPS-simulation scalability and correctness of verification methods, statistical model checking has been employed to formally verify ECPS [21].

In addition, Shoukry *et al.* [109] tackled another important problem related to ECPS: *security*. Indeed, the essence of ECPS claims for sophisticated interfacing (*e.g.*, sensors and network connections), which normally leads to security issues [110], given that wrong control actions can be generated through corrupted measurements. The mentioned authors proposed a methodology for estimating physical-system state, through formal methods, even if sensor inputs are contaminated with erroneous or intentionally corrupted data, in order to still be able to execute suitable control commands. As a result, those authors focused their effort on ensuring system's availability, integrity, and confidentiality. In this respect, the Science of Sensor Systems Software (S4) project[*] brings together researchers to develop new methods, algorithms and tools for sensor system software, which allows one to increase the system's reliability of the ever-expanding networks of sensors, in ECPS.

Choo *et al.* [111] also raised new challenges regarding ECPS, mainly focused on privacy and security, which must operate in sensitive environments. Towards that, Illiano, Muños-Gonzáles, and Lupu [112] proposed a wavelet-based approach that is able to identify malicious data injection (providing counteract), in wireless sensor networks, and distinguish them from faulty behaviors. Cyber-security is also a concern of Fiore *et al.*, who aim to ensure secure-state estimation of UAV systems under sparse attacks [113]. Indeed, many situations can be regarded as satisfiability problems and solved, for instance, with approaches based on SMT [114–116], which indicate many links with other research areas and might lead to a unified approach regarding system verification, security enforcement, and interaction handling.

In summary, one may notice from what was explained that ECPS present high verification-complexity, due to their size and intricacy. As a consequence, the chosen techniques should aim for coverage, while minimizing computational efforts, and also consider aggregate effects, as a means to systemic evaluation. Again, Sim3Tanks, in Section 2.3, is an elucidating example, given that its structure can be easily expanded, when dealing with professional or industrial systems, which would soon raise scalability concerns, and its aggregate effects regarding interaction with stored fluid can compromise the entire system operation and must be taken into account during its

[*] `http://www.dcs.gla.ac.uk/research/S4/`

verification. In that sense, one should focus on accurate system models allied to restricted state space exploration, which could be done, for instance, through symbolic verification and its companion techniques (*e.g.*, abstract interpretation). The next sections address them, in an attempt to devise new paths for ECPS verification.

4.1 Bounded Model Checking (BMC)

Bounded Model Checking (BMC) based on SAT was originally proposed to verify hardware designs and then alleviate the state-space explosion problem by BDD-based procedures [102, 117]. 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 standard SAT solvers. BMC based on SMT [118], in turn, was originally proposed to deal with increasing software verification complexity [119]. In general, BMC techniques aim to check violations of a given (safety) property at a given system depth, as shown in Fig. 4.

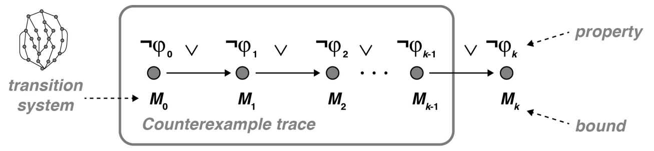


Fig. 4: Bounded Model Checking (BMC).

In order to formulate the BMC problem, let M be an abstract machine that represents a state transition system according to Definition 3.

Definition 3. *A state transition system, denoted by M, is defined by a triple (S, R, S_0), where S represents the set of states, $R \subseteq S \times S$ represents the set of transitions (i.e., pairs of states specifying how the system can move from state to state), and $S_0 \subseteq S$ represents the set of initial states.*

Indeed, given a transition system *M*, which is derived from the control-flow graph of a program, a property ϕ, which represents program correctness and/or a system's behavior, and an iteration bound *k*, which limits loop unrolling, data structures, and context-switches, BMC techniques thus unfold a system *k* times, in order to convert it into a verification condition ψ, which is 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*. The propositional problem associated with SAT-based BMC is formulated by constructing [117]

$$\psi_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \neg\phi_k. \tag{16}$$

Here, ϕ_k represents a safety property ϕ in step *k*, *I* is the set of initial states of M, and $R(s_i, s_{i+1})$ is the transition relation of M at time steps i and $i+1$. Hence, the equation $\bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$ represents the set of all executions of M of length k and $\neg\phi_k$ means the condition that shows ϕ is violated in state k, which is reached by a bounded execution of M of length k. Finally, the resulting (bit-vector) equation is translated into conjunctive normal form, in linear time, and passed to a SAT solver (if (16) is encoded into propositional logic) or SMT solver (if (16) is encoded into a decidable-fragment of first-order logic) for checking satisfiability. One may notice that Eq. (16) can be used to check safety properties [120]. Liveness properties (*e.g.*, starvation and deadlock) that contain the LTL operator F are checked by encoding $\neg\phi_k$ in a loop, within a bounded execution of length at most k, such that ϕ is violated in any state in a loop [102, 121]. In that case, Eq. 16 can be rewritten as

$$\psi_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \left(\bigvee_{i=0}^{k} \neg\phi_i \right), \tag{17}$$

where ϕ_i is the propositional variable ϕ at time step i. Thus, this equation can be satisfied if and only if, for some i ($i \leq k$), there exists a reachable state at time step i in which ϕ is violated. Indeed, Eq. (17) differs slightly from (16), because it represents a violation of length k or less to the considered and safety property, while (16) represents a violation of exactly length k. It means that if a system deadlocks in $l \leq k$ steps and the respective cause is at step $j \leq l$, then formula (16) turns out to be unsatisfiable and therefore it will not detect such an error.

One may notice that BMC analyzes only bounded program runs, but generates verification conditions (VCs) that reflect the exact path in which a statement is executed, the context in which a given function is called, and the bit-accurate representation of expressions. A verification condition is a logical formula (constructed from a bounded program and desired correctness properties), whose validity implies that a program's behavior agrees with its specification [22]. Correctness properties, in programs, can be specified by users via *assert* statements or automatically generated from a specification language [122]. If all of a bounded program's VCs are valid, then it is in compliance with its specification, up to a given bound.

As an example, one may consider a simple C program (slightly modified from [123]) with an exponential number of paths, as shown in Fig. 5(a). The corresponding C program, in single static assignment (SSA) form [124], is shown in Fig. 5(b). The SSA form is an intermediate representation, which is used by compilers to facilitate optimizations and transformations of source code. The common property in SSA form is that every variable state has only one definition in a program text, which is achieved by introducing a fresh variable from an original name (*e.g.*, with a subscript), at every assignment, such that there is an unique left-hand side for each new state, as shown in Fig. 5(b).

Apart from that, this program has an exponential number of paths, since each element of x can be either greater than one or less than or equal to one. Despite the large number of paths through that program, BMC unwinds it up to a bound k and translates it into a VC ψ, such that ψ is satisfiable if and only if the assertion $a <= N$ fails. One may also notice that BMC still encodes program stages with a size that grows linearly with N. More precisely, the program in Fig. 5(a) is converted into ψ, through a decidable fragment of first-order logic [22], as

$$\psi := \begin{bmatrix} a_1 = N \\ \wedge\ a_2 = ite\,(x[0] > 1, a_1 - 1, a_1) \\ \wedge\ a_3 = ite\,(x[1] > 1, a_2 - 1, a_2) \\ \wedge\ a_4 = ite\,(x[2] > 1, a_3 - 1, a_3) \\ \wedge \ldots \\ \wedge\ a_{N+1} = ite\,(x[N-1] > 1, a_N - 1, a_N) \\ \wedge \neg\,(a_{N+1} \leq N) \end{bmatrix}. \tag{18}$$

The ternary operator $f \;?\; t_1 : t_2$, shown in Fig. 5(b), is converted into the conditional expression $ite(f, t_1, t_2)$ that takes as its first argument the Boolean formula f and, depending on its value, selects either the second (i.e., t_1) or the third argument (i.e., t_2). In order to verify that the assertion $a <= N$ holds, its negation is added to ψ and a check is performed, regarding whether the entire formula is satisfiable, through an off-the-self SMT solver. Formula (18) can be represented simply as a Boolean logic circuit, which can be further transformed into a (equisatisfiable) conjunctive-normal-form formula over propositional variables, by Tseitin's transform [125], in linear time, and also by introducing at most a linear number of fresh variables; however, checking validity of a first-order logic formula, in a given background theory, is an $\mathcal{NP}$-complete problem [126].

From the practical point of view, SAT- 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

```
#include <assert.h>
int x[N], a;
...
int main(void) {
  a=N;
  for(int i=0; i<N; i++){
    if (x[i]>1){
      a--;
      assert(a<=N);
      return 0;
    }
  }
}
```

(a)

```
a1 = N
a2 = (x[0] > 1) ? a1 - 1 : a1
a3 = (x[1] > 1) ? a2 - 1 : a2
a4 = (x[2] > 1) ? a3 - 1 : a3
...
an+1 = (x[N-1] > 1) ? an - 1 : an
```

(b)

Fig. 5: (a) A simple C program with a loop *for*. (b) The corresponding unwound C program of (a) converted into SSA form.

reported in the available literature [127–131]. 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 depths.

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. Indeed, such an issue has pushed researchers to overcome the depth limitation and propose extensions capable of proving global correctness. In particular, two possible strategies can be adopted, in order to prove properties through traditional BMC: *(i)* compute the *completeness threshold*, which can be smaller than or equal to the maximum number of loop-iterations occurring in a program, or *(ii)* determine the high-level worst-case execution time (WCET), which also gives a bound on the maximum number of loop-iterations [102, 132, 133]. Nonetheless, in practice, complex software systems involve large data-paths and complex expressions. Therefore, verification conditions that arise from BMC of programs become harder to solve and require substantial amounts of memory to build. Finally, another strategy, which is based on complementary analyzes beyond basic BMC, consists in using a companion technique, with the potential to fill the gap between the adopted depth and all subsequent ones, such as induction, which is explained below [134, 135].

4.2 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 [134, 135]; however, the main challenge regarding such an approach relies on computing and strengthening inductive invariants from programs. In particular, loop invariants, which are computed from programs under verification, must be inductive (and not just invariant), in order to check the corresponding VCs, *i.e.*, invariance cannot determine induction of a non-inductive assertion [22].

```
#include <assert.h>
int main(void) {
  float x=2;
  while (*) {
    x = ((2*x) - 1);
  }
  return 0;
}
```

Fig. 6: Motivating example for inductive invariants.

For instance, one may consider the C-code fragment shown in Fig. 6, where the star-notation indicates non-determinism. In addition, suppose that one wants to prove that $P : x > 0$ is invariant and, in order to do that, induction can be applied:

- It holds initially, because

$$\underbrace{x = 2}_{\text{initial condition}} \implies \underbrace{x > 0}_{\text{P}};$$

- Whenever P holds for k loop unwindings, it also holds for $k+1$ steps, given that

$$\underbrace{x > 0}_{\text{P}} \wedge \underbrace{x = 2 \wedge x' = 2 * x - 1}_{\text{transition relation}} \implies \underbrace{x' > 0}_{\text{P'}}.$$

If we consider the IEEE floating-point standard (IEEE 754) [136, 137], then invariant $x > 0$ initially holds; however, after 128 loop iterations (lines 4-6 of Fig. 6), an overflow occurs, whose numerical behavior is then determined by the standard IEEE 754. Nonetheless, this invariant is not inductive, given that $x > 0$ before an initial iteration does not ensure that $x > 0$ after each iteration. In particular, if we initially assign $x = 0.9$, then, after the fourth iteration, $x < 0$. As a consequence, even if invariant generation procedures successfully compute such assertions, which are indeed invariant, those must be inductive, so that *k*-induction verifiers can automatically prove correctness. In that specific example, an inductive invariant would be $x > 1$, given that if $x > 1$ before the initial iteration, then also $x > 1$ after *k* iterations.

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 [138, 139]. 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 [140].

Novel verification algorithms for proving correctness of (a large set of) C programs, by mathematical induction and in a completely automatic way (*i.e.*, users do not need to provide loop invariants), were recently proposed [140–144]. Additionally, *k*-induction based verification was also applied to ensure that (restricted) C programs (1) do not contain violations related to data races [145], considering the Cell BE processor, and (2) do respect time constraints, which are specified during system design phases [134]. Apart from that, *k*-induction is also a well-established technique in hardware verification, where it is easily applied, due to the monolithic transition relation present in such designs [134, 135, 146].

It is worth noticing that *k*-induction with invariants has the potential to be directly integrated into existing BMC approaches, given that the induction algorithm itself can be seen as an extension after *k* unwindings and it is possible to generate program invariants with other software modules, which are then translated and instrumented into an input program [144].

Nonetheless, there is little evidence, in the available literature, that model checking hardware and software systems through k-induction (and invariants) can be efficiently exploited in ECPS verification and synthesis. That happens due to the distinctive characteristics mentioned earlier, which influence ECPS developments and also verification processes. Additionally, there is still a lack of studies for software verifiers to exploit the combination and configuration of different invariant generation and strengthening algorithms, including analysis to discover linear inequalities, polynomial equalities and inequalities, and invariants related to memory and variable aliasing [22].

4.3 Property Directed Reachability (or IC3)

Bradley *et al.* introduced the "property directed reachability" (or IC3) procedure for the safety verification of systems [147, 148] and have shown that IC3 can scale on certain benchmarks, where k-induction fails to succeed. In particular, the success of IC3 over k-induction procedures is due to the ability of the former to guide the search for inductive instances with counterexamples regarding inductiveness (CTIs) of a given property [149].

One may consider again the C-code fragment shown in Fig. 6, where $P : x > 0$ is an invariant.

$$\underbrace{x > 0}_{\text{P}} \wedge \underbrace{x = * \wedge x' = 2 * x - 1}_{\text{transition relation}} \;\not\Rightarrow\; \underbrace{x > 0}_{\text{P'}}.$$

In this specific example, a CTI returned by an SMT solver is $x = 0$. If this state is not eliminated, then the invariant P cannot be established. The generated inductive assertion should establish that the CTI $x = 0$ is unreachable and if no such inductive assertion exists, then other CTIs can be examined instead (*e.g.*, 0.1, 0.2, ..., 0.9). As a result, the lemmas should be strong enough that consecutively revisiting a finite set of CTIs will eventually end up in an assertion, which is inductive relative to them, thus eliminating the CTI. In this example, instance $x > 1$ is inductive and eliminates all possible CTIs.

Jovanović *et al.* [150] presented a reformulation of IC3, by separating reachability checking from inductive reasoning. The authors further replace the regular induction algorithm by the k-induction one and show that it provides more concise invariants. Those authors implemented the mentioned algorithm in the SALLY model checker using Yices2, in order to perform a forward search, and MathSAT5, which executes a backward search. They showed that new algorithm is able to solve a number of real-world benchmarks, at least as fast as other approaches.

4.4 Craig Interpolation

Another feasible alternative to prove properties in BMC is to compute Craig interpolants for inconsistent pairs (or more generally, sets) of formulae [151, 152]. This alternative approach exploits the SAT/SMT solvers' ability to produce refutations, *i.e.*, proofs regarding the nonexistence of counterexamples of depth less than or equal to k, which do not ensure whether a given property holds, but contain information about reachable states of a model.

Definition 4. *Given a pair of formulae (A, B), such that $A \wedge B$ is inconsistent, and a proof by resolution for (A, B), an interpolant for (A, B) is a formula F with the following properties [151, 152]:*

- *$A \Rightarrow F$;*
- *$F \wedge B$ is unsatisfiable;*
- *F expressed only over the common variables (non-logical symbols) of A and B.*

As an example, consider $A = (x_1 \wedge x_2)$ and $B = (\neg x_2 \wedge x_3)$. Given that $(x_1 \wedge x_2)$ must imply F (or simply that $\neg x_1 \vee \neg x_2 \vee F$ hold) and $F \wedge \neg x_2 \wedge x_3$ must be unsatisfiable, one possible interpolant for the given pair of formulae (A, B) is $F = x_2$, since x_2 is a common part of both A and B.

The use of interpolants allows us to define a complete method for finite-state reachability analysis based on SAT and SMT solvers. In order to show how BMC and interpolation can be combined, we refer to Section 4.1, where we define Eq. (16) and the terms I, R, and ϕ. Now, suppose that $Q = I$ and Eq. (16) is partitioned, so that the set of initial states I and the first instance of the transition relation R are in set A, while the remaining instances of R and the property ϕ are in set B, as shown in Fig. 7 (note that k is unknown).

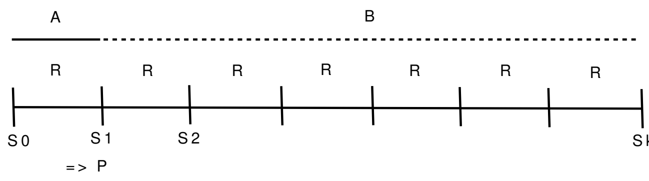


Fig. 7: Computing image by interpolation [151].

Suppose that we use an SMT solver to prove that the $A \wedge B$ is unsatisfiable, *i.e.*, we use an SMT solver to conclude that there is no satisfying assignment to $A \wedge B$.[*] The internal steps performed by SMT solvers for reaching this conclusion can be used to construct a proof of unsatisfiability Π, from which we can derive an interpolant F for the pair of formulae (A, B), *i.e.*, $F = \mathit{interpolant}\,(\Pi, A, B)$. According to Definition 4, A must imply F and since we defined A to be the set of initial states and the first instance of R (*i.e.*, from Fig. 7, $A = s_0 \wedge s_1$), it follows that F is *true* in every state reachable from the initial state, in one step. In other words, we can say that F is an over-approximation of the forward image of I [151, 152]. Also according to Definition 4, the formula $F \wedge B$ must be unsatisfiable (from Fig. 7, $B = s_2 \wedge s_3 \wedge \ldots \wedge s_k$), which means that there is no state satisfying F that can reach a final state s_k. After computing F, we then check whether F implies Q. If F implies Q, then no reachable state can satisfy the property ϕ and we can thus conclude that the property holds; however, since F is an approximation, we can falsely conclude that the final state is reachable. In this case, we update $Q = F \vee Q$ and $A = F \wedge R_0$, increase the value of $k + 1$, and check whether $A \wedge B$ is unsatisfiable. If $A \wedge B$ is satisfiable, we have found a valid counter-example (*i.e.*, a path from the initial state to the final one); otherwise, we compute the interpolant $F = \mathit{interpolant}\,(\Pi, A, B)$ again and check whether F implies Q. This procedure is stopped when we have found a valid counterexample or proved that the final state is not reachable (*i.e.*, the property holds). The details of the algorithm and further information about the use of interpolants, in model checking, can be found in [151, 152].

4.5 Abstraction

Apart from recent advances in automated symbolic verification, the number of functions implemented by current ECPS is growing considerably, which makes design verification using state-of-the-art symbolic verification techniques (*e.g.*, BMC, k-induction and IC3) more difficult. One possible alternative is to divide them in sets and attack isolated modules; however, inter-dependency also increases, which makes such an approach unfeasible. Another viable and more elegant strategy to scale those verification techniques is to combine them with some sort of abstraction, in order to remove irrelevant detail and reduce model size [153].

In this respect, predicate abstraction has been successfully used in software verification, by large organizations, *e.g.*, SLAM at Microsoft, which is a tool to automatically check device drivers for certain errors [154]. The key idea of predicate abstraction is to only track predicates on data and remove variables from models, with the

[*] *Note that if at any stage we can satisfy the property ϕ within k steps from the initial state, then we have found a counterexample.*

goal of pruning state spaces; however, predicate abstraction mostly works with control-flow dominated properties, which compromises ECPS verification. Another successful example of predicate abstraction is SATABS [155], which implements counterexample-guided abstraction refinement (CEGAR) based on SAT solvers, in order to verify single- and multi-threaded software with shared variables. In CEGAR, an initial abstract model is automatically produced from control structures of a program to be verified. Abstract models can admit erroneous (or "spurious") counterexamples, but every single counterexample is analyzed and its abstract model is correspondingly refined, in order to remove unintended behavior incorrectly added during an abstraction process.

Abstract interpretation is another useful formal analysis technique, which soundly approximates semantics of programs [156]. It can infer invariants about program behavior, by means of interval, which computes the maximum and minimum value of a given program variable (*e.g.*, $a < x < b$): octagons, which computes some relation among the program variables (*e.g.*, $x < a$, $x - y < b$), and convex polyhedral, which computes a full relation among the program variables (*e.g.*, $ax + by < c$). A number of tools exists to discover linear/polynomial relations among integer and real variables and then infer loop invariants [138, 139]. Additionally, the combination of invariant generation tools with symbolic model checking, for computing pre- and post-conditions, has also been reported in literature, with successful applications to verify safety properties in (embedded) software systems [140, 144, 157]. Astrée [158], PolySpace [159], and Frama-C [160] are the most successful implementations of the abstract interpretation technique; in particular, Astrée and PolySpace have been successfully used to prove absence of run-time errors, in real-world flight control software. Although those software verifiers scale relatively well for large code bases, their main drawback is that they usually provide false alarms, due to over-approximation of possible program executions.

4.6 Path-based Symbolic Execution

Path-based symbolic execution is another verification technique for addressing the complexity of checking large software systems with a particular focus on path exploration, *i.e.,*, all possible paths in a program are enumerated and individually checked [161]. On the one hand, path-based symbolic execution seems to be similar to BMC, since it will symbolically explore state-spaces, generate (path) constraints, and then check for satisfiability of program assertions. On the other hand, when a branch is found, while a BMC tool evaluates both branch sides (*i.e.*, true and false) and merges states after that branch, a symbolic executor primarily explores each branch, separately, thereby making a copy (*i.e.*, *forking*) of the current state. This process will happen for every branch in a program, until either an assertion violation is reached or all paths have been systematically explored. Similar to BMC, a model is generated, if a formula is satisfiable and a test case can be produced.

If the constraints for a given path are unsatisfiable, then a symbolic executor backtracks to the last visited branch, by removing every constraint added, when evaluating the unsatisfiable side of that branch, and explores the other side of it. One may notice that path exploration grows exponentially, with the number of branches; in literature, it is often referred to as *path explosion*. Path-based symbolic execution tools, as KLEE [162], employ several optimizations to prune the number of paths being explored, thereby caching previous queries and checking whether branch path constraints are satisfiable, before exploring them.

One notable advantage of symbolic executors, over BMC tools, is that the former can be used for coverage test generation. Recent results of software testing competitions do not lead to favorable assessment of BMC tools, over path-based symbolic execution tools, in order to produce test cases for achieving coverage*. Path-based symbolic execution tools explore state-spaces by using a depth-first search (DFS) algorithm, while a test case can be generated for each path of a program: every time a final state is reached, the underlying SMT solver can be queried for a model that satisfies all constraints in that particular path [163]. BMC tools, however, explore state-spaces by using a breath-first search (BFS) algorithm, so using it for coverage test generation is trickier, since many different paths can be evaluated, in a single run, and only one of them will be returned, if a bug is found.

*`https://test-comp.sosy-lab.org/2019/results/results-verified/`

5 SAT- and SMT-based Synthesis for Cyber-Physical Systems

5.1 Control Synthesis of Cyber-physical Systems

Control synthesis for ECPS consists in finding a controller or an invariant control law $\mathcal{C}$, such that a closed-loop system composed of $(\mathcal{C}, \Omega)$ satisfies every property $\mathcal{L} \in \Phi$.

A possible approach to synthesize reliable ECPS is Counterexample Guided Inductive Synthesis (CEGIS) [164]. Its key idea is the definition (by a user) of a set of safe states, *i.e.*, the specification of Φ, as well as a generic template τ for a control system that must satisfy such a specification, and then iteratively refine parameter values through counterexamples, until a final implementation is achieved. Then, a CEGIS engine computes values for the unknown symbols of the chosen template, which hold for Φ [165]. In particular, it splits the parameter-synthesis problem into simpler steps, which are then treated by SMT or SAT solvers. Abate *et al.* [9, 166] presented a method for synthesizing stable controllers that are suitable to continuous plants given as transfer functions [9] and state-space representations [166], which exploits bit-accurate verification of software implemented in digital microcontrollers [96, 105, 106], while the resulting systems must ensure non-fragile stability [9] and safety [166]. Indeed, they provide two complementary approaches based on CEGIS: one computes a completeness threshold k and then checks correctness over k time steps, in a multi-staged approach, *i.e.*, the adopted depth bounds the entire symbolic verification process, while the other relies on a conforming abstraction of a system's continuous dynamics, at specific times. Ravanbakhsh and Sankaranarayanan [167, 168] also employed CEGIS based on the Lyapunov's function, in order to ensure robust reach-while-stay property (*i.e.*, a safety specification) [168] and non-zeno behavior [167]. Although non-linear solvers constitute a bottleneck for those schemes, regarding computation complexity, the iterative nature of CEGIS approaches may be explored, in order to create trade-off points. CEGIS was also employed to synthesize model predictive controllers for a wide variety of applications, *e.g.*, heating ventilation and air conditioning systems, autonomous vehicles control, and aircraft electric power system [116].

The examples above showed promising techniques based on CEGIS, which may or already benefit from its iterative behavior. Indeed, convergence may be promptly achieved or even further oriented, if additional processing steps are placed between consecutive runs. In addition, CEGIS is an important, but not the unique, example of symbolic control synthesis for ECPS. The main advantage of using symbolic synthesis techniques (*e.g.*, SMT- and SAT-based) is the possibility of encoding (via logic formulae) reachability/safety specifications on original concrete systems, which breaks a complex synthesis procedure into smaller steps, as already mentioned. Indeed, there are many studies [55, 169–178] on symbolic control synthesis for ECPS, available in literature, which show the importance and flexibility of such a technique. For instance, Zamani, van de Wouw, and Majumdar [174] tackled incremental stability, through incremental Lyapunov functions and contraction metrics, and proposed a technique suitable to a larger classes of control systems, which enforces incremental input-to-state stability rather than input-to-state convergence. Indeed, incremental Lyapunov functions favored the construction of finite bi-similar abstractions for incrementally stable closed-loop control systems, which is an interesting result that reinforces combination between abstraction and control system properties.

A common procedure for controller synthesis for ECPS is based on finite-state approximations of infinite-states systems, *i.e.*, using hybrid automata. Such an approach allows fully automated synthesis of control systems for ECPS, which ensures complex-specifications realization; however, the main weakness of this approach is that it often produces complex controllers with high implementation cost. In order to avoid that problem, refined controllers have been proposed, which aim to provide some invariant properties for close-loop systems, *e.g.*, safety [178–180], reachability [176, 181], stability [171, 175], and safety/reachability properties, considering quantization effects [182].

5.2 Program Synthesis via Counter-Example Guided Inductive Synthesis (CEGIS)

The basic idea of program synthesis is to automatically construct a program P that satisfies a correctness specification σ. In particular, program synthesis is automatically performed by engines that use a correctness specification σ, as starting point, and then incrementally produce a sequence of candidate solutions that satisfy σ [9, 183]. As a result, a given candidate program p is iteratively refined, in order to match σ more closely. Counter-Example Guided Inductive Synthesis (CEGIS) represents one of the most popular approaches to program synthesis that are currently used in practice [183], whose basic architecture is shown in Fig. 8 and has close connections to algorithmic debugging using counterexamples and abstraction refinement [184].

The correctness specification σ provided to a program synthesizer is of the form $\exists \vec{F}.\forall \vec{x}.\sigma(\vec{x},\vec{F})$, where $\vec{F}$ ranges over functions, $\vec{x}$ ranges over ground terms, and σ is a quantifier-free formula typically supported by SMT solvers. The ground terms are interpreted over some finite domain $\mathcal{D}$, where $\mathcal{D}$ can be encoded using the SMT's bit-vectors part. The phases SYNTHESIZE and VERIFY, in Fig. 8, interact via a finite set of test vectors INPUTS that is incrementally updated. Given the correctness specification σ, the SYNTHESIZE procedure tries to find an existential witness $\vec{F}$ satisfying the specification $\sigma(\vec{x},\vec{F})$, for all $\vec{x}$ in INPUTS (as opposed to all $\vec{x} \in \mathcal{D}$). If SYNTHESIZE succeeds in finding a witness $\vec{F}$, the latter is a candidate solution to the full synthesis formula, which is passed to VERIFY, in order to check whether it is a proper solution (*i.e.*, $\vec{F}$ satisfies the specification $\sigma(\vec{x},\vec{F})$ for all $\vec{x} \in \mathcal{D}$). If this is the case, then the algorithm terminates; otherwise, additional information is provided to the phase SYNTHESIZE, in the form of a new counterexample that is added to the INPUTS set and the loop iterates again.

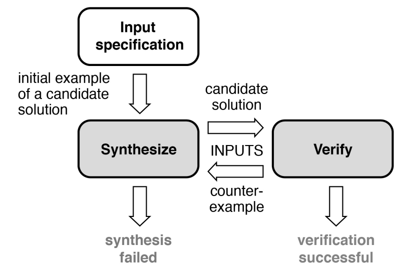


Fig. 8: Counter-Example Guided Inductive Synthesis (CEGIS).

One may notice that each iteration of the CEGIS loop adds a new input to the finite set INPUTS, which is then used for synthesis. Given that the full set of inputs $\mathcal{D}$ is finite, this means that the refinement loop can only iterate over a finite number of times; however, $\mathcal{D}$ can represent a large number of elements for the finite set INPUTS. In order to avoid exploring all possible values, machine learning techniques can be used in the phase SYNTHESIZE, with the goal of learning from experience (input-output samples), *i.e.*, learning from counterexamples provided by a verification oracle [184]. In addition to that, a pre-processing stage could also figure as another block in the scheme shown in Fig. 8, between VERIFY and SYNTHESIZE, which would process counterexamples and provide larger and refined information to the latter, according to specification σ and domain $\mathcal{D}$, in order to speed-up convergence to a final candidate.

Nowadays, program synthesis engines that implement the CEGIS approach [185] can automatically produce solutions for a large variety of specifications, due to the combination of automated testing, genetic algorithms, and SMT-based automated reasoning [186].

CEGIS for ECPS: A typical synthesizer for ECPS controllers iterates between two main phases, an inductive synthesis one, that is, SYNTHESIZE, and a validation one, that is, VERIFY [187]. The two phases interact via a finite set of test vectors, which is updated incrementally. Given an ECPS specification (*e.g.*, stability and safety), a inductive synthesis procedure tries to find a candidate solution satisfying that specification, for the given set of test inputs. If the synthesis phase succeeds in finding a witness, then the latter is a candidate solution for the full synthesis formula. This candidate is passed to the validation phase, which checks whether it is a proper solution (*i.e.*, it satisfies the specification for all possible inputs). If that is the case, then the algorithm terminates; otherwise, additional information is provided to the inductive synthesis phase, in the form of a new counterexample, C-ex, which is added to the set of test inputs, and the loop iterates again.

One possible architecture for a synthesizer is presented in Fig. 9, considering stability and safety specifications. It starts by devising a digital controller, through inductive synthesis, that stabilizes the physical model, while remaining safe for a pre-selected time horizon (k) and a single initial state. Then, it employs a multi-staged verification process, as follows. (i) The first verification stage (SAFETY) checks if the candidate solution, which we synthesized to be safe for at least one initial state, is safe for *all* possible initial states, *i.e.*, it does not reach an unsafe state within k steps. The tentative system is unfolded k steps and we check if the target safety specification holds for any initial state, according to Algorithm 1. (ii) The second verification stage (PRECISION) restores soundness with respect to plant precision, by using interval arithmetic [188] to validate operations performed by the previous stage. (iii) The third verification stage (COMPLETE) checks if k is large enough to ensure safety for any $k' > k$, where k is computed to ensured $k \geq \overline{k}$ for the current candidate controller [166], *i.e.*, k is at least the number of iterations required to sufficiently unwind the closed-loop state-space model, such that the boundaries are not violated for any larger number of iterations.

ALGORITHM 1: Safety check

```
Function Safety check begin
    Check if input is bounded
    for All possible initial states do
        for All time steps until $k$ do
            Compute system evolution
            Check if a safe state was achieved
        end
    end
end
```

As a real example, it is interesting to revisit system Sim3Tanks, which was presented in Section 2.3. Through the CEGIS approach presented by Abate *et al.* [9, 166], an iterative design would then be performed, where candidates are generated and then verified, based on properties ϕ_1, ϕ_2, ϕ_3, and ϕ_4. As already mentioned, the first three are related to safety, regarding fluid height in tanks, while the latter is related to liveness, in such a way that flow Q_3 follows a desired form. Indeed, checking the latter is a little bit more intricate, given that the it must comply with a (possibly dynamic) form for k steps and is highly dependent on the aggregate effects of that system, which includes reservoir and fluid characteristics. As a consequence, introducing a suitable model, which could be used both in the verification and also in an additional pre-processing step, is a interesting

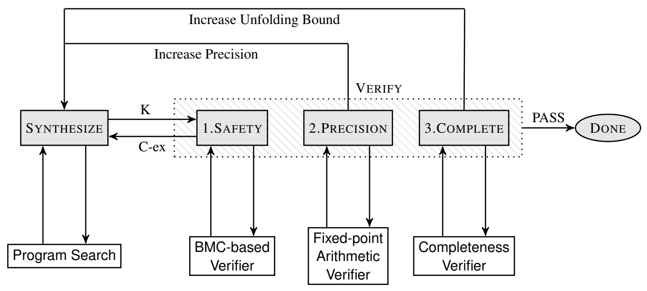


Fig. 9: CEGIS with multi-staged verification for ECPS.

strategy. In the former, a verifier would check if Q_3 follows z_{ref} and, in the latter, a processing and tuning stage would provide refined information aligned with the mentioned system model, in such a way that the synthesis step would converge toward a viable solution, in a system sense. Finally, the verification step could also include a desirable range for Q_3, instead of am exact curve.

5.3 Incorporating System Models to Automated Verification and Synthesis Procedures

Currently, SMT-based BMC approaches check code properties in real programs, which basically address programming-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 source code is correctly written, but also if it will properly respond in real environments or under external problems or corrupted data. 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 or even intentionally tampered [109].

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) [105] and is able to aid engineers to check overflow, limit cycle, output error, timing, stability, and minimum phase, considering 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 [96]. That ultimately leads to the use of analog-to-digital (A/D) converters, which are one of the most important aspects to be considered, given that data loss (quantization) is inevitable. As a consequence, verification procedures have 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. Additionally, the latter may present a different behavior, regarding what was specified in analog domain, due to the inherent discrete-time operation.

DSVerifier is actually a front-end for internal modules of the Efficient SMT-Based Context-Bounded Model Checker (ESBMC) [131], with the goal of converting digital system specifications into C syntax containing representation format, realization structure, and configuration parameters, in order to use the entire ESBMC's verification chain. Its verification methodology is shown in Fig. 10, which starts with a digital system design step that can be done through traditional techniques available in literature. Then, some implementation parameters must be defined, in the second and third steps, which are finite word-length format (fixed-point representation), dynamic range, and realization form, including direct and delta forms. Following that, other hardware and verification parameters are fed to its engine, which includes number of bits, verification time, and property to be checked, the latter being overflow, limit cycle, timing, stability, or minimum phase. As a result, its verification process is started and design parameters are checked as system properties: if there is no property violation, its verification result is "successful"; otherwise, "failed" is returned, together with a counterexample. Based on the latter, other implementations can then be tried, by a designer, in other to avoid failure, in an iterative way.

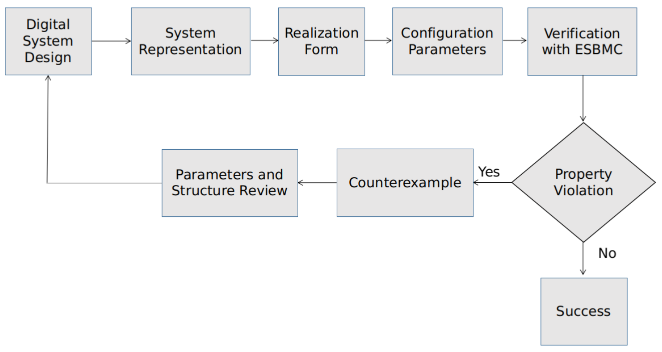
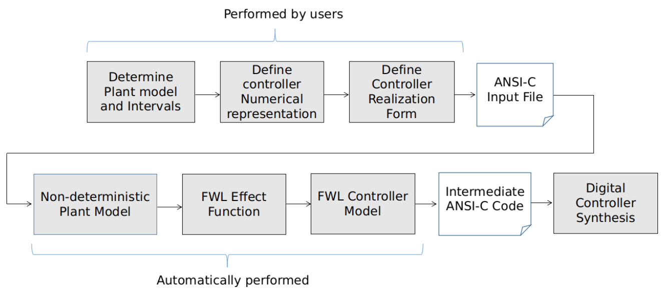


Fig. 10: DSVerifier's verification methodology.

In summary, DSVerifier is a useful test tool regarding ECPS, which takes into account different representations (*e.g.*, transfer-function and state-space), realization forms (*e.g.*, direct, delta, and transposed forms), and other implementation restrictions, in order to explore design spaces. It has been already applied to real-world cases and scenarios, mainly regarding digital filters [105] and controllers [70], while tackling many different representation formats, implementations structures, and safety properties. Finally, if a system's requirements are not met with a given configuration, an analysis of the provided error report may suggest another setup, which can even be incorporated into a given system development procedure, by providing both system verification and model refinement.

In this respect, DSVerifier has been extended to automatically synthesize digital stabilizing controllers for continuous plants represented as transfer functions or state-space equations [9, 166]. In particular, a CEGIS-based approach, implemented in a tool called digital system synthesizer (DSSynth), uses inductive synthesis in conjunction with the DSVerifier's algorithm for verifying robust closed-loop stability that addresses plant variations as interval sets, as well as FWL uncertainties in digital controllers. DSSynth is able to successfully synthesize stable digital controllers for a set of intricate plant models, taken from the control literature, within minutes. The DSSynth's synthesis process is shown in Fig. 11. The first three steps are performed by users and are also similar to what is already done by DSVerifier, while the next three ones are automatically performed, which result in a non-deterministic model for representing a plant family, a function to compute controller parameters, and ANSI-C code for representing a digital system implementation, including a specification for the property to be checked, respectively, which is then used as input for its CEGIS engine. After a synthesis procedure, stability of closed-loop control systems are checked, while considering fragility aspects, such as FWL problems and uncertainty in plant models. Nonetheless, system correction is of paramount importance and, even with such a synthesizing tool, a final verification procedure should be executed, with the goal of ensuring that the generated solution still complies with the provided specs and other issues, which may have not been initially tackled and have the potential to compromise system reliability, are not present.

Apart from the control system domain, Scratch is another example software model-checking tool, which uses knowledge about the system to be verified and the underlying hardware, with the goal of detecting races related to direct memory access (DMA) regarding scratchpad memory, in the Cell BE processor [145]. That tool also uses SAT-based BMC, in order to detect DMA races, and SAT-based BMC with k-induction, which aims to prove the absence of them. Scratch uses four variables for tracking DMA operations, *i.e.*, *valid*, which is true if a DMA operation is tracked, *addr*, *sz* and *tag*, which represent address, size and tag of a pending DMA operation, on



Performed by users
Determine Plant model and Intervals
Define controller Numerical representation
Define Controller Realization Form
ANSI-C Input File
Non-deterministic Plant Model
FWL Effect Function
FWL Controller Model
Intermediate ANSI-C Code
Digital Controller Synthesis
Automatically performed

Fig. 11: DSSynth's synthesis methodology.

instrumented programs, where a given DMA operation is compared with all previous ones. For instance, one might consider an operation $get(l, h, s, t)$ that transfers data of s contiguous bytes, from region h, to local memory region l, while identified by tag t dependent on the processor's word-size. Scratch would then replaces such an operation with

- $assert((unsigned)t\ < ws\ \&\&\ (unsigned)s\ <\ max)$;
- $assert(!valid\ ||\ l + s\ <=\ addr\ ||\ addr + sz\ <=\ s)$;
- $memset(l, *, s)$;
- $if(*)\{valid = true; addr = l; sz = s; tag = t;\}$.

Such statements ensure that parameters are within a hardware's range, check if two DMA operations do not race, model generic transfers, and update variables for a new operation, respectively. Indeed, all operations supported by the Cell architecture could be similarly encoded. 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.

One may also notice that such a paradigm, *i.e.*, verification and synthesis based on an expected system behavior, is not restricted to digital filters and controllers or DMA races, but it can also be applied to possibly any real system, as long as the desired behavior can be expressed as properties in BMC frameworks. For instance, regarding self-driving cars, an important property could be the detection of pedestrians, animals, bicycles, or any obstacle present on urban and rural roads. Indeed, bicycles are considered one of the most difficult problems, due to their myriad of possible shape and colors [189]. In addition, the mentioned problem is closely related to machine learning, from which refinement strategies can be devised, incorporated to BMC approaches [190], and directly applied to ECPS verification.

In summary, current approaches already tackle ECPS characteristics (*e.g.*, interaction with real world entities and digital processing), which are included in verification and synthesis methodologies. Finally, the final complement may come from ML techniques, which have the potential to fill gaps (*e.g.*, lack of complete models including non-determinism), bridge tools and models (*e.g.*, analog and digital domains in ECPS and verification schemes), and provide model and methodology evolution.

6 Verification and Synthesis Challenges for ECPS

Generally, state-of-the-art verification methodologies for embedded systems (including ECPS) generate test vectors (with constraints) and use some assertion-based verification and high-level processor models, during simulation [191, 192], as illustrated in Fig. 12.

In particular, the main challenges regarding verification of embedded systems lie on improving coverage, where more system functions are verified, reducing verification time, *i.e.*, pruning state-space exploration during verification, providing completeness, *i.e.*, if all possible states can be reached and evaluated, and incorporating system models, which allows specific checks regarding system behavior and not only code correctness. Additionally, embedded-system verification raises additional challenges, such as:

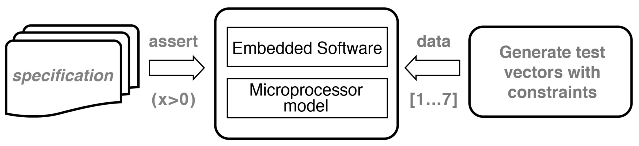


Fig. 12: Verification methodologies for embedded systems.

1. Time and energy constraints;
2. Handling of concurrent software;
3. Platform restrictions;
4. Security issues;
5. Verification of code pieces or modules that rely on larger structures;
6. Legacy designs;
7. Support to different programming languages, frameworks, and interfaces;
8. Correct code instrumentation;
9. Handling of non-linear and non-convex optimization problems;
10. Incorporation of new and adapted checks, due to system or environment change;
11. Provision of system evolution, in such a way that a version more adapted to a given scenario is obtained.

Indeed, the first two aspects are of extreme relevance in micro-grids 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. In that sense, as already mentioned, DSVerifier [70, 105] is a powerful tool, which has been evolved and now is capable of checking many properties in digital systems, including digital filters, digital controllers, and closed-loop systems [193], while taking into account fragility aspects (*e.g.*, FWL problems). In addition, it has also inspired synthesis methodologies [9, 166] based on CEGIS. Indeed, DSVerifier is being developed towards complete ECPS verification, which includes components and entire system behavior, based on models tackling aggregate behavior and fragility. Finally, Sim3Tanks (see Section 2.3) can be considered as a suitable benchmark for future versions of DSVerifier, given that it includes all mentioned challenges.

The fourth aspect is very important, mainly in the context of IoT and ECPS. Indeed, remote management is of paramount importance, leads to connection through private networks and also the Internet, and may suffer from cyber attacks. For instance, Beg, Johnson, and Davoudi [194] studied DC micro-grids, which are ECPS with sophisticated interactions between physical and computer entities, and devised a framework to detect possible false-data injection attacks, through identification of changes in candidate invariants. In summary, reachability analysis based on hybrid automata was employed and actual invariants, obtained from the reach sets contained within the candidate ones, were compared with the latter, which then indicated occurrence of attacks.

The fifth aspect deals with code that relies on existing infrastructures and must be compliant with those, such as Linux kernel modules. Witkowski [195] addressed that problem and proposed a tool called DDVerify, which is able to verify Linux device drivers through SatAbs, with focus on Linux kernel application programming interfaces (APIs) described by natural language. The sixth aspect, in turn, is inherent to a large number of embedded systems from telecommunications, control systems, and medical devices, including ECPS. In particular, software developed for those types of embedded system has been extensively tested and verified, and also optimized for efficiency over years of development. Therefore, when a new product is derived from a given platform, a lot of legacy code is usually reused for improving development time and code quality. Nonetheless, legacy code usually presents specifications written *post hoc*, it is often poorly developed, and its verification may require interaction, which present hard challenges for any employed approach [196].

The seventh aspect is related to evolving development processes and technologies, which may delay the application of suitable verification and synthesis approaches, if verifiers and synthesizers do not support different programming languages and interfaces. In this context, Monteiro *et al.* [197] proposed the use of a QT operational model for verifying code written with the Qt framework, which is a simplified version of the latter and is already instrumented for formal verification. It is not exactly an ECPS verification technique; however, it shows that if ECPS are developed with a specific framework, its verification is still possible and can scale on that concept. The eighth aspect highlights a subjective task in verification: where one must insert an evaluation point or assertion, in order to correctly address a property. In that direction, Li *et al.* [198] proposed an automatic approach for software verification based on instrumentation and concolic testing, which uses C intermediate language and control C graphs for identifying security-sensitive parts (*e.g.*, buffer overflow). Indeed, instrumentation driven by target issues (not a generic analysis) seems to be a promising approach, which could also greatly benefit from behavior models. For instance, in Section 2.3, we pointed out target properties for Sim3Tanks, but we did not define where they should be checked, which could be resolved through a complete system model. The ninth one is related to the widespread use of embedded systems in autonomous-vehicle navigation systems [199], which demands optimization solving during their execution for a wide range of functions, including non-linear and non-convex problems using fixed- and floating-point arithmetic. One example is optimization through counterexample guided inductive optimization (CEGIO) [114], which uses an iterative approach, based on formal verification, to perform inductive generalization and then reduce optimization domains. As a consequence, one may notice that formal methods may be regarded as a base framework, which can be applied to many problems related to ECPS design and verification.

The tenth aspect is a consequence of changes in environment or different applications and scenarios, which may present new problems to embedded systems that did not exist or were not tackled during their development. Such a challenge is also closely related to ML techniques, given that they may be integrated into commercial systems and have the potential to devise new and refined control strategies, which must then be identified, checked, and validated [200]. In a simplified manner, verification of ML-based systems can be classified into two groups: development of monitors to ensure safety rules, such as maintenance of a safe distance from fragile elements (*e.g.*, bicycles and animals), as already mentioned earlier in the present article, and verification regarding satisfaction of safety criteria, such as erratic or dangerous behaviors, which is very difficult to formally define, in any given language. For instance, a diagnostic system may employ techniques that make it able to learn new tumor patterns and them warn their presence on patients' bodies, but a wrong updated model may compromise earlier traditional assessments and erroneously perform new ones, which should be verified and validated. Finally, the latter also raises another question: an embedded system must be checked only when it is developed/created or also during its life cycle, in order to ensure that improved models and actions are sound and reliable? That is an issue that will probably concern researchers, when ML becomes mature and widely spread in software verification.

The eleventh aspect is related to the current development and test strategies adopted for embedded systems, which aim at simply providing robust structures, instead of anti-fragile ones [201]. In summary, it is not feasible to account for every error or faulty scenario and, as a consequence, it would be more beneficial to provide a module, architecture, or methodology capable of thriving on problems and getting stronger. Such an aspect shares some similarity with the ninth one and may also be achieved through ML techniques. Indeed, this kind of scheme consists in a completely different approach, which would profoundly change the way we design ECPS.

Those eleven challenges place additional difficulties for developing reliable synthesizers for embedded systems, especially for cyber-physical systems and micro-grids, where controlled objects (*e.g.*, physical plants) typically exhibit continuous behavior (which may eventually change), whereas controllers (usually implemented by real-time computer systems) operate in discrete time and over a quantized domain. In particular, synthesizers for those systems [9, 166] need to consider the effects of quantizers (A/D and D/A converters), when a digital equivalent of the controlled object is considered, *i.e.*, a model of their physical environment (*cf.* the intelligent product in Fig. 1). Additionally, finite-precision arithmetic and their related rounding errors need to be considered, when correct-by-construction code is generated for ECPS. Additionally, synthesis of ECPS raises additional challenges, such as:

1. Parametrization of search-space exploration processes;
2. Improvement of learning processes via counterexamples;
3. Use of incremental SAT/SMT solving approaches.

The first aspect is related to word lengths and representations of variables in an employed (embedded) hardware, where ECPS are built upon [105]. Here, floating-point representations can provide better approximation of real numbers, when compared with fixed-point ones with the same number of bits, which typically varies from 16 to 64 bits, in popular hardware architectures used for ECPS. Multiple-precision floating-point arithmetic can further represent real numbers, whose precision digits are bounded by the available memory of a system, and practical software packages do exist to implement that type of arithmetic (*e.g.*, MPFR* and MPFI†). Nonetheless, using floating-point arithmetic in software synthesizers usually leads to higher verification time and memory consumption, as previously observed by Duggirala and Viswanathan [103]. In order to synthesize ECPS, typical synthesizers need to parameterize search-space exploration via multiple choices of word-length and variable representations, in order to produce a candidate solution that considers time and memory limits.

The second aspect deals with the way software synthesizers can learn from counterexamples produced during a verification phase, in order to quickly propose a candidate solution that meets a given specification. Here, critical aspects for any CEGIS approach, such as counterexample selection, are still poorly understood by the research community, thereby resulting in completely ad-hoc designs. Further investigation of those aspects would allow software synthesizers to come up with efficient synthesis approaches, for instance, through the use of an additional processing step before counterexample input, so that reliable implementations of communication and control software in ECPS could be learned, in order to quickly achieve a correct-by-construction design. Machine learning techniques, such as reinforcement learning and decision tree learning, could be used, in order to achieve that goal and then speed up synthesis processes.

Lastly, the third aspect is related to the ability of SAT/SMT solvers to incrementally solve propositional or fragment of first-order logic formulae. In particular, incremental SAT solving [202] allows software synthesizers to deal with sufficiently large logical formulae, incrementally, by learning from previous checks, *i.e.*, check VCs in iteration $k+1$ based on the work done for iteration k, thus optimizing search procedures and potentially eliminating a large amount of formula state-space to be tackled. Some prior studies [134, 203] show that incremental learning can cut runtimes by one order of magnitude, in comparison with standard non-incremental approaches over a large set of industrial embedded benchmarks (mainly from the automotive industry); however, we have not seen much effort, in the research community, regarding use of incremental SAT/SMT solving to efficiently synthesize control software for ECPS, through incremental learning (from previous checks).

7 Current Achievements and

* *https://www.mpfr.org/*

† *https://directory.fsf.org/wiki/MPFI*

Future Trends

In the preceding sections, we identified and discussed many aspects regarding ECPS, which included design, synthesis, behavior, verification, and bottlenecks. Nonetheless, from the point of view of computer-aided verification and synthesis of ECPS, it is possible to highlight seven major research problems (RPs), which are tackled here and can be regarded as (partially) open in current published research.

(RP1) provision of suitable encoding into SMT [118], which may extend background theories typically supported by SMT solvers, with the goal of reasoning accurately and effectively about realistic ECPS.

(RP2) exploitation of SMT techniques to leverage (bounded) model checking of multi-threaded software, in order to mitigate the state-explosion problem due to thread interleaving, when verifying ECPS implementations that require multi-core processors with scalable shared memory.

(RP3) proof of correctness and timeliness of ECPS, by taking into account stringent constraints imposed by hardware platforms.

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

(RP5) provision of 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.

(RP6) development of automated synthesis approaches that are algorithmically and numerically sound, in order to handle (control) software that is tightly coupled with physical environments, by considering uncertain models and FWL effects.

(RP7) provision of a unified framework, which is able to tackle high and low level properties, as well as system behavior and even its evolution, with the potential to lead to another degree of completeness, in such a way that every implementation aspect of ECPS is addressed and evaluated.

Due to the fact that the mentioned research problems are some of the main issues found in the technology area addressed in this article, many studies have already tackled them and other researchers still perform experiments and suggest new methodologies and schemes for mitigating or solving their consequences. As a result, we can cite the following contributions toward them, which already provide some answers or at least present promising approaches.

7.1 Research Problem 1: SMT Encoding

Symbolic model checking using SAT solvers instead of BDDs was first proposed by Biere *et al.* [117], in order to verify hardware systems. A successful implementation of that approach to verify C programs was proposed by Clarke *et al.* [127], via CBMC. Armando *et al.* proposed the first SMT-based BMC for C programs [119], which was further extended by Cordeiro *et al.* to support the SMT encoding of full C programs via ESBMC [131]. The latter, in turn, was then extended to support C++03 programs [204], CUDA programs [19], and Qt-based consumer electronics applications [197]. 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 their creators (*e.g.*, NOKIA, NEC, NXP, and VERISEC) [129, 131, 204].

Other symbolic verification approaches have also been presented in literature [128, 140, 155, 205], but the coverage and performance of all existing ones are still limited to specific program classes, suffering performance degradation for programs that contain intensive floating-point arithmetic and dynamic memory allocation [206]. *k*-Induction based verifiers, which use BMC as a "component" to prove partial correctness, are continuously gaining popularity in the software verification community. Beyer, Dangl, and Wendler [140] described a software verification-framework called CPAchecker, which automatically generates invariants through a combination of data-flow-based invariant generator and dynamic precision adjustment, with the goal of injecting (inductive) invariants into the *k*-induction algorithm. Even though the combination between *k*-induction and continuously-refined invariants significantly enhances verification results, there is still room for improvements, particularly for learning from counterexamples, in order to improve bug finding capabilities (*cf.* Section 7.3).

Here, one possible research direction is to bridge the gap between symbolic verifiers and SMT solvers, propose background theories, and develop more efficient decision procedures, in order to handle specific program classes. In this particular research direction, the European Project SC^2 (Satisfiability Checking and Symbolic Computation: uniting two communities to solve real problems) aims to further extend background theories supported by SMT solvers, in order to solve problems related to security and safety of computer systems (or large mathematical problems), through the development of radically improved software tools [207].

One may notice that the main bottleneck of existing symbolic verifiers is to unfold complex transition systems. As described in Section 3, there are approaches that extend BMC without unrolling the transition system (*e.g.*, IC3 and Interpolants), but they do not cope well with software systems, as recently reported in SV-COMP [206], either due to the type of VCs that are produced or simply because they might require further engineering effort, in order to make them useful for software verification.

Another interesting direction for future research consists in investigating the application of separation logic [208], in order to reason about ECPS. The key idea of separation logic is to perform a local reasoning based on specifications and proofs of a given program component, by considering only the portion of memory used by that same component, and not the entire program global state; thus, this allows modular reasoning among different program components, with the goal of scaling formal verification to larger software systems. Infer [209] represents one of the most successful implementations of that approach; it is an open-source static code analysis tool used for verifying the Facebook code base, was employed on many mobile apps (*e.g.*, WhatsApp), and was adopted by many companies (*e.g.*, Mozilla and Spotify).

7.2 Research Problem 2: Verification of Multi-thread Software

The symbolic verification methods proposed in literature were further developed to verify correct lock acquisition ordering and absence of deadlocks, data races, and atomicity violations, in multi-threaded software based on POSIX and CUDA libraries [19, 129, 210], while considering monotonic partial-order reduction [211] and state-hashing [212] techniques, in order to prune state-space exploration.

Recent advances on verification of multi-threaded C programs have been proposed to speed up verification times, which significantly prune state-space exploration [210, 213], and also extensions to verify the Debian GNU/Linux distribution, based on a context- and thread-sensitive abstract interpretation framework [214], were made available. As an example, Lazy-CSeq is a code-to-code transformation tool based on Lazy Sequentialization techniques, initially introduced by La Torre *et al.* [215] for non-deterministically sequentializing concurrent C programs, which re-uses existing BMC tools as backends, in order to find code violations [210]. Its main idea is to pre-process a concurrent program and convert it into a non-deterministic sequential one, with the goal of simplifying verification tasks. Nonetheless, the set of concurrent-program classes (*e.g.*, CUDA, OpenCL, and MPI) that can be verified is still very limited.

Given that typical implementations of ECPS now require multi-core processors with scalable shared memory, in order to meet the increasing computational power demands, one possible research direction is to further extend symbolic verification methods of multi-threaded programs via Lazy Sequentialization [210], by using SMT solvers, in order to analyze unsatisfiability cores [216], with the goal of removing redundant behavior or analyzing interpolants [152, 205] and then prove non-interference of context switches. This will enable one to capture, within a single verification framework, multi-threaded programs that communicate via shared memory, for the variety of (weak) memory models that are implemented in today's computer architectures, and also distributed programs, which typically rely on a message-passing communication style. Scaling up this single verification framework, for multi-threaded programs, has the potential to increase the level of safety and security of ECPS implementations and thus contribute to the vision of fully verified embedded software.

7.3 Research Problem 3: Proof of Correctness and Timeliness

Novel approaches to model check embedded software using k-induction and invariants were proposed and evaluated in literature, which demonstrate its effectiveness in some real-life embedded-system applications [141, 142, 144, 145]. In particular, the k-induction proof rule of ESBMC (named ESBMC-kind) has been ranked at the top 3 in SV-COMP 2018*; however, the main challenge remains still open, *i.e.*, to compute and strengthen loop invariants for proving program correctness and timeliness, in a more efficient and effective way, in order to be competitive with other model-checking approaches.

Invariant-generation algorithms have substantially evolved over the last years, with the goal of discovering inductive invariants of programs [138, 139] or continuously refine them during verification [140]. For instance, Beyer, Dangl, and Wendler [140] proposed k-induction combined with automatically-generated continuously-refined invariants, in order to iteratively increment the induction parameter k and then search for stronger invariants. As a conclusion, the authors stated that efficient k-induction approaches are possible; however, there is still room for improvement. Indeed, there is 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. In this respect, Jovanović *et al.* [217] present a reformulation of IC3, by separating the reachability checking from the inductive reasoning. They further replace the regular induction proof rule by k-induction and show that it provides more concise invariants. The authors implemented their algorithm in the SALLY model checker, with Yices2 for the forward search and MathSAT5 for the backward one. They showed that the new algorithm can solve a number of real-world benchmarks, at least as fast as other approaches. Lastly, optimal configurations for invariant generators are still of paramount importance, given that applications with code based on different aspects may benefit from different setups.

7.4 Research Problem 4: System-level Properties

State-of-the-art symbolic verification approaches were extended to verify overflow, limit cycle, time constraints, stability, and minimum phase, in digital systems, which was first proposed by Cox *et al.* [218]. Indeed, digital filters and controllers [105, 218, 219] were tackled, in order to specify system-level properties of those systems, by employing either LTL for verifying embedded software used in medical devices [121] or simply using user-specified assertions for verifying embedded software used in the automotive industry [203]. Additionally, a specific UAV application was tackled using DSVerifier, with the goal of checking UAV's attitude controllers [106], which integrated knowledge into symbolic verification procedures regarding a quadrotor's orientation, w.r.t. an inertial reference system described by the Euler angles: pitch, roll, and yaw [70].

There are other verification tools that provide similar features for verifying digital filters and controllers, in ECPS, such as Astrée [158], PolySpace [159], and Simulink Design Verifier (SDV) [220]. Those tools are able to verify, to some extend, low-level properties in digital systems. Although Astrée works on pre-processed C code, it tackles only digital filters and is focused on verifying overflow and register dimensioning, which means that it is not prepared to handle digital controllers and physical plants. SVD is focused on block level (Simulink) and needs substantial work regarding requirement expression and its respective encoding, in order to support digital filters and controllers. Finally, PolySpace is more software oriented and generically handles potential run-time errors, in ECPS, while also leaves code fragments for further review.

More recently, other implementation aspects were added to those existing verification framework, including magnitude, phase, poles, and zeros, which provide deeper analysis regarding digital systems, by tackling frequency-domain parameters, permissible deviation, and associated natural response [219]. In general, however, there is still a lack of studies to verify system-level properties related to ECPS: emphasis should be given to micro-grids [2], which present 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, with the goal of making symbolic verification tools useful for ECPS engineers [221].

7.5 Research Problem 5: Support to Different Tools and Applications

Although existing symbolic verification tools [127, 128, 131, 222] were extended to support Java, C/C++, and some variants, new application interfaces and programming languages are often developed, which require suitable software verification tools for specific frameworks and embedded platforms, in order to build ECPS. Indeed, it would be interesting if a new programming language model could be loaded, which, along with symbolic verification core algorithms, were able to check different programs and frameworks.

Some work towards that goal was already presented by Monteiro *et al.* [197], which employed operational models for checking Qt-based programs from consumer electronics. In summary, the symbolic verification core (in that case, ESBMC) is not changed (it is still C/C++ code), but instead an operational model, which implements the behavior and features of Qt libraries, is used to provide the new code structure to be checked. This approach could also adopt the Clang compiler [223], which is already widely used in industry [224], in order to produce an Abstract Syntax Tree (AST) for a wide variety of programming languages (*e.g.*, C/C++/ObjectiveC/ObjectiveC++). This AST can then be converted either into an intermediate representation (IR) of the underlying verifier or into LLVM bitcode that is already produced by Clang. In this respect, Clang even provides a static analyzer (called CSA) [225], which is an open-source project built on top of Clang that can perform context-sensitive inter-procedural analysis for programs written in languages supported by Clang. It is designed to be fast, so that it can provide results for common mistakes (*e.g.*, division by zero or null pointer dereferencing), even in complex programs.

Obviously, a completely new language may present new structures and resources; however, if different inputs are parsed and converted into a form that preserves every aspect of their structure, it can then be analyzed and verified in an unified way. Such a research problem is closely related to the first one **(RP1)** and has the potential to devise a new paradigm in ECPS verification.

7.6 Research Problem 6: Formal Synthesis

State-of-the-art synthesis approaches for embedded (control) systems typically disregard the platform in which the embedded system software operates and restrict themselves to generate code that do not take into account FWL effects [226]. Synthesized systems, however, must include physical plants, in order to avoid malfunctioning (or even a catastrophe), due to embedded (control) software, *e.g.*, the Mars Polar Lander did not account for leg compression, prior to landing [227].

Research in this direction has made some progress to design, implement, and evaluate an automated approach for generating correct-by-construction digital controllers, which is based on state-of-the-art inductive synthesis techniques [9, 166]. Nonetheless, there is still little evidence whether that approach can scale on larger

* *https://sv-comp.sosy-lab.org/2018/results/results-verified/*

systems modeled through other representation types (*e.g.*, Multiple-Input Multiple-Output).

In addition to that, another research direction for synthesizers is to generate code for UAV trajectory and mission planning, by taking into account system's dynamics and nonholonomic constraints [114]. As a consequence, verifiers and synthesizers need to handle a wide range of functions, including non-linear and non-convex optimization problems based on both fixed- and floating-point arithmetic.

Lastly, the provision of a unified synthesis framework to a very large class of problems is a challenging task for synthesis-based tools. The investigation and development of an automated formal synthesis framework based on machine learning techniques (*e.g.*, reinforcement learning and decision tree learning), which is algorithmically and numerically sound, will enable one to synthesize embedded software for ECPS that is tightly coupled with physical environments, with the goal of achieving safety and security in real implementations.

7.7 Research Problem 7: Simultaneous Verification of Low- and High-Level Properties

Embedded systems are typically implemented in low- and medium-level programming languages (*e.g.*, ANSI-C/C++, Java, and Python). Given the availability of different programming languages for writing code for ECPS, the provision of a unified framework able to tackle low- and high-level properties, as well as system behavior, is needed to achieve code coverage in real applications. With that goal in mind and also considering the widespread use of embedded software that implements ML algorithms, *e.g.*, deep neural networks (DNNs), this unified framework should also be able to check for system's evolution, given the learning nature of ML-based algorithms used in a range of hard problems in artificial intelligence, *e.g.*, games, robotics, natural language processing, and image classification. In addition, adequate modeling for formal verification is still needed and synthesis of ECPS phenomenons and associated risks are still incipient, *i.e.*, it is necessary the development of simplified models that are able to represent some particular ECPS problems, such as communication delays, network availability, faults, and cyber attacks.

The Clang compiler [223] offers a simple and strong approach to lead software verification to another degree of completeness, so that every implementation aspect of ECPS can be addressed and evaluated. In particular, by using the Clang compiler as front-end for a software verification tool, one can avoid the need for maintaining a proprietary front-end, which is a real challenge nowadays, given that the C and C++ standards are rapidly evolving, and then focus on the main objective: formal program verification and synthesis for ECPS. In RP5 (*cf.* Section 7.5), we describe the use of operation models to tackle low- and high-level system properties, via an abstract representation of the associated libraries, which conservatively approximates their semantics. Those operational models could also be developed to check specific properties in ML-based software, *e.g.*, the NVIDIA CUDA deep neural network library (cuDNN),[*] which is a library for implementing DNNs in graphical processing units (GPUs). Those operational models could then describe behavior, pre-, and post-conditions of routines, such as forward and backward convolution, pooling, normalization, and activation layers, in order to achieve high code coverage and find adversarial examples in DNNs. The Clang compiler offers an industrial quality analyzer and an application programming interface to access and traverse its internal AST, which can then be used by software verifiers to generate their IR or alternatively analyze the LLVM bitcode produced by Clang.

Finally, BMC tools like CBMC [127], ESBMC [228], SMACK [229], and LLBMC [128] represent the most prominent approaches for verifying C programs, as observed in the Intl. Competition on Software Verification [206], where verifiable (correctness and violation) witnesses are important for evaluating software verifiers [230]. The increasing number of verification tasks being added every year to this competition allows developers to further improve their verifiers, by implementing new types of abstraction and simplification techniques. As a consequence, the continuous development and improvement of verifiers can leverage more efficient synthesizers and further increase the reliability of verified ECPS.

[*] *https://developer.nvidia.com/cudnn*

7.8 Relation among the RPs and Challenges

The presented research problems cover most open challenges regarding ECPS verification and synthesis, while a brief discussion including current results and ongoing work is provided. In addition, even the challenges raised in Section 6 are related to those RPs, as shown in Table 1, which has the goal of further clarifying what was previously explained.

Research Problem	Challenges
RP1	1, 2, 3, and 4
RP2	2 and 4
RP3	1
RP4	7 and 9
RP5	5 and 6
RP6	10
RP7	5, 6, and 9

Table 1 Relation among RPs and challenges.

7.9 Vision for Future Research

Fig. 13 illustrates our vision for employing symbolic verifiers, as discussed in Section 4, to synthesize correct-by-construction ECPS implementations, where we start with a correctness specification σ of a system, *e.g.*, stability, safety, performance, and behavior. The basic idea is to automatically construct a program P that satisfies σ. In particular, our program synthesizer will automatically use σ as starting point and then incrementally produce a sequence of candidate solutions that partially satisfy σ. As a result, a given candidate program p is iteratively refined, in order to match σ more closely. On every refinement iteration to find our candidate solution, we add the respective counterexample produced by our symbolic verification engine to a test suite, which can be used later for automated (regression) testing. This way, the main challenge lies on exploiting effectively and efficiently counterexamples provided by symbolic verifiers (including their scalability to handle complex ECPS models), in order to quickly learn reliable embedded software implementations.

8 New Applications: Beyond Code Correctness

As already mentioned in section 2, even problems that are not directly related to code correctness may be tackled with formal methods, as long as they are conveniently modeled. Araújo *et al.* [114] employed SMT for handling non-convex optimization problems, with the goal of ensuring optimal solutions. Indeed, it is possible to recursively re-constrains a model checking procedure, by using the current counterexample as a seed for the next run, until a global minimum is achieved. Nonetheless, elapsed execution times are still a bottleneck for some applications, such as changes in UAV trajectories, but that can be improved, through the use of simplification techniques and problem constraints.

In the same fashion, other problems may benefit from the recursive-refinement approach provided by SMT-based verification schemes. For instance, de Jesus *et. al* [231] developed a simplified methodology for antenna alignment that makes use of many

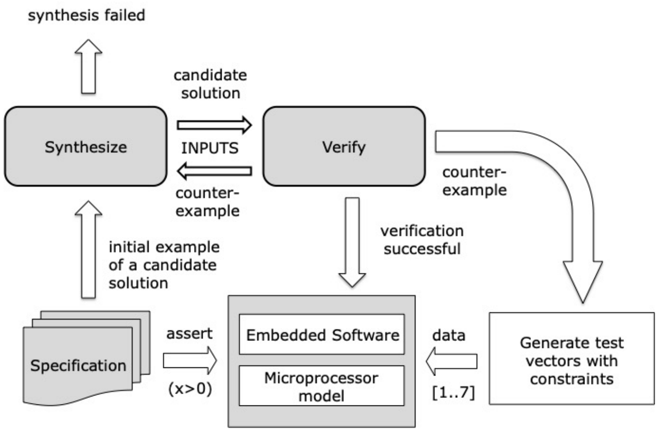


Fig. 13: Automated Verification and Synthesis for ECPS.

modules already available in satellite TV receivers, along with a suitable structure for performing position correction. In summary, a reference signal is evaluated, until an (approximately) error-free reception is acquired, which can be measured through a simple transport stream (TS) error-indication interface and further refined with other parameters, such as carrier-to-noise ratio and signal level. This way, by using an SMT-based verification method, one may map TS error indication and other performance figures as properties and then recursively perform realignment procedures, until no counterexample is output. It is worth noticing that associated timing requirements should be more relaxed, as mentioned in the work developed by Araújo *et al.* [114], which can be seen as an advantage for such an implementation.

Regarding behavior correctness, many other applications can be tackled, such as the ones related to automatic classification. Amoedo *et al.* [232] proposed a modulation classification methodology, with the goal of spectrum sensing for opportunistic transmissions. Besides a correct software implementation, *i.e.*, correct source code, which is already covered by formal methods already available in literature [127, 131, 219], one may argue if the proposed methodology is able to recognize modulations under some reception condition, such as severe analog interference. This way, it would be interesting to develop a verification module for transmission and reception software, which could be capable of creating inputs related to sampled radio frequency signals, under a variety of conditions, and then unify source code and behavior verification. In addition, that kind of implementation usually employs digital filters with specific parameters, which could also be addressed. Indeed, communication aspects of ECPS could be tackled, as properties in system verification procedures.

ML techniques are already being used in the software verification area [233–236]. For instance, Hamel [233] proposed to induce a theory, through ML, that fits a test set for a program, in such a way that it implements the respective test set. Phuc [234], in turn, applied decision tree C4.5 and multi-layer perceptron to annotated programs (metrics related to structure and semantics) and tackled the generation of test sets satisfying desired constraints. Bridge, Holden, and Paulson [235] studied the automation of heuristic selection based on features, regarding first-order logic theorem proving, and Hutter *et. al* [236] showed that ML can be used for improving SAT solvers regarding huge verification tasks, when applied to parameter tuning [236]. Seshia *et al.* [237], in turn, showed that control synthesis based on ML is also a trend, since it has been applied to different applications and model classes [238, 239]. Finally, ML can also be used for invariant computation and strengthening, which is a major bottleneck in k-induction verification based on invariant algorithms. As a consequence of what was presented, it seems worthwhile to pursue integration of ML into BMC-based checkers, given that it has the potential to tackle many open problems or at least enhance existing solutions.

In addition, another aspect closely related to ML techniques, as mentioned in Section 6, is system fragility. Currently, most design strategies try to produce robust systems, which are able to deal with a number of errors that happen when they are operating. Nonetheless, that is effective only if problems foreseen during design phases happen; otherwise, operation may be severely compromised and even cause loss of lives.

Indeed, depending on the employed point of view, fragility and robustness may be regarded as the same property. A system that is more robust is less fragile, but it is still prone to problems, which depends on operating scenarios, use cases, and uncertainties. For instance, Keel and Bhatactacharyya [59] showed that robust and optimal control system can still be fragile, with respect to implementation issues. In summary, if the chosen model does not tackle every possible problem or error that might occur in a given environment, system-failure events are still possible, which is true even for the Ariane 5's failure [201].

As a result, it may be more interesting to develop methodologies that are not simply robust, regarding predetermined conditions, but instead thrive on problems and grow stronger (as the human body, which gets stronger when exposed to germs), which leads to anti-fragile systems. In engineering, we are only beginning to develop solutions that can be classified as anti-fragile, which can be carried out via analytical or physical redundancy to achieve performance enhancement [240]. For instance, multiple-input/multiple-output (MIMO) systems, which use multiple signal copies together with multi-path signal copies, would initially result in a destructive effect; however, with the use of a methodology that takes advantage on them, the obtained net result is performance improvement.

Systems that could greatly benefit from being anti-fragile are the cyber-physical ones, which interact with other external elements and use their internal representations to interpret physical properties. As faults occur, which may be treated with some sort of system elasticity, and external environments are normally dynamic, given that they naturally change with time, variations in operating conditions happen and should be tolerated, in order to keep behavior quality [201]. Besides, when discussing the fundamentals of anti-fragility, it is difficult not to tackle machine learning [201], which has the potential to provide some wisdom regarding correct behavior and internal representations, in order to keep systems working in volatile environments. An interesting scenario regarding that is the Sim3Tanks in Section 2.3, which could be able to cope with changes in fluid and reservoir characteristics, in such a way it adapts to different environmental conditions, while still keeping the chosen properties: ϕ_1, ϕ_2, ϕ_3, and ϕ_4.

For instance, we could think of a system including an anti-fragility module composed by two elements: a ML engine, capable of capturing current behaviors and identifying different trends, and a SMT-based verification method, in charge of recursively performing re-tuning procedures, which could test new parameters estimated from data provided by the machine learning module and then test new behaviors based on them, through counterexamples. In addition, there could also be a measurement module, which would judge some metrics regarding the new behavior (*e.g.*, signal-to-noise-error). As a consequence, we would be able to design systems with potential of being anti-fragile, given that they would learn from new challenges and then modify its behavior model, in order to account for new errors and threats and even benefit from them.

9 Limitations of this Survey

We have systematically carried out a literature review about symbolic verification and synthesis techniques for ECPS and also outlined our vision for future research. One of the major problems with systematic literature reviews is to find all relevant studies in the field; this process typically includes three stages: *searching*, *screening*, and *synthesis* [241]. All those three stages present limitations and bias, which we have tried to alleviate during our systematic review process. In particular, in order to minimize limitations regarding our searching stage, we have adopted an automated search mechanism via the main academic digital libraries, in order to find the most relevant studies regarding ECPS, namely, Google Scholar, Web of Science, and Scopus. In order to reduce any bias introduced by us, when using those digital libraries, we first performed a manual search of relevant papers addressed by peer-reviewed conferences and journals, in order to determine whether they had published any relevant (research) paper, related to symbolic verification and synthesis techniques for ECPS, in the last ten years, which we denoted as "control list". We have used the latter to validate publications returned by our search engines, in order to help us double-check execution of string searching in Google Scholar, Web of Science, and Scopus; however, these digital libraries might not cover the entire spectrum of papers published in the ECPS field. For future work, we will extend this survey, by including other digital libraries, with the goal of gathering new publications not yet indexed by them.

Additionally, we have carefully included criteria to screen potentially-relevant studies. In particular, our criteria include strings that encompass the main focus of our survey, *i.e.*, symbolic schemes to automatically verify and synthesize software for ECPS. Since the latter is a very broad research topic, we have restricted our search strings to keywords typically used in that field, *e.g.*, "Internet-of-Things", "Concurrent Systems", "Hybrid Systems", "Control Systems", and so on. Additionally, we have carefully discussed the screening process with each author of this survey, by commonly including studies based on our main goal, which is "automated symbolic verification and synthesis for ECPS". Nonetheless, we do agree that this screening process might still have some degree of subjectivity and we also identify it as a limitation of our survey. Lastly, the synthesis stage requires extraction of relevant studies to be described and evaluated in a survey, with the goal of explaining the state-of-the-art and identifying bottlenecks for future research. In our study, we have informally defined a protocol to describe definitions, search strings, search strategy, inclusion, and exclusion criteria, in order to make the synthesis stage feasible, regarding our main goal; however, this informal protocol represents a limitation of our survey, since they rely on searching and screening phases in which they are, by themselves, limited as described above.

10 Conclusions

This paper presented the main challenges related to verification of design correctness, in ECPS, and also raised some important side considerations about synthesis. In particular, we have covered six symbolic verification methods: BMC, k-induction, IC3, Craig interpolation, abstraction (including predicate abstraction and abstract interpretation), and path-based symbolic execution; we have also described various state-of-the-art verifiers that successfully implement these symbolic verification techniques. Additionally, we have emphasized one particular application of such symbolic verification methods that represents an approach beyond code and behavior verification, *i.e.*, formal ECPS synthesis.

In this respect, we pointed out that stringent constraints imposed by underlying hardware (*e.g.*, real-time, word length, memory allocation, interrupts, and concurrency), along with system behavior models, must be considered during verification and synthesis. Additionally, there is little evidence that model checking ECPS using k-induction, IC3, invariants, and learning, which extends promising symbolic-based approaches from falsification to verification, can be applied to formally verify correctness and timeliness of ECPS.

In addition, it is worth noticing that formal synthesis consists in designing systems that are correct, according to evaluation procedures performed by formal verification algorithms. Therefore, formal synthesis is dependent on formal verification and both are based on sound models of systems and properties.

Given that software complexity has significantly increased in ECPS, there are still some (recent) advances to stress and exhaustively cover system state space, in order to verify low-level properties that have to meet an application's deadline, access memory regions, handle concurrency, and control hardware registers. There is a trend towards incorporating knowledge about a system to be verified, which may take software verification and synthesis one step further, where not only code correctness will be addressed, but also full system reliability. In addition, it seems interesting to provide behavioral models, when new application interfaces or programming language features are used, in order to extend capabilities of current verification tools, without changing the core symbolic verification module. Finally, there are some bottlenecks in software verification that are possible to be handled with machine learning, given some initial studies carried out in the current literature, which may represent another step towards a unified and complete approach that combines code verification, behavior correctness, environment adaptation, and system evolution.

As future work, the main goal of this research is to extend symbolic verification as a design and synthesis tool for achieving correct-by-construction ECPS implementations. In particular, we noticed that the current literature lacks further studies that exploit the use of different symbolic verification methods beyond system code and behavior verification. Special attention will be given to 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 and synthesis, probably through behavior models. Additionally, ML techniques will be incorporated to symbolic verification frameworks, in order to provide flexibility and specific-problem tuning, or even allow system adaptation and behavior tuning.

11 Acknowledgments

We thank the anonymous reviewers for their comments, which helped us substantially improve the draft version of this paper. This research was supported in part by FAPEAM (Grant 062.00719/2016), in part by CNPq (Grant 432341/2018-8), and in part by the PROPG-CAPES/FAPEAM Scholarship Program (Grant/Award Number: 88887.217045/2018-00).

12 References

1 Kopetz, H.: 'Real-Time Systems: Design Principles for Distributed Embedded Applications'. Real-Time Systems Series. (Springer US, 2011)

2 Xu, X., Jia, H., Wang, D., Yu, D.C., Chiang, H.D.: 'Hierarchical energy management system for multi-source multi-product microgrids', *Renewable Energy*, 2015, **78**, pp. 621 – 630

3 Lee, E.A. 'Cyber physical systems: Design challenges'. In: 11th International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2008. pp. 363–369

4 Lee, E.A.: 'Computing Foundations and Practice for Cyber-Physical Systems: A Preliminary Report'. (EECS Department, University of California, Berkeley, 2007)

5 Lee, E.A.: 'The past, present and future of cyber-physical systems: A focus on models', *Sensors*, 2015, **3**, (15), pp. 4837–4869

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

7 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, 2009. pp. 396–403

8 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, 2013. p. 1

9 Abate, A., Bessa, I., Cattaruzza, D., Lucas, C., David, C., Kesseli, P., et al. 'Sound and automated synthesis of digital stabilizing controllers for continuous plants'. In: 20th International Conference on Hybrid Systems: Computation and Control, 2017. pp. 197–206

10 Witkowski, T., Blanc, N., Kroening, D., Weissenbacher, G. 'Model checking concurrent linux device drivers'. In: 22nd International Conference on Automated Software Engineering, 2007. pp. 501–504

11 Virtanen, S., Virtanen, S.: 'Advancing Embedded Systems and Real-Time Communications with Emerging Technologies'. 1st ed. (Hershey, PA, USA: IGI Global, 2014)

12 Huang, X., Kwiatkowska, M., Wang, S., Wu, M. 'Safety verification of deep neural networks'. In: Computer Aided Verification. vol. 10426 of *LNCS*, 2017. pp. 3–29

13 Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J. 'Reluplex: An efficient SMT solver for verifying deep neural networks'. In: Computer Aided Verification. vol. 10426 of *LNCS*, 2017. pp. 97–117

14 Buccafurri, F., Eiter, T., Gottlob, G., Leone, N.: 'Enhancing model checking in verification by AI techniques', *Artif Intell*, 1999, **112**, (1), pp. 57 – 104

15 Bortolussi, L., Milios, D., Sanguinetti, G. 'Machine learning methods in statistical model checking and system design – tutorial'. In: Bartocci, E., Majumdar, R., editors. Runtime Verification, 2015. pp. 323–341

16 Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., et al. 'Verification of markov decision processes using learning algorithms'. In: Automated Technology for Verification and Analysis, 2014. pp. 98–114

17 Jha, S., Seshia, S.A.: 'A theory of formal synthesis via inductive learning', *Acta Informatica*, 2017, **54**, (7), pp. 693–726

18 Kroening, D., Liang, L., Melham, T., Schrammel, P., Tautschnig, M. 'Effective verification of low-level software with nested interrupts'. In: Design, Automation & Test in Europe Conference & Exhibition, 2015. pp. 229–234

19 Pereira, P., Albuquerque, H., da Silva, I., Marques, H., Monteiro, F., Ferreira, R., et al.: 'SMT-based context-bounded model checking for CUDA programs', *Concurrency and Computation: Practice and Experience*, 2016,

20 Zheng, X., Julien, C. 'Verification and validation in cyber physical systems: Research challenges and a way forward'. In: 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems, 2015. pp. 1–4

21 Clarke, E.M., Zuliani, P. 'Statistical model checking for cyber-physical systems'. In: 9th International Conference on Automated Technology for Verification and Analysis, 2011. pp. 1–12

22 Bradley, A.R., Manna, Z.: 'The Calculus of Computation: Decision Procedures with Applications to Verification'. (Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007)

23 Nilsson, P., Hussien, O., Balkan, A., Chen, Y., Ames, A.D., Grizzle, J.W., et al.: 'Correct-by-construction adaptive cruise control: Two approaches', *IEEE Transactions on Control Systems Technology*, 2016, **24**, (4), pp. 1294–1307

24 Prabhakar, P., García.Soto, M. 'Formal synthesis of stabilizing controllers for switched systems'. In: 20th International Conference on Hybrid Systems: Computation and Control, 2017. pp. 111–120

25 Esmaeil.Zadeh.Soudjani, S., Majumdar, R. 'Controller synthesis for reward collecting markov processes in continuous space'. In: 20th International Conference on Hybrid Systems: Computation and Control, 2017. pp. 45–54

26 Ames, A.D., Tabuada, P., Jones, A., Ma, W.L., Rungger, M., Schürmann, B., et al.: 'First steps toward formal controller synthesis for bipedal robots with experimental implementation', *Nonlinear Analysis: Hybrid Systems*, 2017, **25**, pp. 155 – 173

27 Tabuada, P.: 'Verification and Control of Hybrid Systems: A Symbolic Approach'. (Springer US, 2009)

28 Hasuo, I.: 'Metamathematics for systems design', *New Generation Computing*, 2017, **35**, (3), pp. 271–305

29 Zamani, M., Abate, A., Girard, A.: 'Symbolic models for stochastic switched systems: A discretization and a discretization-free approach', *Automatica*, 2015, **55**, pp. 183 – 196

30 Khoo, T.P. 'Model-based testing of cyber-physical systems'. In: International Conference on Formal Engineering Methods, 2018. pp. 423–426

31 Zhang, Y., Dong, Y., Xie, F. 'Bounded model checking of hybrid automata pushdown system'. In: 14th International Conference on Quality Software, 2014. pp. 190–195

32 Simko, G., Jackson, E.K. 'A bounded model checking tool for periodic sample-hold systems'. In: 17th International Conference on Hybrid Systems: Computation and Control, 2014. pp. 157–162

33 Sanwal, M.U., Hasan, O. 'Formal verification of cyber-physical systems: Coping with continuous elements'. In: 13th International Conference on Computational Science and Its Applications. vol. 7971 of *LNCS*, 2013. pp. 358–371

34 Lee, H.Y. 'Towards model checking of simulation models for embedded system development'. In: International Conference on Parallel and Distributed Systems, 2013. pp. 452–453

35 Li, T., Tan, F., Wang, Q., Bu, L., Cao, J.N., Liu, X.: 'From offline toward real time: A hybrid systems model checking and CPS codesign approach for medical device plug-and-play collaborations', *IEEE Transactions on Parallel and Distributed Systems*, 2014, **25**, (3), pp. 642–652

36 Jhala, R., Majumdar, R.: 'Software model checking', *ACM Computing Surveys*, 2009, **41**, (4), pp. 21:1–21:54

37 Baier, C., Katoen, J.: 'Principles of model checking'. (MIT Press, 2008)

38 Kroening, D., Strichman, O.: 'Decision Procedures - An Algorithmic Point of View, Second Edition'. Texts in Theoretical Computer Science. An EATCS Series. (Springer, 2016)

39 Rungger, M., Tabuada, P.: 'A notion of robustness for cyber-physical systems', *IEEE Transactions on Automatic Control*, 2016, **61**, (8), pp. 2108–2123

40 Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: 'Discrete abstractions of hybrid systems', *Proceedings of the IEEE*, 2000, **88**, (7), pp. 971–984

41 Girard, A., Pappas, G.J.: 'Approximate bisimulation: A bridge between computer science and control theory', *European Journal of Control*, 2011, **17**, (5), pp. 568 – 578

42 Lunze, J., Lamnabhi.Lagarrigue, F.: 'Handbook of Hybrid Systems Control: Theory, Tools, Applications'. (Cambridge University Press, 2009)

43 Bargmann, H.: 'The role of stochastic modelling in engineering science', *Acta Mechanica*, 1997, **125**, (1), pp. 63–71

44 Alur, R., Dill, D.L.: 'A theory of timed automata', *Theoretical Computer Science*, 1994, **126**, (2), pp. 183–235

45 Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H. 'Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems'. In: Hybrid Systems, 1993. pp. 209–229

46 Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P. 'What's decidable about hybrid automata?'. In: 27th Annual Symposium on Theory of Computing, 1995. pp. 373–382

47 Julius, A.A., Pappas, G.J.: 'Approximations of stochastic hybrid systems', *IEEE Transactions on Automatic Control*, 2009, **54**, (6), pp. 1193–1203

48 Pola, G., Bujorianu, M.L., Lygeros, J., Benedetto, M.D.D.: 'Stochastic hybrid models: An overview', *IFAC Proceedings Volumes*, 2003, **36**, (6), pp. 45 – 50

49 Lafferriere, G., Pappas, G.J., Yovine, S. 'A new class of decidable hybrid systems'. In: 2nd International Conference on Hybrid Systems: Computation and Control, 1999. pp. 137–151

50 Puri, A., Varaiya, P. 'Decidability of hybrid systems with rectangular differential inclusion'. In: Computer Aided Verification, 1994. pp. 95–104

51 Henzinger, T.A., Ho, P.H. 'Algorithmic analysis of nonlinear hybrid systems'. In: Computer Aided Verification, 1995. pp. 225–238

52 Broucke, M.E., Varaiya, P. 'Decidability of hybrid systems with linear and nonlinear differential inclusions'. In: 4th International Workshop on Hybrid Systems. vol. 1273 of *LNCS*, 1996. pp. 77–92

53 Kesten, Y., Pnueli, A., Sifakis, J., Yovine, S. 'Integration graphs: A class of decidable hybrid systems'. In: Hybrid Systems. vol. 736 of *LNCS*, 1992. pp. 179–208

54 Tabuada, P., Caliskan, S.Y., Rungger, M., Majumdar, R.: 'Towards robustness for cyber-physical systems', *IEEE Transactions on Automatic Control*, 2014, **59**, (12), pp. 3151–3163

55 Rungger, M., Mazo, M. Jr., Tabuada, P. 'Specification-guided controller synthesis for linear systems and safe linear-time temporal logic'. In: 16th International Conference on Hybrid Systems: Computation and Control, 2013. pp. 333–342

56 Li, Y., Liu, J. 'An interval analysis approach to invariance control synthesis for discrete-time switched systems'. In: 55th Conference on Decision and Control, 2016. pp. 6388–6394

57 Tabuada, P.: 'Symbolic models for control systems', *Acta Informatica*, 2007, **43**, (7), pp. 477–500

58 Istepanian, R.S.H., Whidborne, J.F., editors. 'Digital Controller Implementation and Fragility: A Modern Perspective'. (London: Springer, 2001)

59 Keel, L.H., Bhattacharyya, S.P.: 'Robust, fragile, or optimal?', *IEEE Transactions on Automatic Control*, 1997, **42**, (8), pp. 1098–1105

60 Bicchi, A., Marigo, A., Piccoli, B.: 'On the reachability of quantized control systems', *IEEE Transactions on Automatic Control*, 2002, **47**, (4), pp. 546–563

61 Petreczky, M., van Schuppen, J.H.: 'Realization theory for linear hybrid systems', *IEEE Transactions on Automatic Control*, 2010, **55**, (10), pp. 2282–2297

62 Petreczky, M.: 'Realization theory for linear switched systems: Formal power series approach', *Systems & Control Letters*, 2007, **56**, (9), pp. 588 – 595

63 Ye, H., Michel, A.N., Hou, L.: 'Stability theory for hybrid dynamical systems', *IEEE Transactions on Automatic Control*, 1998, **43**, (4), pp. 461–474

64 Ben.Sassi, M.A., Girard, A.: 'Computation of polytopic invariants for polynomial dynamical systems using linear programming', *Automatica*, 2012, **48**, (12), pp. 3114–3121

65 Paul, T., Kimball, J.W., Zawodniok, M., Roth, T.P., McMillin, B., Chellappan, S.: 'Unified invariants for cyber-physical switched system stability', *IEEE Transactions on Smart Grid*, 2014, **5**, (1), pp. 112–120

66 Li, Y., Liu, J. 'Computing maximal invariant sets for switched nonlinear systems'. In: Conference on Computer Aided Control System Design, 2016. pp. 862–867

67 Fisher, A., Jacobson, C.A., Lee, E.A., Murray, R.M., Sangiovanni.Vincentelli, A., Scholte, E. 'Industrial cyber-physical systems – iCyPhy'. In: Complex Systems Design & Management, 2014. pp. 21–37

68 Farias, A.O., Queiroz, G.A.C., Bessa, I.V., Medeiros, R.L.P., Cordeiro, L.C., Palhares, R.M.: 'Sim3tanks: A benchmark model simulator for process control and monitoring', *IEEE Access*, 2018, **6**, pp. 62234–62254

69 Song, H., Rawat, D.B., Jeschke, S., Brecher, C.: 'Cyber-Physical Systems: Foundations, Principles and Applications'. (Hershey, PA, USA: Academic Press, 2017)

70 Chaves, L., Bessa, I., Ismail, H., dos Santos.Frutuoso, A.B., Cordeiro, L.C., de Lima.Filho, E.B.: 'DSVerifier-Aided verification applied to attitude control software in unmanned aerial vehicles', *IEEE Transactions on Reliability*, 2018, **67**, (4), pp. 1420–1441

71 McMillan, K.L.: 'Symbolic Model Checking'. (Norwell, MA, USA: Kluwer Academic Publishers, 1993)

72 Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: 'NUSMV: a new symbolic model checker', *International Journal on Software Tools for Technology Transfer*, 2000, **2**, (4), pp. 410–425

73 Cimatti, A., Mover, S., Tonetta, S. 'A quantifier-free SMT encoding of nonlinear hybrid automata'. In: Formal Methods in Computer-Aided Design, 2012. pp. 187–195

74 Platzer, A. 'Differential dynamic logic for verifying parametric hybrid systems'. In: 16th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, 2007. pp. 216–232

75 Pkatzer, A. 'Logic & proofs for cyber-physical systems'. In: 8th International Joint Conference on Automated Reasoning, 2016. pp. 15–21

76 Platzer, A., Quesel, J.D. 'KeYmaera: a hybrid theorem prover for hybrid systems'. In: International Joint Conference on Automated Reasoning, 2008. pp. 171–178

77 Sanwal, M.U., Hasan, O. 'Formal verification of cyber-physical systems: Coping with continuous elements'. In: International Conference on Computational Science and Its Applications. vol. 1, 2013. pp. 358–371

78 Li, B.: 'Wireless Cyber-Physical Simulator and Case Studies on Structural Control'. Master of Science Thesis. (Washington University in St. Louis, 2013)

79 Canadasa, N., Machado, J., Soares, F., Barros, C., Varela, L.: 'Simulation of cyber physical systems behaviour using timed plant models', *Mechatronics*, 2018, **54**, pp. 175–185

80 Gerdsmeier, T., Cardell.Oliver, R.: 'Analysis of scheduling behaviour using generic timed automata', *Electronic Notes in Theoretical Computer Science*, 2001, **42**, pp. 143–157

81 Junjie, T., Jianjun, Z., Jianwan, D., Liping, C., Gang, X., Bin, G., et al. 'Cyber-physical systems modeling method based on Modelica'. In: 6th International Conference on Software Security and Reliability Companion, 2012. pp. 188–191

82 Thacker, R.A., Jones, K.R., Myers, C.J., Zheng, H. 'Automatic abstraction for verification of cyber-physical systems'. In: 1st International Conference on Cyber-Physical Systems, 2010. pp. 12–21

83 Ishigooka, T., Saissi, H., Piper, T., Winter, S., Suri, N. 'Practical formal verification for model based development of cyber-physical systems'. In: International Conference on Computational Science and Engineering, 2016. pp. 1–6

84 Radojicic, C., Grimm, C., Jantsch, A., Rathmair, M. 'Towards verification of uncertain cyber-physical systems'. In: 3nd International Workshop on Symbolic and Numerical Methods for Reachability Analysis, 2017. pp. 1–17

85 Majumdar, R., Saha, I., Shashidhar, K.C., Wang, Z. 'CLSE: closed-loop symbolic execution'. In: NASA Formal Methods Symposium. vol. 7226 of *LNCS*, 2012. pp. 356–370

86 Borda, A., Pasquale, L., Koutavas, V., Nuseibeh, B. 'Compositional verification of self-adaptive cyber-physical systems'. In: 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2018. pp. 1–11

87 Brings, J. 'Verifying cyber-physical system behavior in the context of cyber-physical system-networks'. In: 25th International Requirements Engineering Conference, 2017. pp. 556–561

88 Kang, E.Y., Mu, D., Huang, L., Lan, Q. 'Verification and validation of a cyber-physical system in the automotive domain'. In: International Conference on Software Quality, Reliability and Security, 2017. pp. 326–333

89 David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: 'Uppaal smc tutorial', *International Journal on Software Tools for Technology Transfer*, 2015, **17**, (4), pp. 397–415

90 Silva, L.C., Almeida, H.O., Perkusich, A., Perkusich, M.: 'A model-based approach to support validation of medical cyber-physical systems', *Sensors*, 2015, **15**, (11), pp. 27625–27670

91 Nelson, A., Chakraborty, S., Wang, D., Singh, P., Cui, Q., Yang, L., et al. 'Cyber-physical test platform for microgrids: Combining hardware, hardware-in-the-loop, and network-simulator-in-the-loop'. In: IEEE Power and Energy Society General Meeting, 2016. pp. 1–5

92 Klein, G., Andronick, J., Fernandez, M., Kuz, I., Murray, T.C., Heiser, G.: 'Formally verified software in the real world', *Communications of the ACM*, 2018, **61**, (10), pp. 68–77

93 Alur, R. 'Formal verification of hybrid systems'. In: 9th International Conference on Embedded Software, 2011. pp. 273–278

94 Prajna, S., Jadbabaie, A., Pappas, G.J.: 'A framework for worst-case and stochastic safety verification using barrier certificates', *IEEE Transactions on Automatic Control*, 2007, **52**, (8), pp. 1415–1428

95 Prajna, S., Jadbabaie, A. 'Safety verification of hybrid systems using barrier certificates'. In: 7th International Conference on Hybrid Systems: Computation and Control. vol. 2993 of *LNCS*, 2004. pp. 477–492

96 Bessa, I., Ismail, H., Palhares, R., Cordeiro, L., Filho, J.E.C.: 'Formal non-fragile stability verification of digital control systems with uncertainty', *IEEE Transactions on Computers*, 2017, **66**, (3)

97 Maler, O., Nickovic, D. 'Monitoring temporal properties of continuous signals'. In: International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS, 2004. pp. 152–166

98 Donzé, A., Maler, O., Bartocci, E., Nickovic, D., Grosu, R., Smolka, S. 'On temporal logic and signal processing'. In: 10th International Conference on Automated Technology for Verification and Analysis, 2012. pp. 92–106

99 Veanes, M., Bjorner, N., Gurevich, Y., Schulte, W.: 'Symbolic bounded model checking of abstract state machines', *International Journal of Software and Informatics*, 2009, **3/2-3**, pp. 149–170

100 Phan, A.D.: 'Modelling and Analysis for Cyber-Physical Systems: An SMT-based approach'. (Technical University of Denmark (DTU, 2015)

101 Nipkow, T., Wenzel, M., Paulson, L.C.: 'Isabelle/HOL: A Proof Assistant for Higher-order Logic'. (Berlin, Heidelberg: Springer-Verlag, 2002)

102 Biere, A. 'Bounded model checking'. In: Handbook of Satisfiability, 2009. pp. 457–481

103 Duggirala, P.S., Viswanathan, M. 'Analyzing real time linear control systems using software verification'. In: IEEE Real-Time Systems Symposium, 2015. pp. 216–226

104 Anta, A., Majumdar, R., Saha, I., Tabuada, P. 'Automatic verification of control system implementations'. In: 10th International Conference on Embedded Software, 2010. pp. 9–18

105 Ismail, H., Cordeiro, I.B.L., Filho, E.B.L., ao Edgar.Chaves.Filho, J. 'DSVerifier: A bounded model checking tool for digital systems'. In: 22nd International SPIN Workshop on Model Checking of Software. vol. 9232 of *LNCS*, 2015. pp. 126–131

106 Bessa, I.V., Ismail, H.I., Cordeiro, L.C., Filho, J.E.C.: 'Verification of fixed-point digital controllers using direct and delta forms realizations', *Design Autom for Emb Sys*, 2016, **20**, (2), pp. 95–126

107 Phan, A., Hansen, M.R., Madsen, J. 'EHRA: specification and analysis of energy-harvesting wireless sensor networks'. In: Specification, Algebra, and Software. vol. 8373 of *LNCS*, 2014. pp. 520–540

108 Nakajima, S., Furukawa, S., Ueda, Y. 'Co-analysis of sysml and simulink models for cyber-physical systems design'. In: International Conference on Embedded and Real-Time Computing Systems and Applications, 2012. pp. 473–478

109 Shoukry, Y., Nuzzo, P., Puggelli, A., Sangiovanni.Vincentelli, A.L., Seshia, S.A., Tabuada, P.: 'Secure state estimation for cyber physical systems under sensor attacks: A satisfiability modulo theory approach', *IEEE Transactions on Automatic Control*, 2017, **PP**, (99), pp. 1–1

110 Liu, Y., Ning, P., Reiter, M.K. 'False data injection attacks against state estimation in electric power grids'. In: 16th Conference on Computer and Communications Security, 2009. pp. 21–32

111 Choo, K.K.R., Kermani, M.M., Azarderakhsh, R., Govindarasu, M.: 'Emerging embedded and cyber physical system security challenges and innovations', *IEEE Transactions on Dependable and Secure Computing*, 2017, **14**, (3), pp. 235

112 Choo, V.P.I., noz González, L.M., Lupu, E.C.: 'Don't fool me!: Detection, characterisation and diagnosis of spoofed and masked events in wireless sensor networks', *IEEE Transactions on Dependable and Secure Computing*, 2017, **14**, (3), pp. 279–293

113 Fiore, G., Chang, Y.H., Hu, Q., Benedetto, M.D.D., Tomlin, C.J. 'Secure state estimation for cyber physical systems with sparse malicious packet drops'. In: American Control Conference, 2017. pp. 1898–1903

114 Araújo, R.F., Albuquerque, H.F., de Bessa, I.V., Cordeiro, L.C., ao E..Chaves.Filho, J.: 'Counterexample guided inductive optimization based on satisfiability modulo theories', *Science of Computer Programming*, 2017,

115 Trindade, A.B., Cordeiro, L.C.: 'Applying SMT-based verification to hardware/-software partitioning in embedded systems', *Design Automation for Embedded Systems*, 2016, **20**, (1), pp. 1–19

116 Rahman, M.A., Duan, Q., Al.Shaer, E. 'Energy efficient navigation management for hybrid electric vehicles on highways'. In: International Conference on Cyber-Physical Systems, 2013. pp. 21–30

117 Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y. 'Symbolic model checking without bdds'. In: 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, 1999. pp. 193–207

118 Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C. 'Satisfiability modulo theories'. In: Handbook of Satisfiability, 2009. pp. 825–885

119 Armando, A., Mantovani, J., Platania, L.: 'Bounded model checking of software using SMT solvers instead of SAT solvers', *International Journal on Software Tools for Technology Transfer*, 2009, **11**, (1), pp. 69–83

120 Prasad, M.R., Biere, A., Gupta, A.: 'A survey of recent advances in SAT-based formal verification', *International Journal on Software Tools for Technology Transfer*, 2005, **7**, (2), pp. 156–173

121 Morse, J., Cordeiro, L., Nicole, D., Fischer, B.: 'Model Checking LTL Properties over ANSI-C Programs with Bounded Traces', *Software & Systems Modeling*, 2015, **14**, (1), pp. 65–81

122 Ball, T., Rajamani, S.: 'SLIC: A Specification Language for Interface Checking (of C)'. (Microsoft Research, 2002). available at: https://www.microsoft.com/en-us/research/publication/slic-a-specification-language-for-interface-checking-of-c/

123 Kroening, D., Strichman, O.: 'Decision Procedures: An Algorithmic Point of View'. 1st ed. (Springer Publishing Company, Incorporated, 2008)

124 Appel, A.W.: 'Modern Compiler Implementation in C: Basic Techniques'. (New York, NY, USA: Cambridge University Press, 1997)

125 Siekmann, J.H., Wrightson, G., editors. 'Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970'. (Berlin, Heidelberg: Springer Berlin Heidelberg, 1983)

126 Patarin, J., Goubin, L. 'Trapdoor one-way permutations and multivariate polynominals'. In: First International Conference on Information and Communication Security. vol. 1334 of *LNCS*, 1997. pp. 356–368

127 Clarke, E.M., Kroening, D., Lerda, F. 'A tool for checking ANSI-C programs'. In: 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. vol. 2988 of *LNCS*, 2004. pp. 168–176

128 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. vol. 7152 of *LNCS*, 2012. pp. 146–161

129 Cordeiro, L., Fischer, B. 'Verifying multi-threaded software using SMT-based context-bounded model checking'. In: 33rd International Conference on Software Engineering, 2011. pp. 331–340

130 Ivančić, F., Shlyakhter, I., Gupta, A., Ganai, M.K. 'Model checking c programs using F-SOFT'. In: International Conference on Computer Design, 2005. pp. 297–308

131 Cordeiro, L., Fischer, B., Marques.Silva, J.: 'SMT-based bounded model checking for embedded ANSI-C software', *IEEE Transactions on Software Engineering*, 2012, **38**, (4), pp. 957–974

132 Clarke, E., Kroening, D., Strichman, O., Ouaknine, J. 'Completeness and complexity of bounded model checking'. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. vol. 2937 of *LNCS*, 2004. pp. 85–96

133 Ganai, M.K., Gupta, A. 'Completeness in SMT-based BMC for software programs'. In: Design, Automation and Test in Europe, 2008. pp. 831–836

134 Eén, N., Sörensson, N. 'An extensible sat-solver'. In: 6th International Conference on Theory and Applications of Satisfiability Testing. vol. 2919 of *LNCS*, 2003. pp. 502–518

135 Sheeran, M., Singh, S., Stålmarck, G. 'Checking safety properties using induction and a sat-solver'. In: Formal Methods in Computer-Aided Design, 2000. pp. 108–125

136 IEEE: 'IEEE standard for floating-point arithmetic', *Std 754-2008*, 2008, pp. 1–70

137 Goldberg, D.: 'What every computer scientist should know about floating-point arithmetic', *ACM Computing Surveys*, 1991, **23**, (1), pp. 5–48

138 CRI (MINES ParisTech). 'PIPS: Automatic Parallelizer and Code Transformation Framework'. (`https://pips4u.org/`, 2016). accessed 21st of February 2016

139 Henry, J., Monniaux, D., Moy, M.: 'PAGAI: A path sensitive static analyser', *Electronic Notes in Theoretical Computer Science*, 2012, **289**, pp. 15–25

140 Beyer, D., Dangl, M., Wendler, P. 'Boosting k-induction with continuously-refined invariants'. In: Computer Aided Verification. vol. 906 of *LNCS*, 2015. pp. 622–640

141 Gadelha, M.Y.R., Ismail, H.I., Cordeiro, L.C.: 'Handling loops in bounded model checking of c programs via k-induction', *International Journal on Software Tools for Technology Transfer*, 2017, **19**, (1), pp. 97–114

142 Brain, M., Joshi, S., Kroening, D., Schrammel, P. 'Safety verification and refutation by k-invariants and k-induction'. In: International Static Analysis Symposium. vol. 9291 of *LNCS*, 2015. pp. 145–161

143 Donaldson, A.F., Haller, L., Kroening, D., Rümmer, P. 'Software verification using k-induction'. In: International Static Analysis Symposium. vol. 6887 of *LNCS*, 2011. pp. 351–368

144 Rocha, W., Rocha, H., Ismail, H., Cordeiro, L.C., Fischer, B. 'Depthk: A k-induction verifier based on invariant inference for C programs - (competition contribution)'. In: 23rd International Conference on Tools and Algorithms for the

Construction and Analysis of Systems. vol. 10206 of *LNCS*, 2017. pp. 360–364

145 Donaldson, A.F., Kroening, D., Ruemmer, P. 'SCRATCH: A tool for automatic analysis of DMA races'. In: 16th Symposium on Principles and Practice of Parallel Programming, 2011. pp. 311–312

146 Grosse, D., Le, H.M., Drechsler, R. 'Induction-based formal verification of SystemC TLM designs'. In: 10th International Workshop on Microprocessor Test and Verification, 2009. pp. 101–106

147 Bradley, A.R. 'IC3 and beyond: Incremental, inductive verification'. In: Computer Aided Verification. vol. 7358 of *LNCS*, 2012. pp. 4–4

148 Hassan, Z., Bradley, A.R., Somenzi, F. 'Better generalization in IC3'. In: Formal Methods in Computer-Aided Design, 2013. pp. 157–164

149 Bradley, A.R.: 'Understanding IC3'. (Technical Report. Accessed on February 23rd 2018: ECEE Department, University of Colorado at Boulder, 2018)

150 Jovanović, D., Dutertre, B. 'Property-directed k-induction'. In: Formal Methods in Computer-Aided Design, 2016. pp. 85–92

151 McMillan, K.L. 'Interpolation and sat-based model checking'. In: Computer Aided Verification. vol. 2725 of *LNCS*, 2003. pp. 1–13

152 McMillan, K.L. 'Applications of craig interpolants in model checking'. In: 11th International Conference on Theory and Practice of Software. vol. 3440 of *LNCS*, 2005. pp. 1–12

153 Clarke, E.M., Grumberg, O., Long, D.E.: 'Model checking and abstraction', *ACM Transactions on Programming Languages and Systems*, 1994, **16**, (5), pp. 1512–1542

154 Flanagan, C., Qadeer, S. 'Predicate abstraction for software verification'. In: 29th Annual Symposium on Principles of Programming Languages, 2002. pp. 191–202

155 Clarke, E.M., Kroening, D., Sharygina, N., Yorav, K. 'SATABS: sat-based predicate abstraction for ANSI-C'. In: 11th International Conference on Theory and Practice of Software. vol. 3440 of *LNCS*, 2005. pp. 570–574

156 Cousot, P., Cousot, R. 'Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints'. In: 4th Symposium on Principles of Programming Languages, 1977. pp. 238–252

157 Nguyen, T.L., Fischer, B., La Torre, S., Parlato, G. 'Concurrent program verification with lazy sequentialization and interval analysis'. In: 5th International Conference on Networked Systems. vol. 10299 of *LNCS*, 2017. pp. 255–271

158 Monniaux, D. 'Compositional analysis of floating-point linear numerical filters'. In: Computer Aided Verification. vol. 3576 of *LNCS*, 2005. pp. 199–212

159 Munier, P.: 'Polyspace'. (Wiley-ISTE, 2013)

160 Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: 'Frama-C: A software analysis perspective', *Formal Aspects of Computing*, 2015, **27**, (3), pp. 573–609

161 King, J.C.: 'Symbolic Execution And Program Testing', *Communications of the ACM*, 1976, **19**, (7), pp. 385–394

162 Cadar, C., Dunbar, D., Engler, D. 'KLEE: Unassisted And Automatic Generation Of High-coverage Tests For Complex Systems Programs'. In: Symposium On Operating Systems Design And Implementation, 2008. pp. 209–224

163 Godefroid, P. 'Compositional Dynamic Test Generation'. In: Symposium On Principles Of Programming Languages, 2007. pp. 47–54

164 Solar-Lezama, A., Tancau, L., Bodík, R., Seshia, S.A., Saraswat, V.A. 'Combinatorial sketching for finite programs'. In: 12th international conference on Architectural support for programming languages and operating systems, 2006. pp. 404–415

165 Riener, H., Könighofer, R., Fey, G., Bloem, R. 'SMT-based CPS parameter synthesis'. In: 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems, 2016. pp. 126–133

166 Abate, A., Bessa, I., Cattaruzza, D., Cordeiro, L.C., David, C., Kesseli, P., et al. 'Automated formal synthesis of digital controllers for state-space physical plants'. In: Computer Aided Verification. vol. 10426 of *LNCS*, 2017. pp. 462–482

167 Ravanbakhsh, H., Sankaranarayanan, S. 'Counterexample-guided stabilization of switched systems using control Lyapunov functions'. In: 18th International Conference on Hybrid Systems: Computation and Control, 2015. pp. 297–298

168 Ravanbakhsh, H., Sankaranarayanan, S. 'Robust controller synthesis of switched systems using counterexample guided framework'. In: 13th International Conference on Embedded Software, 2016. pp. 8:1–8:10

169 Gol, E.A., Lazar, M., Belta, C.: 'Language-guided controller synthesis for linear systems', *IEEE Transactions on Automatic Control*, 2014, **59**, (5), pp. 1163–1176

170 Holub, O., Zamani, M., Abate, A. 'Efficient HVAC controls: A symbolic approach'. In: European Control Conference, 2016. pp. 1159–1164

171 Tabuada, P.: 'An approximate simulation approach to symbolic control', *IEEE Transactions on Automatic Control*, 2008, **53**, (6), pp. 1406–1418

172 Zamani, M., Arcak, M.: 'Compositional abstraction for networks of control systems: A dissipativity approach', *IEEE Transactions on Control of Network Systems*, 2018, **5**, (3), pp. 1003–1015

173 Zamani, M., Abate, A.: 'Approximately bisimilar symbolic models for randomly switched stochastic systems', *Systems & Control Letters*, 2014, **69**, pp. 38 – 46

174 Zamani, M., van de Wouw, N., Majumdar, R.: 'Backstepping controller synthesis and characterizations of incremental stability', *Systems & Control Letters*, 2013, **62**, (10), pp. 949 – 962

175 Zamani, M., Pola, G., Mazo, M., Tabuada, P.: 'Symbolic models for nonlinear control systems without stability assumptions', *IEEE Transactions on Automatic Control*, 2012, **57**, (7), pp. 1804–1809

176 Khatib, M.A., Girard, A., Dang, T.: 'Stability verification and timing contract synthesis for linear impulsive systems using reachability analysis', *Nonlinear Analysis: Hybrid Systems*, 2017, **25**, pp. 211 – 226

177 Lesser, K., Abate, A.: 'Controller synthesis for probabilistic safety specifications using observers', *IFAC-PapersOnLine*, 2015, **48**, (27), pp. 329 – 334

178 Girard, A.: 'Low-complexity quantized switching controllers using approximate bisimulation', *Nonlinear Analysis: Hybrid Systems*, 2013, **10**, pp. 34 – 44

179 Dallal, E., Colombo, A., Del.Vecchio, D., Lafortune, S.: 'Supervisory control for collision avoidance in vehicular networks using discrete event abstractions', *Discrete Event Dynamic Systems*, 2017, **27**, (1), pp. 1–44

180 Dallal, E., Colombo, A., Vecchio, D.D., Lafortune, S. 'Supervisory control for collision avoidance in vehicular networks with imperfect measurements'. In: 52nd Conference on Decision and Control, 2013. pp. 6298–6303

181 Habets, L.C.G.J.M., Collins, P.J., van Schuppen, J.H.: 'Reachability and control synthesis for piecewise-affine hybrid systems on simplices', *IEEE Transactions on Automatic Control*, 2006, **51**, (6), pp. 938–948

182 Reissig, G., Weber, A., Rungger, M.: 'Feedback refinement relations for the synthesis of symbolic controllers', *IEEE Transactions on Automatic Control*, 2017, **62**, (4), pp. 1781–1796

183 David, C., Kroening, D., Lewis, M. 'Using program synthesis for program analysis'. In: 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, 2015. pp. 483–498

184 Alur, R., Bodik, R., Juniwal, G., Martin, M.M.K., Raghothaman, M., Seshia, S.A., et al. 'Syntax-guided synthesis'. In: Formal Methods in Computer-Aided Design, 2013. pp. 1–8

185 Solar.Lezama, A.: 'Program sketching', *International Journal on Software Tools for Technology Transfer*, 2013, **15**, (5), pp. 475–495

186 Sharma, R., Aiken, A. 'From invariant checking to invariant inference using randomized search'. In: Computer Aided Verification, 2014. pp. 88–105

187 Abate, A., Bessa, I., Cattaruzza, D., Chaves, L., Cordeiro, L.C., David, C., et al. 'Dssynth: an automated digital controller synthesis tool for physical plants'. In: 32nd International Conference on Automated Software Engineering, 2017. pp. 919–924

188 Moore, R.E.: 'Interval analysis'. vol. 4. (Prentice-Hall, 1966)

189 Fairley, P.: 'Self-driving cars have a bicycle problem [news]', *IEEE Spectrum*, 2017, **54**, (3), pp. 12–13

190 Bortolussi, L., Milios, D., Sanguinetti, G.: 'Smoothed model checking for uncertain continuous-time markov chains', *Information and Computation*, 2016, **247**, (C), pp. 235–253

191 Behrend, J., Lettnin, D., Grünhage, A., Ruf, J., Kropf, T., Rosenstiel, W.: 'Scalable and optimized hybrid verification of embedded software', *Journal of Electronic Testing*, 2015, **31**, (2), pp. 151–166

192 Lettnin, D., Nalla, P.K., Behrend, J., Ruf, J., Gerlach, J., Kropf, T., et al. 'Semiformal verification of temporal properties in automotive hardware dependent software'. In: Design, Automation Test in Europe Conference Exhibition, 2009. pp. 1214–1217

193 Chaves, L.C., Ismail, H.I., Bessa, I.V., Cordeiro, L.C., de Lima.Filho, E.B.: 'Verifying fragility in digital systems with uncertainties using DSVerifier v2.0', *Journal of Systems and Software*, 2019, **153**, pp. 22–43

194 Beg, O.A., Johnson, T.T., Davoudi, A.: 'Detection of false-data injection attacks in cyber-physical dc microgrids', *IEEE Transactions on Industrial Informatics*, 2017, **13**, (5), pp. 2693–2703

195 Witkowski, T.: 'Formal Verification of Linux Device Drivers'. Master of Science Thesis. (Technishe Universiät Dresden, 2007)

196 Beckert, B., Bormer, T., Grahl, D. 'Deductive verification of legacy code'. In: International Symposium on Leveraging Applications of Formal Methods, 2016. pp. 749–765

197 Monteiro, F.R., Garcia, M., Cordeiro, L.C., de Lima.Filho, E.B.: 'Bounded model checking of C++ programs based on the qt cross-platform framework', *Software Testing, Verification and Reliability*, 2017, **27**, (3)

198 Li, H., Oh, J., Oh, H., Lee, H. 'Automated source code instrumentation for verifying potential vulnerabilities'. In: International Conference on ICT Systems Security and Privacy Protection, 2016. pp. 211–226

199 Vilca, J., Adouane, L., Mezouar, Y.: 'Optimal multi-criteria waypoint selection for autonomous vehicle navigation in structured environment', *Journal of Intelligent & Robotic Systems*, 2016, **82**, (2), pp. 301–324

200 van Wesel, P., Goodloe, A.E.: 'Challenges in the Verification of Reinforcement Learning Algorithms'. (National Aeronautics and Space Administration, NASA STI Program, 2017)

201 De.Florio, V. 'Antifragility = elasticity + resilience + machine learning'. In: 1st International Workshop From Dependable to Resilient, From Resilient to Antifragile Ambients and Systems. Procedia Computer Science, 2014. pp. 834–841

202 Audemard, G., Lagniez, J., Simon, L. 'Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction'. In: 16th International Conference on Theory and Applications of Satisfiability Testing. vol. 7962 of *LNCS*, 2013. pp. 309–317

203 Schrammel, P., Kroening, D., Brain, M., Martins, R., Teige, T., Bienmüller, T.: 'Incremental bounded model checking for embedded software', *Formal Aspects of Computing*, 2017, **29**, (5), pp. 911–931

204 Ramalho, M., Freitas, M., Sousa, F., Marques, H., Cordeiro, L., Fischer, B. 'SMT-based bounded model checking of c++ programs'. In: 20th International Conference and Workshops on the Engineering of Computer Based Systems, 2013. pp. 147–156

205 Wachter, B., Kroening, D., Ouaknine, J. 'Verifying multi-threaded software with impact'. In: Formal Methods in Computer-Aided Design, 2013. pp. 210–217

206 Beyer, D.: 'Reliable and reproducible competition results with benchexec and witnesses (report on SV-COMP 2016)', *LNCS*, 2016, **9636**, pp. 887–904

207 Ábrahám, E., Abbott, J., Becker, B., Bigatti, A.M., Brain, M., Buchberger, B., et al.: 'Satisfiability checking and symbolic computation', *ACM Communications in Computer Algebra*, 2016, **50**, (4), pp. 145–147

208 O'Hearn, P.W.: 'Separation logic', *Communications of the ACM*, 2019, **62**, (2), pp. 86–95

209 Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., et al. 'Moving fast with software verification'. In: 7th International Symposium on NASA Formal Methods. vol. 9058 of *LNCS*, 2015. pp. 3–11

210 Inverso, O., Tomasco, E., Fischer, B., La.Torre, S., Parlato, G. 'Bounded model checking of multi-threaded c programs via lazy sequentialization'. In: Computer Aided Verification, 2014. pp. 585–602

211 Kahlon, V., Wang, C., Gupta, A. 'Monotonic partial order reduction: An optimal symbolic partial order reduction technique'. In: Computer Aided Verification. vol. 5643 of *LNCS*, 2009. pp. 398–413

212 Morse, J., Cordeiro, L.C., Nicole, D.A., Fischer, B. 'Context-bounded model checking of LTL properties for ANSI-C software'. In: 9th International Conference on Software Engineering and Formal Methods. vol. 7041 of *LNCS*, 2011. pp. 302–317

213 Zheng, M., Rogers, M.S., Luo, Z., Dwyer, M.B., Siegel, S.F. 'CIVL: formal verification of parallel programs'. In: 30th International Conference on Automated Software Engineering, 2015. pp. 830–835

214 Kroening, D., Poetzl, D., Schrammel, P., Wachter, B. 'Sound static deadlock analysis for C/Pthreads'. In: 31st International Conference on Automated Software Engineering, 2016. pp. 379–390

215 La Torre, S., Madhusudan, P., Parlato, G. 'Reducing Context-Bounded Concurrent Reachability to Sequential Reachability'. In: Computer Aided Verification. vol. 5643 of *LNCS*, 2009. pp. 477–492

216 Grumberg, O., Lerda, F., Strichman, O., Theobald, M. 'Proof-guided underapproximation-widening for multi-process systems'. In: 32nd Symposium on Principles of Programming Languages, 2005. pp. 122–131

217 Jovanovic, D., Dutertre, B. 'Property-directed k-induction'. In: Formal Methods in Computer-Aided Design, 2016. pp. 85–92

218 Cox, A., Sankaranarayanan, S., Chang, B.E.: 'A bit too precise? verification of quantized digital filters', *International Journal on Software Tools for Technology Transfer*, 2014, **16**, (2), pp. 175–190

219 Abreu, R.B., Gadelha, M.Y.R., Cordeiro, L.C., de Lima.Filho, E.B., da Silva.Jr., W.S.: 'Bounded model checking for fixed-point digital filters', *Journal of the Brazilian Computer Society*, 2016, **22**, (1), pp. 1:1–1:20

220 Hamon, G. 'Simulink design verifier – applying automated formal methods to simulink and stateflow'. In: 3rd Workshop on Automated Formal Methods, 2008. pp. 1–2

221 Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F.: 'A survey on software fault localization', *IEEE Transactions on Software Engineering*, 2016, **42**, (8), pp. 707–740

222 Cordeiro, L.C., Kesseli, P., Kroening, D., Schrammel, P., Trtík, M. 'JBMC: A bounded model checking tool for verifying java bytecode'. In: Computer Aided Verification. vol. 10981 of *LNCS*, 2018. pp. 183–190

223 Lopes, B.C., Auler, R.: 'Getting Started with LLVM Core Libraries'. (Packt Publishing, 2014)

224 Metz, C.: 'Why Apple's swift language will instantly remake computer programming'. (http://www.wired.com/2014/07/apple-swift/, 2016). Accessed 14th of April 2018

225 Arroyo, M., Chiotta, F., Bavera, F. 'An User Configurable Clang Static Analyzer Taint Checker'. In: Conference Of The Chilean Computer Science Society, 2016. pp. 1–12

226 Roy, P., Tabuada, P., Majumdar, R. 'Pessoa 2.0: a controller synthesis tool for cyber-physical systems'. In: 14th International Conference on Hybrid Systems: Computation and Control, 2011. pp. 315–316

227 Jackson, D., Vaziri, M. 'Correct or usable? the limits of traditional verification (impact paper award)'. In: 24th International Symposium on Foundations of Software Engineering, 2016. pp. 11–11

228 Morse, J., Ramalho, M., Cordeiro, L.C., Nicole, D., Fischer, B. 'ESBMC 1.22 - (competition contribution)'. In: Tools and Algorithms for the Construction and Analysis of Systems. vol. 8413 of *LNCS*, 2014. pp. 405–407

229 Haran, A., Carter, M., Emmi, M., Lal, A., Qadeer, S., Rakamarić, Z. 'SMACK+Corral: A modular verifier'. In: Tools and Algorithms for the Construction and Analysis of Systems, 2015. pp. 451–454

230 Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A. 'Witness validation and stepwise testification across software verifiers'. In: 23th International Symposium on Foundations of Software Engineering, 2015. pp. 721–733

231 Jesus, A.S., Rodrigues, R.N., Ferreira, A.N.G., Melo, W.C., Lima.Filho, E.B., Silva.Júnior, W.S. 'Automatic antenna alignment system for satellite receivers operating in c and ku bands'. In: Brazilian Symposium on Telecommunications and Signal Processing (in Portuguese), 2017. pp. 1–5

232 Amoedo, D.A., da Silva.Júnior, W.S., de Lima.Filho, E.B. 'Parameter selection for SVM in automatic modulation classification of analog and digital signals'. In: International Telecommunications Symposium, 2014. pp. 1–5

233 Hamel, L. 'On the use of machine learning in formal software verification'. (Dept. of Computer Science and Statistics, University of Rhode Island, 2003. technical Report TR03-294

234 Phuc, N.V.: 'The Application of Machine Learning Methods in Software Verification and Validation'. (University of Texas at Austin, 2010)

235 Bridge, J.P., Holden, S.B., Paulson, L.C.: 'Machine learning for first-order theorem proving', *Journal of Automated Reasoning*, 2014, **53**, (2), pp. 141–172

236 Hutter, F., Babic, D., Hoos, H.H., Hu, A.J. 'Boosting verification by automatic tuning of decision procedures'. In: Formal Methods in Computer-Aided Design, 2007. pp. 27–34

237 Seshia, S.A., Hu, S., Li, W., Zhu, Q.: 'Design automation of cyber-physical systems: Challenges, advances, and opportunities', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017, **36**, (9), pp. 1421–1434

238 Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A. 'Synthesizing switching logic for safety and dwell-time requirements'. In: 1st International Conference on Cyber-Physical Systems, 2010. pp. 22–31

239 Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S., Seshia, S.A. 'A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications'. In: 53rd Conference on Decision and Control, 2014. pp. 1091–1096

240 Lucky, R.W.: 'Antifragile Systems'. (https://spectrum.ieee.org/telecom/wireless/antifragile-systems, 2013). Accessed 13rd of December 2017

241 Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M.: 'Lessons from applying the systematic literature review process within the software engineering domain', *Journal of Systems and Software*, 2007, **80**, (4), pp. 571–583