# Formal Methods in Industry

MAURICE H. TER BEEK, Formal Methods and Tools Lab, CNR-ISTI, Pisa, Italy
ROD CHAPMAN, Automated Reasoning Group, Amazon Web Services, Bath, United Kingdom of Great Britain and Northern Ireland
RANCE CLEAVELAND*, University of Maryland, College Park, United States
HUBERT GARAVEL, INRIA, Grenoble, France
RONG GU, Mälardalen University, Västerås, Sweden
IVO TER HORST, ASML, Veldhoven, Netherlands
JEROEN J. A. KEIREN, Eindhoven University of Technology, Eindhoven, Netherlands
THIERRY LECOMTE, CLEARSY Systems Engineering, Aix-en-Provence, France
MICHAEL LEUSCHEL, Heinrich Heine University Düsseldorf, Düsseldorf, Germany
KRISTIN YVONNE ROZIER, Iowa State University, Ames, United States
AUGUSTO SAMPAIO, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil
CRISTINA SECELEANU, Mälardalen University, Västerås, Sweden
MARTYN THOMAS, Gresham College, London, United Kingdom of Great Britain and Northern Ireland
TIM A. C. WILLEMSE, Eindhoven University of Technology, Eindhoven, Netherlands
LIJUN ZHANG, Institute of Software, Chinese Academy of Sciences, Beijing, China

Formal methods encompass a wide choice of techniques and tools for the specification, development, analysis, and verification of software and hardware systems. Formal methods are widely applied in industry, in activities ranging from the elicitation of requirements and the early design phases all the way to the deployment, configuration, and runtime monitoring of actual systems. Formal methods allow one to precisely specify the environment in which a system operates, the requirements and properties that the system should satisfy, the models of the system used during the various design steps, and the code embedded in the final implementation, as well as to express conformance relations between these specifications. We present a broad scope

*The author died during the second revision of this paper. We gratefully acknowledge his contributions in the appropriate section at the end of this paper.

Authors' Contact Information: Maurice H. ter Beek, Formal Methods and Tools Lab, CNR-ISTI, Pisa, Italy; e-mail: maurice.terbeek@isti.cnr.it; Rod Chapman, Automated Reasoning Group, Amazon Web Services, Bath, United Kingdom of Great Britain and Northern Ireland; e-mail: rodchap@amazon.co.uk; Rance Cleaveland, University of Maryland, College Park, Maryland, United States; e-mail: rance@cs.umd.edu; Hubert Garavel, INRIA, Grenoble, France; e-mail: hubert.garavel@inria.fr; Rong Gu, Mälardalen University, Västerås, Västmanland, Sweden; e-mail: rong.gu@mdu.se; Ivo ter Horst, ASML, Veldhoven, Netherlands; e-mail: ivo.ter.horst@asml.com; Jeroen J. A. Keiren, Eindhoven University of Technology, Eindhoven, Noord-Brabant, Netherlands; e-mail: j.j.a.keiren@tue.nl; Thierry Lecomte, CLEARSY Systems Engineering, Aix-en-Provence, France; e-mail: thierry.lecomte@clearsy.com; Michael Leuschel, Heinrich Heine University Düsseldorf, Düsseldorf, Nordrhein-Westfalen, Germany; e-mail: leuschel@hhu.de; Kristin Yvonne Rozier, Iowa State University, Ames, Iowa, United States; e-mail: kyrozier@iastate.edu; Augusto Sampaio, Centro de Informática, Universidade Federal de Pernambuco, Recife, Pernambuco, Brazil; e-mail: acas@cin.ufpe.br; Cristina Seceleanu, Mälardalen University, Västerås, Västmanland, Sweden; e-mail: cristina.seceleanu@mdu.se; Martyn Thomas, Gresham College, London, United Kingdom of Great Britain and Northern Ireland; e-mail: martyn@mctar.uk; Tim A. C. Willemse, Eindhoven University of Technology, Eindhoven, Noord-Brabant, Netherlands; e-mail: t.a.c.willemse@tue.nl; Lijun Zhang, Institute of Software, Chinese Academy of Sciences, Beijing, China; e-mail: zhanglj@ios.ac.cn.


ACM 1433-299X/2024/8-ART
https://doi.org/10.1145/3689374

of successful applications of formal methods in industry, not limited to the well-known success stories from the safety-critical domain, like railways and other transportation systems, but also covering other areas such as lithography manufacturing and cloud security in e-commerce, to name but a few. We also report testimonies from a number of representatives from industry who, either directly or indirectly, use or have used formal methods in their industrial project endeavours. These persons are spread geographically, including Europe, Asia, North and South America, and the involved projects witness the large coverage of applications of formal methods, not limited to the safety-critical domain. We thus make a case for the importance of formal methods, and in particular of the capacity to abstract and mathematical reasoning that are taught as part of any formal methods course. These are fundamental Computer Science skills that graduates should profit from when working as computer scientists in industry, as confirmed by industry representatives.

CCS Concepts: • **Software and its engineering → Formal methods**; • **Social and professional topics → Computer science education**.

Additional Key Words and Phrases: Formal Methods, Computer Science Education



## 1 Introduction

Formal methods collectively refer to an array of methods for mathematically specifying and verifying computer-system behaviour. In such approaches, systems are interpreted as mathematically precise structures, whether as state machines, as functions mapping initial to final states, or as logical formulas describing system behaviour. Specifications also are given in a mathematically well-founded manner, whether as logical properties or state machines, and the notion of a system satisfying a specification is also given a mathematical definition. Given these elements, formally verifying a system involves constructing a mathematical proof that the system satisfies the specification. The key motivation for these techniques is the strength of the correctness guarantees they provide: in contrast to testing-based and inspection-based techniques, a proof conclusively demonstrates that the system in question, *at the level of abstraction that it is presented*, is correct with respect to its specification. Formal methods complement other verification and validation techniques, such as testing or simulation [169].

Formal methods have been studied in the computing community since at least the 1960s, with seminal work by Floyd [133], Hoare [170], and Dijkstra [113] defining techniques for proving programs correct. Later work by Pnueli, Lamport, Clarke, Emerson, and others considered the automated verification of state machines *vis à vis* properties in temporal logics [87, 201, 256]. Still others, including Boyer, Moore, Gordon, and Coquand [66, 100, 152], developed automated theorem provers for checking the correctness of verification proofs. Later researchers have built on and improved these efforts, and formal methods remain a vital and fundamental area of basic computing research. Notably, the Cost of Poor Software Quality (CPSQ) is astonishing. In the 2022 Report[1] for the CPQS in the US, the amount is considered at least US$ 2,41 trillion. Considering the more specific cost of finding and fixing bugs, the estimation is US$ 607 billion. Therefore, there is scope for applications of formal methods and tools to improve this situation in several domains, much beyond the context of safety-critical systems. Historical references from long-time advocates of formal methods reflect on their industrial application through the metaphors of myths and commandments of formal methods and invite their uptake in industry in order to realize their benefits [29, 61–65, 84, 102, 161, 162, 195, 227, 249, 276, 287, 298]. Throughout the last three decades several major surveys of formal methods have appeared in the literature [90, 135, 138, 144, 299].

[1] https://www.it-cisq.org/

This paper demonstrates that formal methods have wide-ranging practical value by reporting on the increasing *use* of formal methods in industry, and it makes a case for the inclusion of formal methods as a separate topic in Computer Science education. This is well agreed upon by the formal methods community. To make this point better known to the Computer Science community at large and, in particular, to those involved in Computer Science education, the remainder of this article develops as follows. First, in Section 2, we give more detail about what formal methods are, precisely. Then, in Section 3, we survey the vast array of formal-methods applications, much beyond the context of safety-critical systems[2]. In our brief survey, we cite specific instances across a variety of different domains of successful applications of the techniques in delivering systems that, by their nature, must be reliable. These include very recent work, which was not available five years ago when the most recent above-mentioned surveys were conducted, as well as testimonies contributed by current industry leaders. Next, in Section 4, we argue that undergraduate curricula should include formal methods as a topic, not only because of their growing importance in industry, as witnessed by the evidence presented in this paper, but also because formal methods contribute to cultivate abstract thinking, enabling students to better understand and solve complex problems, and because of the discipline they instil in students as they learn to develop systems. In the words of Rance Cleaveland, "students who learn formal methods are better developers, because they learn to think about correctness while they are building systems." Further support for such statements can be found in [34] and in the accompanying papers [118], which underlines the importance of formal methods *thinking* in Computer Science education, and [70], which argues that every computer scientist needs to *know* formal methods: "software developers not being aware of the various benefits of formal methods cannot be called computer scientists or software engineers." Finally, in Section 5, we conclude the paper.



## 2 Formal Methods and Tools

Complex, industrial systems typically come into being according to some standardized software development methodology. Most methodologies distinguish different phases in the development of a system. While these phases may differ across development methodologies, and carry different names, in general, there is a requirements analysis phase, an architecture design phase, a system design phase, and an implementation phase. Each phase yields a set of artefacts. Apart from describing the type of artefacts delivered at each phase in the development of a system, a methodology also describes the validation and verification methods and activities needed to ensure the internal consistency of an artefact, and the consistency of artefacts across different phases. For instance, the artefacts produced during the requirements phase need to be non-contradictory, whereas the artefacts produced in the design phase need to be consistent with the artefacts delivered in the requirements phase. Formal methods, and their implementation in widely available tools, provide automated, repeatable, easily checkable evidence to support these needs. Triggered by their successful applications in industry, we mostly focus on *a posteriori* verification of (software) systems. We do not discuss in detail the verification of formal specifications of requirements nor the alternative of *a priori* verification techniques, such as developing correct-by-construction system implementations based on formal methods that provide correctness-preserving refinement transformations (like in the seminal work on the refinement calculus and related calculi by Back [10], Morris [231], and Morgan [230], based on Dijkstra's guarded command language [114]).

In the remainder of this section, we first define and introduce formal methods in Section 2.1, after which we provide an overview of selected formal methods and tools in Section 2.2.

### 2.1 What are Formal Methods?

[2]We survey formal-methods applications from well-known and representative domains, including papers and testimonies that have been selected based on the expertise and experience of the authors, spread geographically as well as across different application domains.

In line with [138], in this paper we define formal methods as mathematically rigorous techniques for the specification, design, validation, and (manual or automated) verification of software and (logical) hardware systems. In short, formal methods enable checking that behaviours (statements in a formal semantics such as mathematical logic) always hold in a system (specified in a language with a formal semantics). They examine the entire behaviour space of the system, covering all possible inputs, to provide assurance derived from a mathematical proof that the specified system's behaviour is correct. In other words, formal methods address the verification problem: given formal definitions of what a system does ($M$) and what it should do ($\varphi$), formal methods show that $M$ *satisfies* $\varphi$. Intuitively, formal methods show that the system does what you think it should do *and nothing else*.

That last part, "and nothing else," distinguishes exhaustive formal methods from non-exhaustive methods for verification, such as testing [14, 50, 58, 226] and simulation [181, 291]. These popular methods reason over the *input space* of a system by producing a set of *individual system executions*. By aggregating a large set of executions, they can provide probabilistic answers to questions such as: how often does some behaviour occur, how many inputs produce a certain type of output, or how likely is something to occur. They can provide a proof of the existence of a run of a system, such as a run that exhibits a fault, but since neither testing nor simulation explores all possible system behaviour (state space), they can never show the absence of such a run. For that, we need (exhaustive) formal methods.

Formal methods reason over the *behaviour space* of a system. They yield Boolean (typically true or false, not probabilistic) answers to questions like: is it possible for something to occur, does this property hold for all system executions, or even how many system execution paths lead to a certain output. While testing and simulation involve executing the system many times to gather examples for verification, formal methods tools generally execute once but reason exhaustively over the complete set of system behaviours, covering all possible executions. This is also why there is such a variety of formal methods (as opposed to a singular formal method) — different methods scale differently and different systems require different proof strategies to enable such exhaustive reasoning. Successful application of formal methods requires some knowledge of the underlying system, and therefore which formal method(s) to apply.

In practice, (exhaustive) formal methods provide quite a different understanding of a system than non-exhaustive or informal methods. Informal or semi-formal methods refer to techniques and tools that are not fully formal, i.e., lacking a precise and unambiguously defined syntax and semantics. Formal methods can prove both the presence and the absence of given behaviour. Because mathematical proofs reason equally well over partially-defined systems as fully-defined ones, formal methods can check systems starting from the earliest stages of the system design. While simulation and testing require some form of executable to run, formal methods do not. Nor are formal methods sensitive to systems being used in ways they were not intended, such as receiving unanticipated inputs. These are precisely the properties that make formal methods essential tools for industry, e.g., for certifying software and hardware systems, also for non-safety-critical systems. Anticipating the testimonies from Rod Chapman (Amazon Web Services) in Section 3.6 and Ivo ter Horst (ASML) in Section 3.8, respectively:

> **Chapman:** "A key point is that AR [automated reasoning] builds trust with customers by allowing universal and sound verification of properties of our infrastructure and customers' applications. By "universal," AWS means properties that hold for all users, all storage buckets, all networks, all compute instances, all configurations, and so on — freeing the user from having to "test" a nearly infinite state space."

> **ter Horst:** "To make ASML's lithography systems run reliably and consistently ASML needs software that sends unambiguous instructions in every situation to the carefully engineered hardware. One way that ASML ensures this is by formally verifying (model checking) the specified machine behaviour and automatically generating correct and semantically equivalent code from those models."

For this reason, experts continue to argue for the continuous integration of formal methods throughout the complex industrial system development lifecycle, and for the formal methods education of system developers [138] (cf. Section 3.1). In our experience, reported in this paper, which covers 50 years of work in industry as well as in academia, most software projects lack the discipline seen in other branches of engineering. Few developers approach their work with the understanding that mistakes will be made and that they should select and use the methods and tools that were designed to prevent errors and to detect errors as early as possible after they are made. Too few Computer Science graduates work in a way that shows that they understand their responsibility to be able to provide evidence that their software meets their customer's or end user's requirements — and that evidence from testing can never be strong enough on its own.

This does not mean that we are suggesting that formal methods exclude or are a replacement for testing or simulation. As mentioned in the Introduction, these are complementary techniques for the central task of verification and validation (cf. [169] and, for a concrete example, Section 3.9: Formal Testing of Mobile Devices from Natural Language Requirements). Our purpose is to emphasize the distinguishing feature of formal methods in being capable of mathematically ensuring the absence of errors with respect to a given specification.

## 2.2 Overview of Formal Methods and Tools

Formal methods offer the means to both formally describe the artefacts delivered at each phase in the system development lifecycle, and aid in their automated validation and verification [7, 135]. Some tools prove useful throughout the development methodology; others may specialize to address specific phases and activities. As mentioned in the beginning of this section, we focus on formal methods for the verification phase. We refer to [70] for a discussion on formal methods for all phases: identifying and formalizing requirements, modelling and formal specification, design and implementation, verification and validation, and maintenance and evolution. There are two core formal (verification) methods from which all other formal (verification) methods derive: *model checking* [17, 88] and *theorem proving* [238, 266].

Theorem provers, such as Coq[3] [51], Isabelle [241], Lean [233], Vampire [264], KeYmaera X [254, 255], LP [140], PVS [248], Z3 [232], and others allow users to mechanically, and sometimes even automatically, prove generic statements and theories about system artefacts formalized as mathematical theories. Theorem proving has found its way in tool sets for state-based refinement approaches (as originally advocated by VDM [188], Z [282], and subsequently B [2, 3]), such as Atelier B[4] and Event-B[5]. A theorem prover's independence of specific problem domains is one of its major strong points. Each theorem prover brings its own library of previous proofs upon which it can draw to efficiently prove new theorems. Theorem provers differ in the contents of these libraries and in their input logics. Typically users choose which theorem prover to use for a particular job by choosing the input language that most intuitively describes the verification question at hand and checking the proof library for previous proofs useful in constructing the needed proof. While a theorem prover can complete some proofs automatically after the user sets up the theorem correctly, others require substantial interaction from the user to complete the proof, so utilizing previous results from the proof library and having a clearly-defined proof strategy are essential. Theorem provers are very powerful tools that can reason about very large, or even infinite-state, systems and complex mathematical algorithms.

Process-algebraic approaches such as ACP [15], CCS [228], CSP [171, 269], LOTOS [57], or LNT [137] have inspired the development of model-checking toolsets such as CADP[6] [136], CWB [91] / CWB-NC [93], FDR [143], and mCRL2 [9, 75, 157] (cf. [92]). These and other model checkers, such as SPIN[7] [172], NuSMV [86] / nuXmv [80],

[3]Coq received the 2013 ACM Software System Award.
[4]https://www.atelierb.eu/
[5]http://www.event-b.org/
[6]CADP received the ETAPS Test-of-Time Tool Award 2023.
[7]SPIN received the 2001 ACM Software System Award.

UPPAAL[8] [38, 39, 109, 110], ProB [211], and ABC [69], provide a convenient "push-button" technique for automatically assessing the consistency of an artefact (e.g., an algorithm or design described in a formal semantics) by automatically verifying whether it satisfies behavioural specifications typically expressed as assertions or using some form of temporal logic. The industrial appeal of model checking includes the limited user interaction required to achieve a complete, exhaustive verification result. To effectively use a model checker, the user needs only two inputs: the formal artefact or system description, and the logical specification to check it against. The user then pushes a button and receives either confirmation, e.g., in the form of a certificate, that the system artefact always upholds the specification, or a counterexample proving that it does not. A counterexample is a system trace stepping state-by-state through a valid execution of the system until a clear violation of the specification occurs. Counterexamples are therefore incredibly useful for debugging. Model checkers produce results that are *exhaustive*: if there exists any system execution that violates the specification, they will produce a counterexample.

In this way, both theorem provers and model checkers effectively prove both the presence and the absence of bugs. Moreover, both theorem provers and model checkers require guidance from a knowledgeable user to structure and organize their specifications, and in the case of theorem provers, also their proofs. Model checkers, on the other hand, have fully-automatic proofs but come with some limitations when compared to theorem provers. Due to their exhaustive exploration of the state space, model checkers are sensitive to the shape and size of the state space of the input system description and often do not allow elements like floating-point variables or unbounded integers. Also, the presence of superfluous information not relevant to the core algorithm under verification can dramatically slow down the model checker or cause it to time-out. Therefore the user must be careful to describe only the relevant system logic in the input description to mitigate the state-space explosion problem, where the number of states needed to model the system accurately may exceed the amount of available computer memory. "Despite the development of several very effective methods to combat this problem [...], models of realistic systems may still be too large to fit in memory" [17, Section 1.2.2: Strengths and Weaknesses].

Similarly to choosing a theorem prover, users choose model checkers based on the efficiency of the input modelling language at describing the verification problem at hand. There are two types of model checkers, explicit and symbolic, and some knowledge of which of these two types is best-suited to the problem is also helpful. Explicit model checkers explicitly represent the systems's behaviour space as a type of graph in memory and the model checker systematically explores each state and verifies whether it satisfies the given specification. Such an enumerative representation is suitable for systems with smaller state spaces, or state spaces with certain types of repeated patterns. Instead of representing individual states or transitions, symbolic model checkers represent sets of states and sets of transitions symbolically using data structures such as Boolean formulas or Binary Decision Diagrams (BDDs). Symbolic model checking is particularly suitable for systems with a large, or even infinite state space. Finally, for both theorem proving and model checking, the user must be careful to specify the behaviour property correctly. Since many system requirements in natural language are vague, incomplete, or confusing, this can be a very challenging task [271]. NASA's Formal Requirements Elicitation Tool (FRET)[9] [142] for the elicitation, formalization and understanding of requirements may be of help. FRET assigns unambiguous semantics to requirements written in a structured natural language and allows to export the requirements into forms that can be used by a variety of analysis tools, among which Simulink Design Verifier and SMV (cf., e.g., [123], where it is reported that the industrial partner found the FRET tool "very easy to use").

Next to these exhaustive qualitative verification techniques, it is worth mentioning exhaustive quantitative methods such as probabilistic (a.k.a. stochastic) model checking, and non-exhaustive methods such as runtime

[8] https://uppaal.org/
[9] https://software.nasa.gov/software/ARC-18066-1

verification, model-based testing and statistical model checking, and light-weight formal methods such as static analysis.

Static analysis concerns the derivation of properties of interest from source code (or an intermediate representation) without executing the code [265], meaning precision is the price to pay. Well-known static analysis tools include Astrée[10] [112], Coccinelle[11] [204], Frama-C[12] [194], and Lint [187] (cf. [116] for key lessons from designing the static analyses tools Infer and Zoncolan and [210] for a comparison of the static analysis tools Better Code Hub, CheckStyle, Coverity Scan, FindBugs, PMD, and SonarQube). Typically, one has to decide between under- and overapproximations (e.g., abstract interpretation [103]), with the possibility of both false positives and false negatives. Underapproximation[13] is a consequence of an approach representing all possible program behaviours in a way that includes some, but not necessarily all actual behaviour (thus giving rise to false negatives, i.e., the analysis may fail to detect certain properties of the code), whereas an overapproximation represents the set of all possible program behaviour in a way that includes all actual behaviour as well as possibly some that are not possible (thus giving rise to false positives, i.e., the analysis may detect certain problems that do not actually exist in the code). The choice between under- and overapproximation depends on the specific goals of the analysis and the trade-off between precision and completeness, where the challenge is to find a suitable abstraction that is both computationally feasible and provides meaningful insights into the program's behaviour while dealing with false positives and false negatives.

Model-based testing is a formal-methods approach to testing that complements formal verification and model checking and increases the efficiency and effectiveness of software testing [290]. It uses a formal or semi-formal model to represent the desired behaviour of a system under test, which serves as the basis for generating test cases and executing tests. It is typically more efficient than traditional testing approaches, since it automates the test case generation process. Moreover, by systematically deriving test cases from the model, often a better coverage of the system's behaviour is achieved. Model-based testing complements other testing methodologies and is part of the broader landscape of model-driven engineering [73].

Runtime verification monitors and analyzes actual software (and hardware) system behaviour while the system is running [27, 94]. It offers improved practical applicability and scalability compared to exhaustive formal verification, such as model checking and theorem proving, by analyzing only one—or a few—execution traces of the actual system. Runtime monitoring derives from model checking, except that in model checking the running system is the input system description, so instead of an exhaustive analysis of all possible system runs (like model checking), runtime monitoring analyses only the "current" system run against the input logical specification (cf. [272] for a disambiguation from simulation).

Compared to model checking, which focuses on absolute guarantees of correctness, probabilistic or stochastic model checking focus on modelling and analyzing systems that exhibit probabilistic or stochastic behaviour [16, 17, 198, 200, 263]. Such aspects are essential in cases of unreliable or unpredictable system behaviour and performance evaluation. Instead of providing a yes/no answer to the question as to whether a system model ($M$) satisfies a temporal logic property ($\varphi$), the answers are of the form "with a likelihood of 99%, $M$ will satisfy $\varphi$," where $\varphi$ is expressed in a stochastic or probabilistic temporal logic. Statistical model checking [5, 208] uses a simulation- and sample-based approach to reason about precise properties specified in a stochastic temporal logic, offering a scalability advantage over exhaustive (or probabilistic) model checking due to the fact that there is no need to analyze entire state spaces. Moreover, even though the outputs of sample-based methods

[10] https://www.astree.ens.fr/

[11] https://coccinelle.gitlabpages.inria.fr/website/

[12] https://frama-c.com/

[13] Not to be confused with underapproximate triples in incorrectness logic [245], a logical underapproximate theory for proving the presence of problems.

are not always correct, statistical inference enables quantifying the confidence in the obtained result[14], thus compensating for the lack of exact results (100% confidence).

## 3 Formal Methods in Industry

In this section, we present a broad scope of applications of formal methods in industry, not limited to the safety-critical domain, including testimonies contributed by Rod Chapman from Amazon Web Services, leader in cloud computing (cf. Section 3.6), and Ivo ter Horst from ASML, leader in the semiconductor industry (cf. Section 3.8). The reported applications of formal methods in industry range from experiments with formal methods in industry to routine applications of formal methods in industry as part of the development process. After a summary of recent literature on successful applications of formal methods in Section 3.1, we describe a selection of success stories for applying formal methods in the safety-critical domains of railways, automotive, and aerospace in Sections 3.2–3.4. Subsequently, we report testimonies of success stories in the non-safety-critical domains of operating systems in Section 3.5, e-commerce in Section 3.6, hardware design in Section 3.7, lithography manufacturing in Section 3.8, and mobile devices in Section 3.9.

### 3.1 Summary of Recent Literature

Success stories of the application of formal methods in industry traditionally focus on their application to safety-critical systems, such as transport, nuclear power plants, and medical devices. Arguably one of the most cited ones concerns the fatal accidents with the infamous Therac-25 software-controlled radiation therapy machine that were, among others, due to software coding errors. As demonstrated in [213, 214, 288, 289] (using, among others, the process algebra LOTOS [57] and the theorem prover LP [140]) these software errors could have been avoided if "basic software engineering principles" and "sophisticated modeling and analysis tools" had been applied. Alas, what Nancy Leveson wrote in 1993, "software should be subjected to extensive testing and formal analysis" [214], and in 2017, "it's time for computer science practitioners to be better educated about engineering for safety." [213], is still true. Transport applications include, but are not limited to, the railway [33, 77, 207] and aerospace [234, 281] domains. We refer to, e.g., [32, 36, 128, 145, 174, 299] for more complete overviews of such applications. A recent survey among 216 participants studying the use of formal methods for mission-critical software indicates "an increased intent to apply FMs in industry, suggesting a positively perceived usefulness" [144].

Outside safety-critical applications, the literature also reports a recent uptake in the application of formal methods [82]. Formal methods have, for example, been applied to ensure the quality of cloud services at Amazon [12, 239], cloud databases at Huawei [134], and mobile apps at Facebook [116]. Sadowski et al. [277] describe how formal methods are integrated in the software development workflow at Google. Godefroid reviews concolic testing and various forms of fuzzing, which are capable of scaling to Microsoft applications (e.g., Excel or PowerPoint) with millions of lines of code [146]. Concrete symbolic (concolic) testing is a hybrid software verification technique that performs symbolic execution along concrete execution paths in an attempt to systematically explore the execution of a program, focusing on both specific input values (as for traditional testing) and symbolic representations of various alternative program paths, to achieve improved path coverage compared to traditional testing. Scalability is a challenge due to the path explosion problem, i.e., the number of possible paths grows significantly as the program's complexity increases [280]. Fuzzing is a more light-weight testing technique focused on quickly exploring a large input space by providing random or semi-random inputs, typically generated by mutation-based or generation-based fuzzing, to a program to discover vulnerabilities or unexpected

[14]The level of confidence is usually stated as a percentage $100 \times (1 - \alpha)\%$, meaning that $100 \times (1 - \alpha)\%$ of the time the actual expected value belongs to the confidence interval $[X - \delta/2, X + \delta/2]$, where $X$ is the estimated value, $\alpha$ is the confidence, and $\delta$ is the width of the confidence interval, which is typically determined based on $\alpha$ and a large enough $n$, the number of samples obtained from $n$ independent simulations.

behaviour [286]. Formal methods have also been used successfully to show the incorrectness of widely used software such as Timsort [154], the Java LinkedList implementation [168], and implementations of the MCS mutual exclusion locks [225] in open-source weak memory models and at Huawei [244], as well as the correctness of the seL4 operating-system kernel [192, 193] and the CompCert C compiler [26, 55, 183][15]. In the same realm, modern programming language features such as Rust's memory safety and Go's concurrency have a solid foundation in formal methods (cf., e.g., [104, 189]). Finally, formal methods have also been applied in other domains, like in medical imaging for model checking the segmentation of glioblastoma and nevi [22, 41, 42].

A recent survey among 130 experts in formal methods (including 3 Turing Award winners[16], all 4 FME Fellowship Award winners[17] and 16 CAV Award winners[18]) investigated the factors that limit the uptake of formal methods in industry practice. In this survey, 71.5% of the respondents identifies that "engineers lack proper training in formal methods" as a *limiting factor for a wider adoption of formal methods by industry* [138, Section 5: Formal Methods in Industry]. Other key limiting factors are that "academic tools have limitations and are not professionally maintained" (66.9%), formal methods "are not properly integrated in the industrial design life cycle" (66.9%) and "have a steep learning curve" (63.8%). Related to this, 62.3% indicates that "developers are reluctant to change their way of working." Another survey [138] concludes that "the current situation [of formal methods education] is very heterogeneous across universities, and many experts call for a standardisation of university curricula with respect to formal methods."

In the following sections, we describe a selection of success stories of applying formal methods in both safety-critical (cf. Sections 3.2–3.4) and non-safety-critical domains (cf. Sections 3.5–3.9). Moreover, we relate the formal methods and tools mentioned in Sections 3.2–3.9 to the comprehensive classifications and explanations in Section 2. We acknowledge the need for performing more empirical studies on formal methods according to well-established guidelines [35] to establish at what point formal methods are being applied, which are the most frequently applied techniques and tools, and related questions. From two recent surveys from the literature involving, respectively, 216 professionals from Europe and North America using formal methods in dependable systems engineering [144] and 328 papers on formal methods in railways [128], we know that (i) the professionals employ formal methods mainly for assurance (e.g., proof, error removal), specification (i.e., formal description techniques), and inspection (e.g., error detection, bug finding), while in most of the papers formal methods are applied in the Architecture (66%) and Detailed Design (45%) development phases; (ii) the professionals mainly use formal methods analysis techniques for assertion checking, followed by consistency checking and model checking, while in the papers formal verification is the dominant analysis technique (67%), in particular model checking (47%) and theorem proving (19.5%), whereas static analysis is hardly used (1%); (iii) the professionals were not asked for their experiences with formal methods tools as it was left for future work "to find out which particular FM (and tool) is used in which domain for which particular purpose and role," while in the papers the tool landscape is rather scattered with ProB (9%), NuSMV (8%), and UPPAAL (7%) among the most frequently used ones, but not much more than Atelier B (5%), Event-B/Rodin (4%), SPIN (4%), and Simulink (4%).

## 3.2 Formal Methods for Railways

Railway signalling used to be done manually by observing trains and operating signals, which is error-prone and restricts the capacity of railway transportation. Automatic signalling is obviously needed for modern railway

[15]The seL4 developers received the 2022 ACM Software System Award for "the first industrial-strength, high-performance operating system to have been the subject of a complete, mechanically-checked proof of full functional correctness;" the 2021 ACM Software System Award went to "CompCert, the first practically useful optimizing compiler targeting multiple commercial architectures that has a complete, mechanically checked proof of its correctness."

[16]https://amturing.acm.org/byyear.cfm

[17]https://www.fmeurope.org/awards/

[18]http://i-cav.org/cav-award/

control systems. However, the safety of such automatic systems is crucial as a small error in signalling may have catastrophic consequences, such as train collisions. Moreover, replacing manual railway signalling with an automatic solution means huge investments and the extremely high standards of safety make it even more expensive. Formal methods can be a solution both for ensuring the safety of such systems and saving costs [40, Section 3: Cost-Benefit Analysis]. CENELEC EN 50128 is a European standard for the development of software for use in the railway industry [122]. It highly recommends formal methods for the design and verification of products that need to meet the highest safety integrity levels (SIL 3 or SIL 4, i.e., with a maximum accepted probability of dangerous failure between $10^{-7}$ and $10^{-9}$ per hour). The above mentioned cost-benefit analysis reported, which follows EU guidelines and covers both financial and economic analysis, is the only such analysis applied to formal methods that we are aware of. We agree with the authors of [40] who call for "greater attention of the formal methods community to the quantification of costs and benefits parameters [...] since the evidence of the beneficial effects of formal methods is mostly given instead in the literature in a qualitative way." In [247], the authors evaluated (without any monetary measurements) the effect of applying the commercial formal technique Analytical Software Design (ASD) to an industrial project, and they compared the positive results concerning code quality (good) and productivity (high) with those of 13 similar projects that used other formal methods (e.g., B and VDM). The above mentioned recent survey among 130 experts in formal methods also contained a question that asked the experts to make an informal cost-benefit analysis over time [138, Section 5.3: Return on Investment]. A small majority of 58.5% of the respondents answered that the application of formal methods is *profitable in medium and long terms*; 15% answered that they are *immediately profitable* and 12.3% that they are *profitable in the long term only*, while 2.3% answered that there is *no return on investment* and 11.5% had *no opinion*.

From the above mentioned recent survey of the landscape of research on applications of formal methods to the development of railway systems [128], involving 328 high-quality papers from 1989–2020[19], we know that formal methods in railways is a thriving research field with strong industrial ties, since 143 papers were published solely in the last five years (44% of the total of 328 papers) and 79 papers (24%) involved industry. Well-known success stories throughout the years concern the development and verification of the Automatic Train Protection (ATP) system for the RER Line A of Paris [159], the Subway Speed Control System (SSCS) of the subway of Calcutta [108], Line 14 of the Paris Metro [117], and derivatives thereof, like line 1 or the NY Canarsie line [121], and the driverless Paris–Roissy Airport shuttle [37], all developed with the B method. B was also used for an industrial scale system-level analysis of Alstom's U400 system [95], which is in operation in about 100 metro lines worldwide. Another success story concerns the metro control system of Rio de Janeiro, developed with the support of Simulink/Stateflow [125]. Simulink[20] is a model-based development tool for graphical system design, supporting simulation, test generation and code generation. A Simulink model's basic unit is a block diagram such that each block represents a different system component and their connections represent interactions between these components. Simulink comes with Stateflow, a graphical language inspired by Harel's hierarchical statecharts [166], for modelling and simulating the behaviour of complex systems in the form of state machines and flow charts, and it supports model checking through Simulink Design Verifier, which is part of the Simulink Verification and Validation tools. Further success stories concern the verification of the ERTMS/ETCS European standard for railway control and management with NuSMV [85] and of Hybrid ERTMS/ETCS Level 3 with a variety of formal methods and tools [30, 76]. In particular, in [164], the new system was modelled in B, identifying

[19]The survey was conducted following the guidelines for systematic mapping studies [252]. In particular, the 328 high-quality papers were selected from an initial set of 4346 papers retrieved from scientific databases upon the application of predefined criteria for inclusion (e.g., the study is written in English language) and exclusion (e.g., the study does not use a formal or semi-formal method) plus a quality checklist (e.g., is there a clear description of the task addressed with formal methods?) used for grading the papers. All papers with an insufficient overall quality score were excluded from the selection.

[20]https://nl.mathworks.com/products/simulink.html

over 30 issues and using the formal model as a runtime artefact for a real-life demonstration. Moreover, in [6], the system structure of the movement authority scenario of the Chinese Train Control System Level 3 (CTCS-3) was modelled by core constructs of the Architectural Analysis and Design Language (AADL) [111, 278], with its extensions Behavior Language for Embedded Systems with Software (BLESS) [203] for the discrete behaviour and Hybrid CSP [184] for the continuous behaviour, and verified with the Hybrid Hoare Logic (HHL) Prover [295], an interactive theorem prover based on Isabelle/HOL. Recently, the Autonomous Positioning System (APS) of the Florence tramways was verified with the support of the model checker UPPAAL [28]. See [126, 127, 223] for comparisons of different formal methods and tools for railway system design.

Railway transportation, such as trains, metros, and trams, is one of the most environmentally friendly and energy-efficient means of transportation. In the domain of railway control systems, a large number of research projects that involve formal methods have been carried out during the past decades, such as RobustRail[21] and more than one hundred projects funded under the Shift2Rail initiative[22], including the X2Rail series. Shift2Rail and its successor Europe's Rail are joint efforts of railway stakeholders and the EU to advance the railway domain through innovative research projects involving both academia and industry, in which formal methods are considered to be fundamental to the provision of safe and reliable technological advances in railways [33]. Notable initiatives outside the EU are the UK Rail Research and Innovation Network (UKRRIN)[23] and the Chinese State Key Laboratory of Rail Traffic Control and Safety[24].

Companies such as Alstom and Siemens are using formal methods, such as the B language, in the development of their train control systems as well as for data validation [206, 212], notably using the model checker ProB within tools like Systerel's OVADO[25] [1, 13] and the ClearSy Data Solver[26], both of which are certified T2 (i.e., tools where a fault could lead to an error in the results of verification or validation) for SIL 4 in accordance to the CENELEC EN 50128 standard. Prover is another industrial leader in formal methods for railway signalling automation. They develop software tools and services to support railway signalling design automation. Their solution covers a formal high-level language for formal verification and tools for developing, testing, and verifying railway control systems, like railway interlocking systems. They apply formal verification techniques, like theorem proving, in their interlocking software, digital twins, and development tools for railway signalling, which have been used in projects worldwide, like Sweden, Norway, China, France, and Canada [27]. The Prover Certifier formal verification tool, which includes the Prover PSL model checker, is also certified T2 for SIL 4 in accordance to the CENELEC EN 50128 standard. Moreover, the successful application of formal methods in Prover shows that formal verification can cut on-site testing time by up-to 50% and detect bugs that are overlooked by traditional testing[28]: "Formal Verification provides **much** higher coverage than testing. At Prover, we always find errors when doing formal verification, even on systems that have gone through regular verification"[29]

In the near future, the Railway domain is expected to contribute significantly to the European Green Deal by improved digitalization and data analytics[30]. Challenges include the extension of formal methods and tools to cope with AI-based systems, such as equipping verification tools with certificate generation, and their integration in the CENELEC standards [279].

[21] http://www.robustrails.man.dtu.dk
[22] https://projects.shift2rail.org
[23] https://www.ukrrin.org.uk/
[24] http://en.bjtu.edu.cn/research/institute/laboratory/16583.htm
[25] https://www.ovado.net/
[26] https://www.clearsy.com/en/tools/data-solver/
[27] https://www.prover.com
[28] https://www.prover.com/categories/verification-validation
[29] Daniel Fredholm from Prover Technology in his presentation *Formal Verification in the Railway Domain* during FME's "InFM: Industry talks on Formal Methods" series on May 16, 2024.
[30] https://transport.ec.europa.eu/system/files/2021-04/2021-mobility-strategy-and-action-plan.pdf

### 3.3 Formal Methods for Automotive

ISO 26262 is an international standard for functional safety in the automotive industry [179]. It provides guidelines and requirements for the development of safety-critical electrical and electronic systems (E/E systems) in vehicles. The standard is focused on ensuring the safety of E/E systems that are involved in the operation of passenger cars, motorcycles, trucks, and buses. The standard defines a safety life-cycle that encompasses various phases, including requirements, system, hardware, and software development. It emphasizes the identification and assessment of potential hazards, as well as the implementation of safety measures to mitigate risks. ISO 26262 also outlines processes for safety management, hazard analysis, risk assessment, and *verification and validation* of functional safety, defined as the absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems[31].

Compliance with ISO 26262 is typically required by regulatory bodies and is expected by customers in the automotive industry. Adhering to the standard helps ensure that vehicles and their associated systems are designed, developed, and produced with a focus on safety, reducing the likelihood of failures or malfunctions that could lead to accidents or injuries. Formal methods are considered the best choice in order to handle complexity and improve confidence in the automotive system's correctness, and although not mandated by ISO 26262, they are clearly encouraged and actually used, as demonstrated in the next paragraphs.

The need for advanced formal methodologies for design, development, and verification of automotive systems was identified by both industry and academia. Several projects were launched, and many problems were addressed. The systems modelling language (SysML) [180] is a general-purpose modelling language for systems engineering applications defined as an extension to UML addressing the structuring of requirements and their verification. Within several EU research projects, the Architecture Description Language, a metamodel and an ontology for representing engineering information for automotive embedded systems, called EAST-ADL[32] [56], was developed. EAST-ADL went further and applied an automotive ontology and representation aligned with AUTOSAR for the structuring of engineering information. The EAST-ADL model is structured in abstraction levels, where each sub-model represents the complete embedded system, at the relevant level of detail. The EAST-ADL abstraction levels map to the abstraction levels given in ISO 26262. EU projects like ATESST, CESAR, SafeCer, MAENAD, and MBAT all addressed the use of models and tools to automate the representation and formal verification of automotive systems' requirements.

Volvo Group Trucks Technology (VGTT) in Sweden is a division of Volvo Trucks that is a world-leading truck manufacturer, providing total transport solutions. In its *Model-based Continuous Integration of Automotive Embedded Systems* [215], VGTT applies the following engineering principles in order to address their product development and process challenges: (i) *Go Virtual*, to allow daily deliveries and maximize verification confidence while exercising dangerous and rare events, (ii) *Go Rigorous*, which requires the use of models, data, and formal verification to provide means to secure products versus needs and requirements, and allow engineering rigour and automation, (iii) *Go Multi-Method*, which incorporates in the engineering workflow a multitude of tools for the model representation (EAST-ADL, Simulink, the object-oriented, declarative modelling language Modelica[33], its Association Project Function Mockup Interface[34], and the model checker UPPAAL), as well as software-centric and physics-centric simulation (e.g., EAST-ADL/Simulink and Function Mockup Units) and formal verification of components' behaviour and timing (e.g., by employing the model-checking toolset UPPAAL), (iv) *Go Consistent*, to ensure that binaries and components are faithful realizations of models and code, and (v) *Go Continuous* to deliver daily/weekly/monthly component versions and tests. All these methods and tools for modelling and

[31] https://www.iso.org/obp/ui/en/#iso:std:iso:26262:-1:ed-2:v1:en
[32] https://www.east-adl.info/
[33] https://modelica.org/
[34] https://fmi-standard.org/

verification have been integrated or are under integration in VGTT's *ADAPT Integration Environment*, leading to *development efficiency* via iterative and incremental development, and *assured product quality* by incorporating *formal techniques*, such as SAT-based requirements consistency-checking with the Z3 theorem prover [216, 217] and model checking with the UPPAAL model checker [131, 158, 218], in the engineering workflow.

The recent rise of autonomous vehicles has brought exciting new challenges to the automotive industry to make such vehicles safe, reliable, and trustworthy, respect legal regulations, and ready for societal acceptance. These include dealing with uncertainties and incomplete or inaccurate information, as well as the development of effective formal methods and tools for the verification of AI-based systems based on transparent and explainable components that can be certified [219].

### 3.4 Formal Methods for Aerospace

Formal methods are now an expected and required part of the development processes of intelligent, autonomous, and safety-critical air and space systems. Their use is codified into flight certification, e.g., by the US Federal Aviation Administration (FAA) via DO-178B [257], DO-178C [260], DO-333 [259], and DO-254 [258], and in the EU by Regulation (EU) 2018/1139.[35] International standards agencies IEEE and IEC (International Electrotechnical Commission) maintain tens of standards for avionics involving formal methods [177, 178]. See [132] for a more detailed discussion of regulations for formal methods in certification of reliable autonomous systems, which include Unmanned Aerial Systems (UAS), covering airborne vehicles ranging from toy quadcopters to military UAS and autonomous missiles, but also driverless trains and self-driving cars.

Systems whose requirements, design, verification, and maintenance were shaped by formal methods continue to further the frontiers of modern aerospace engineering. For example, explicit-state model checking with model checkers like SPIN and the software model checker Java PathFinder [292] increased the robustness of the Small Aircraft Transportation System (SATS) [236]; proved the absence of synchronization faults in the Tactical Separation Assurance Flight Environment (TSAFE) [52]; verified a design-time hierarchical, concurrent spacecraft model [224]; and analyzed the Mars Science Laboratory's flight software [156, 173]. Symbolic model checking of temporal logic formulas [270] with model checkers like (Nu)SMV / nuXmv verified the Traffic Alert and Collision Avoidance System (TCAS) flying on-board commercial aircraft [83]; ensured internal aircraft modes followed the A-7E aircraft software requirements [283]; provided the basis for the Correctness, Modelling and Performability of Aerospace Systems (COMPASS) [67]; robustified Boeing's AIR6110 wheel braking system [68]; and changed NASA's design for the NextGen automated air traffic control system [139, 222, 302]. Such successes convinced the engineers at Dassault Aviation of the feasibility of verifying Esterel programs [49], which they use for parts of the safety-critical software of flight control systems but also for mission management systems. Theorem proving with theorem provers like KeYmaera and PVS provided many core verification results, e.g., for full-scale, real-life air traffic control systems including KB3D pair-wise conflict detection and resolution algorithms [235], Stratway (a modular approach to strategic conflict resolution) [160], ACCoRD (state-based conflict detection and resolution algorithms) [237], Chorus (tactical conflict and loss of separation detection and resolution) [78], and ACAS-X (Airborne Collision Avoidance System X) [182]. The theorem prover Isabelle/HOL [300] provided proofs for partition scheduling of a commercial real-time operating system implemented following the ARINC 653 international aerospace industry standard [4].



[35] Regulation (EU) 2018/1139 of the European Parliament and of the Council of 4 July 2018 on common rules in the field of civil aviation and establishing a European Union Aviation Safety Agency, and amending Regulations (EC) No 2111/2005, (EC) No 1008/2008, (EU) No 996/2010, (EU) No 376/2014 and Directives 2014/30/EU and 2014/53/EU of the European Parliament and of the Council, and repealing Regulations (EC) No 552/2004 and (EC) No 216/2008 of the European Parliament and of the Council and Council Regulation (EEC) No 3922/91: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32018R1139

Next to these exhaustive qualitative verification techniques, the exhaustive quantitative method of probabilistic model checking, in particular the probabilistic model checker PRISM[36] [199], proved instrumental in increasing the robustness of ACAS-X [294]; carrying out comparative analysis of automated air traffic control systems [303]; assessing the reliability, availability, and maintainability of a satellite [250]; and analyzing NASA's SPIDER distributed, fault-tolerant operating system [120]. We conclude with non-exhaustive formal methods. Static analysis made possible the verification of a large, complex software system that provides separation assurance between multiple airplanes up to 20 minutes ahead of time [141]. Dynamic analysis, a scalable alternative to static analysis for models with nonlinear dynamics, enabled rigorous safety-checking of the nonlinear predicates that arise from dynamics-based predictions used in alerting logic for a state-of-the-art parallel aircraft landing protocol [119]. Three runtime verification engines have been designed specifically for aerospace use-cases: NASA Langley's Copilot [251, 253], DLR's RTLola [31], and R2U2 [185, 273].

Advances in the scalability, adaptability, and connectivity of all of these tools and techniques have created an ecosystem advancing the system lifecycle of robust aerospace systems through their combined use. For example, NASA's Lunar Gateway is currently being designed from formal requirements authored as assume-guarantee contracts, verified at design time and carried through the system lifecycle all the way to on-board, real-time runtime verification [106, 107, 186].

### 3.5 Formal Methods for Operating Systems

From the beginning, formal methods have been inspired by the problems arising from multi-user operating systems, especially parallel programming and communication protocols, for which specification languages and automated protocol validation techniques based on state-space exploration have been developed since the 1970s, e.g., [274, 296]. The application field of formal methods further expanded to also encompass the verification of distributed algorithms, such as the atomic multicast protocol used for the DELTA-4 distributed dependable architecture [23].

There have been many success stories of formal methods in this area. One can mention the formal verification of the seL4 general-purpose commercial microkernel using the Isabelle/HOL theorem prover [192, 193], the SLAM verification platform based on static analysis and symbolic model checking for analyzing the source code of Microsoft Windows drivers [18–21], Microsoft's SAGE whitebox fuzzer, which found roughly one third of all the bugs discovered by file fuzzing during the development of Windows 7 [147–150], and the Coccinelle static analysis tool for automatically updating the Linux kernel and drivers [204, 205, 267].

### 3.6 Formal Methods for Cloud Security and e-Commerce

Amazon Web Services (AWS) has developed and deployed formal and automated reasoning technology for more than a decade. AWS leadership have recently described a "Golden Age" [275] for automated reasoning (AR), with AR Group founder Byron Cook noting:

> "Formal methods is transforming how Amazon Web Services (AWS) secures the cloud. Security has historically been a manual, high-judgement and thus un-scalable field; automated formal reasoning is challenging that entire structure, changing both the quality of AWS products and the cost structure to support them. The key at AWS has been to avoid "shiny-object syndrome" [99] and instead build and apply tools that quietly but reliably change the behaviour of engineers. Many leaders at AWS were skeptical of this type of work in 2016, but the success in areas such as cryptography, identity, storage and virtualization has changed minds."

A key point is that AR builds trust with customers by allowing universal and sound verification of properties of AWS' infrastructure and customers' applications. By "universal," AWS means properties that hold for all users,

[36] PRISM received the ETAPS Test-of-Time Tool Award 2024.

all storage buckets, all networks, all compute instances, all configurations, and so on — freeing the user from having to "test" a nearly infinite state space.

The deployment of AR within AWS covers a broad spectrum — from deep proofs of foundational code, through properties of protocols and internal services, to universal properties of customer-facing applications at enormous scale. A small selection of examples includes:

(1) At the foundational level, AWS has proven the memory- and type-safety of the first-stage boot code of its servers [98], giving them confidence that the code is crash-proof and resistant to code injection attacks for all possible configurations. The authors used the C Bounded Model Checker (CBMC) [89], but mention that "any other bit-precise, sound, automated static analysis tool could be used." More recently, AWS has produced a library, called "s2n-bignum," that provides primitive operations for elliptic curve field elements and points [163]. These operations underpin billions of cryptographic operations per day. s2n-bignum is the first cryptographic library that combines formal proofs of functional correctness for multiple variants of the ARM64 and x86_64 micro-architectures, resistance to timing-based side-channel attacks, and performance that is equal to or exceeds all other contemporary implementations. The s2n-bignum code [167] and proofs are freely available under a permissive licence.

(2) The AWS authorization system evaluates each request to AWS against relevant access control policies to determine if access is allowed or denied. An internal service called Zelkova [11] takes in a set of policies and uses automated reasoning to analyze every possible request that would be allowed by those policies. Under the hood, Zelkova translates each policy into a set of SMT constraints that are passed to a "portfolio" of solvers, such as the theorem provers Z3, CVC4 [25] and CVC5 [24] for solving Satisfiability Modulo Theories (SMT) in a "winner-takes-all" race. In 2019, AWS extended Zelkova to introduce "IAM Access Analyzer," which removes the burden of requiring the user to write formal access control specifications. Instead, the tool presents the user with a set of "findings" that the user can review and mark as "intended" or "not intended." This interaction is actually a form of formal specification refinement, although the user does not have to interact with the underlying formal model. By late 2022, Zelkova and its customer-facing services were generating over 1 Billion SMT queries per day [275][37].

(3) In late 2020, AWS announced the availability of strong read-after-write consistency in the S3 storage service. S3 operates at a currently preposterous scale, storing over 100 Trillion objects and handling over 10 Million requests per second [293]. Strong consistency ensures that the same view of an object is available to all readers instantly following a write operation to that object. Consistency properties were specified and verified using Dafny [209], a verification-aware programming language which uses the Z3 automated theorem prover for discharging proof obligations. To deal with continued evolution of the system, formal verification activities are built into the development team's continuous integration pipeline and run before traditional testing [97].

## 3.7 Formal Methods for Hardware Design

In the design of modern circuits (e.g., processors, co-processors, systems on chip), the largest part of the effort is dedicated to testing (which is usually called "verification" in this hardware domain) and "formal verification" (which implies the use of formal methods). A fair estimation is that more than 50% of the effort is dedicated to verification (formal or not) [72, 262], which is much higher than what is typically observed in the software industry.

The main difference between hardware and software is the impossibility to apply late patches to circuits, as usually done to fix bugs and design mistakes in software systems (such as the famous "patch Tuesday" [74]). Some degree of patching is possible for processors, e.g., by releasing updates for the firmware, or by embedding

[37] https://www.amazon.science/blog/a-billion-smt-queries-a-day

slow yet reliable fall-back algorithms, to be used as replacements for modern, highly optimized algorithms if these happen to be wrong or cause run-time errors. Yet, in most cases, patching circuits after they are shipped is nearly impossible (if these circuits are embedded in larger systems) and very expensive. For instance, the design error in the Intel Sandy Bridge chipset cost an estimated US$ 700 million [176].

Formal verification led to many success stories in the hardware-design industry, e.g., for checking the correctness of implementations of instruction sets, for checking the many communication protocols and distributed algorithms (as modern circuits heavily rely on concurrency), for checking that code generation produces correct results (this is known as "equivalence checking"), for checking that asynchronous logic performs well, etc.

Theorem proving is instrumental in the design of correct circuits. For instance, the ACL2 theorem prover[38] [190, 191] has been used by companies such as AMD, Arm, Centaur Technology, IBM, Intel, Oracle, and Collins Aerospace. For example, it gave a formal proof that the security policy of the Rockwell Collins AAMP7 microprocessor enforces a static separation kernel and is thus able to concurrently process information ranging from unclassified to top secret [155, 165].

Model checking also plays an important role in hardware verification. For instance, the Mur$\phi$ model checker [115, 285] and its many derivatives have been helpful for checking cache-coherence and security protocols. Also, the CADP model-checking toolbox [136] has been used by hardware companies such as Bull, CEA/Leti, STMicroelectronics, and Tiempo in numerous case studies (cf., e.g., [60, 101, 196, 202, 221, 301]).

CAD tool vendors (e.g., Cadence and Synopsis) provide tools that, to a certain extent, embody formal methods under the hood. High-level languages such as VHDL or SystemVerilog are used to describe the components, while languages based on temporal logic, such as SVA or PSL, are used to describe the expected properties. Further dissemination of formal methods in the hardware industry is currently limited by the insufficient number of experts, an issue that is addressed in various ways: in-house training by experienced engineers, tutorials given by CAD tool vendors, and—more recently—specific training delivered by small, dedicated service companies[39]. In [81], the specific shortage of verification engineers in the hardware design domain of microelectronics is addressed, emphasizing the importance of teaching the "verification mindset," accelerating the learning curve for verification techniques, and incorporating new paradigms like AI into the verification process.

## 3.8 Formal Methods for Lithography Manufacturing

ASML[40] is one of the world's leading manufacturers of chip-making equipment, such as lithography machines which drive Moore's Law [229] forward. Lithography machines are complex cyber-physical systems which use light to print tiny patterns on silicon; a fundamental step in mass producing microchips.

ASML's lithography machines aim to print microchip patterns as accurately and consistently as possible, even in high-volume manufacturing environments. Reliable chip manufacturing requires extremely tailored processes for each customer, so any unexpected change — even an improvement — comes at a cost. To make ASML's lithography systems run reliably and consistently ASML needs software that sends unambiguous instructions in every situation to the carefully engineered hardware. One way that ASML ensures this is by formally verifying (model checking) the specified machine behaviour and automatically generating correct and semantically equivalent code from those models [54].

To this aim, ASML uses the Coco platform [41], which integrates the imperative programming language Coco, designed for (a)synchronous event-driven software systems based on state machines, with the model checker Cosmos, designed to formally verify Coco programs (e.g., absence of deadlocks, livelocks, and race conditions,

[38]The Boyer–Moore theorem prover, a precursor to ACL2, received the 2005 ACM Software System Award.
[39]Cf., e.g., https://aedvices.com
[40]https://asml.com
[41]https://cocotec.io/

responsiveness, etc.), and is capable of generating executable code. Cosmos uses a customized process model with respect to the FDR refinement model checker for CSP that can be resolved without state-space explosion.

ASML applies this development methodology to systems ranging from high-level supervisory machine control components to low-level drivers, by expressing behaviour in many asynchronously communicating (via formally defined interfaces) state machines. Some of these state machines have billions of states (reflecting the complexity of the machine's behaviour), but even in much smaller ones there are inherent risks of issues like deadlocks and race conditions. ASML's experience is that humans have difficulty overseeing all parallel behaviour of even the smaller state machines. This is a potential risk to consistent machine operation and might even result in downtime. Formal verification helps ASML engineers to uncover many of these notoriously hard-to-find issues in an early development phase and has therefore become a critical and cost-effective design aid for ASML.

Formally verifying all desired (e.g., end-to-end) properties is currently infeasible due to various reasons, including the size and complexity of ASML's systems. Therefore, ASML incorporates runtime verification techniques in the testing process, which focus on aspects not already covered by formal verification.

The more code that is generated from formally verified models, the less chance of customers encountering bugs in ASML's software. Although that rarely happens, ASML wants it to never happen. By formally verifying more behaviour and more properties, ASML can get even closer to that goal.

## 3.9 Formal Testing of Mobile Devices from Natural Language Requirements and Other Stories from Brazil

Motorola Mobility, a Lenovo Company, has a partnership of over two decades with the Federal University of Pernambuco, in Brazil, to conceive a sound, automated, and industrial-scale testing strategy that can be applied in the mobile device domain. In the period from November 2022 to November 2023, Motorola was ranked second in the mobile vendor market share in Brazil[42] and eighth worldwide[43]. This is clearly not a safety-critical domain but rather a mission-critical domain, in the sense that escaped defects can severely affect the reputation of the company and cause significant financial losses. The overall strategy was implemented in a tool named TaRGeT [129]. It has been used by some Motorola teams that reported gains between 40% and 50% in productivity related to the overall testing process [130, 242]. Prior to this cooperation, testing was mainly a manual task in Motorola. Currently, Motorola instead adopts a formal, model-based, testing approach based on *hidden formal methods*.

The input to the developed testing strategy [243] is a text document written in a Controlled Natural Language (CNL), suitable for writing requirements, use cases and test cases, but with formal syntax and semantics. Benefitting from natural language processing techniques, a formal model (in CSP [171, 269]) is automatically derived from these requirements. Using the CSP model checker FDR [143], test cases are automatically generated as CSP traces and then translated back into CNL (for manual execution) or into scripts for several automation frameworks, for automated execution. The defined formal conformance relation is cspio, a CSP-based conformance relation distinguishing input and output based on the input/output conformance (ioco) implementation relation for input/output labelled transition systems (IOLTS) [53], formalized in the traces model of CSP. The reason for adopting an ioco-based relation is that ioco captures both partial specifications (important in the context of testing mobile devices, as the testing is on a feature-by-feature basis) and allows reduction of nondeterminism, also useful for allowing implementation choices. Many variants of ioco have been proposed in the literature (e.g., uioco, mioco, wioco, and sioco) to deal with under-specification, time, data, and so on. However, while model-based (conformance) testing has been studied intensively, today only a few tools based on variants of the ioco conformance relation are still maintained actively, such as TESTOR, implemented on top of CADP [220].

[42] https://gs.statcounter.com/vendor-market-share/mobile/brazil
[43] https://gs.statcounter.com/vendor-market-share/mobile/worldwide

Typically, the underlying formal models of test case generation approaches are IOLTS or other operational models. The main reason to use CSP as a semantic foundation for the project was that CSP, being a process algebra, offers a variety of process operators, semantic models, and process refinement notions. This provides a rich infrastructure to support the characterization of test generation at a very high level of abstraction, and, particularly, agnostic to algorithms that rely on the model structure. Test generation is accomplished using refinement assertions in the CSP traces model. The initial generation strategy considered only control flow behaviour, but, subsequently, it was straightforward to evolve it, in a conservative way, to incorporate data and time as orthogonal aspects.. Concerning the time to generate the test cases, a tool like TaRGeT is incomparably faster than designing test cases manually. Nevertheless, there are other activities in the process, beyond test design, that need to be performed both in case the tests are designed manually and when they are generated automatically. Particularly, the inspection phase takes a significant amount of time.

Currently, more elaborate frameworks are being developed with the aim of covering the full life cycle of test generation and execution. An exciting area for future investigation is the use of robotic arms to automate test execution that needs human interaction, based on AI, as well as voice, image, and natural language processing techniques.

Concerning other initiatives on the application of formal methods in Brazil, we single out a cooperation with Embraer, a Brazilian commercial aircraft company that is currently one of the largest in the world. This partnership has involved both formal verification using Simulink and the probabilistic model checker PRISM [151] and rigorous approaches to software testing [79], using so-called expanded data-flow reactive systems encoded as TIOTS, an alternative timed model based on IOLTS and ioco. Another successful cooperation was the one with Bang & Olufsen (B & O) in the context of the EU project COMPASS. The particular application was a verified design of a leadership election protocol that ensures the absence of deadlock in the (possibly dynamic) configuration of a network of B & O audio and video equipments [8], using CSP and the model checker FDR.

## 4 Educating for Formal Methods in Industry

It is our firm belief that formal methods, from formal specification to refinement and verification, constitute a core knowledge area in Computer Science with widespread relevance in many of today's innovative applications, like reliable autonomous vehicles and (robotic) systems, in a society that increasingly relies on software. Yet in most of today's Computer Science curricula, discrete mathematics and logic courses are often perceived by Computer Science students as early challenges in their education, apparently disconnected from modern programming languages. "A knowledge area directly focused on formal methods can help contextualize discrete mathematics courses for students, and can demonstrate why such courses are taught so early as a starting foundation for a solid computer science education" [70].

Formal methods do not appear in CS2023, the ACM/IEEE-CS/AAAI Computer Science Curricula[44] [197], to the extent that reflects their pivotal role in Computer Science and the benefits that formal methods education can bring to industry. CS2023 encompasses 17 knowledge areas[45]. In [34, 70, 118], it is argumented that eight of them are related to formal methods. Here we list these areas and provide suggestions for what to teach in relation with formal methods:

**Algorithmic Foundations** Teach to reason (at least informally) about the correctness of the classical algorithms (e.g., a bug was found in the TimSort sorting algorithm of the Java standard library using formal methods [153]).

[44] https://csed.acm.org/
[45] https://csed.acm.org/knowledge-areas/

**Architecture and Organization** Teach to validate the accuracy of hardware designs and that the interface behaviour of (software and hardware) components in architectural designs adhere to their specifications (e.g., by verifying security requirements in hardware security architectures [124]).

**Artificial Intelligence** Teach to capture the assumptions of the designs of deep neural networks as used in large language models as well as their verification or counterexample-based retraining (e.g., with model checking or interactive theorem proving [71]).

**Parallel and Distributed Computing** Teach how to understand and justify the correctness of systems in the presence of the topics addressed in this knowledge area (e.g., program parallelisation, atomicity, concurrency, progress, deadlocks, faults, safety, and liveness), which in essence lists formal methods as a prerequisite (viz., logic, discrete mathematics, and software engineering foundations).

**Security** Teach how to understand vulnerabilities of, and threats against, software systems, algorithms and protocols, ensuring resilience against attacks and providing assurance of security properties (including concepts like privacy, cryptography, and encryption properties [96]).

**Software Development Fundamentals** Teach to reason (at least informally) about the correctness of programs (e.g., by specifying requirements and justifying why these are met by the proposed program [230]) and to understand how algorithms impact the performance of programs.

**Databases** Teach description logic for reasoning on data management (e.g., expressing ontologies, integrating multiple data sources, and expressing and evaluating queries [59]).

**Software Engineering** Teach formal methods, which is actually recommended in this knowledge area (defined as "mathematically rigorous mechanisms to apply to software, from specification to verification") as a non-core knowledge unit with suggested learning outcomes like "describe the role formal specification and analysis techniques can play in the development of complex software and compare their use as validation and verification techniques with testing" and "apply formal specification and analysis techniques to software designs and programs with low complexity," while testing is the primary validation technique in other modules. As mentioned in Section 2.1, formal methods and testing are not mutually exclusive.

We believe in the importance of formal methods, and in particular of the capacity to abstract and mathematical reasoning that are taught as part of any formal methods course, as fundamental Computer Science skills that industry would profit from when hiring computer scientists. This is highly relevant, since we have seen that formal methods are becoming widely applied in industry. In Section 3, we have provided evidence of formal-methods applications in industry through papers and testimonies from representatives who, either directly or indirectly, use or have used formal methods in their industrial project endeavours. Importantly, they are spread geographically, including Europe, Asia, North and South America, and involve well-known worldwide companies such as Alstom, Amazon, ASML, Bang & Olufsen, Boeing, Collins Aerospace, Embraer, Facebook, Google, Huawei, IBM, Intel, Microsoft, Motorola, Oracle, Siemens, and Volvo. The current offering of formal methods in Computer Science education is inadequate because every Computer Science graduate needs to be educated in formal methods, since they can support algorithmic problem solving, model-driven engineering, requirements engineering, security, software architecture, software product lines, and many more areas of Computer Science, and they are applicable in numerous industrial domains, not limited to safety-critical applications.

This is confirmed by the aforementioned recent survey among 130 experts in formal methods, which also contained five questions on formal methods in education [138, Section 6: Formal Methods in Education]. The first two questions addressed the course level and the level of importance, while further questions concerned the content of such courses. In particular, the first question asked the experts to indicate the most suitable place for formal methods in an ideal teaching curriculum: *When and where should formal methods be taught?* A convincing 79.2% responded "in bachelor courses at the university." The second question asked the experts about the situation of formal methods in Computer Science education: *What is your opinion on the level of importance currently*

*attributed to teaching of formal methods at universities?* Exactly 50% responded "not enough attention" and 31.5% responded "sufficient attention, but scattered all over." These results indicate a consensus about the essential role of education to give the next generations of students a sufficient background and practical experience in formal methods. This is of paramount importance because in the same survey, 71.5% of the respondents identified as the single most important *limiting factor for a wider adoption of formal methods by industry* the fact that "engineers lack proper training in formal methods" [138, Section 5: Formal Methods in Industry]. This conclusion is shared by a recent white paper [82], which advocates "the inclusion of a compulsory formal methods course in Computer Science and software engineering curricula" based on the observation that "there is a lack of Computer Science graduates who are qualified to apply formal methods in industry," and by a recent textbook [268] on formal methods in software engineering, which claims that "in computer science and software engineering education, Formal Methods usually play a minor role only." In the context of safety-critical and mission-critical applications, a very recent paper recognizes "an urgent need to emphasize and integrate Formal Methods into the undergraduate curriculum in Computer Science in the United States," since "the lack of a well-structured exposure to formal methods is a serious shortcoming in our computing curricula" [261]. The authors also provide several concrete suggestions for introducing the concepts and use of formal methods into existing Computer Science curricula (e.g., Data Structures, Logic circuit design, Concepts of programming languages, Software Engineering). "We cannot expect graduates to become experts in program verification as professionals if they never encountered the ideas as students." The authors of [105] propose to approach formal methods already in basic education (i.e., primary and secondary education) through five fundamental notions (viz., specification, formalization, modelling, verification, and reasoning) and they report on their experience in doing so (by means of gamification of transformation rules of typed graph grammars using Pac-Man).

Support for teachers is available, for instance through recent textbooks on formal methods [43, 175, 240, 246, 268] and advanced lectures on formal methods [44–48, 263], but also via Formal Methods Europe (FME)[46] and the ERCIM working group on Formal Methods for Industrial Critical Systems (FMICS)[47].

## 5 Conclusion

We have demonstrated that formal methods are important to quite a number of industry segments, not limited to safety-critical domains, and we have made a case for the inclusion of formal methods as a separate topic in Computer Science education. This strengthens the evidence put forward in [70] for claiming that formal methods should be taught as a separate topic in undergraduate curricula, not only because of their importance in industry but also because of the discipline they instil in students as they learn to develop systems through abstraction and mathematical reasoning, as demonstrated in [118, 261]. Moreover, we have shown that this can be done without displacing the other "engineering" aspects of Computer Science already widely accepted as essential. On the contrary, we have shown that formal methods have the potential to support and strengthen at least eight of the 17 knowledge areas of CS2023.

The formal methods community recently received support from a rather unexpected source. The White House advocates the use of formal methods over testing for demonstrating the correctness of software and considers it vital to make formal methods widely accessible to accelerate broad adoption [297, Part II: Securing the Building Blocks of Cyberspace—Formal Methods]: "Given the complexities of code, testing is a necessary but insufficient step in the development process to fully reduce vulnerabilities at scale. If correctness is defined as the ability of a piece of software to meet a specific security requirement, then it is possible to demonstrate correctness using mathematical techniques called *formal methods*. These techniques, often used to prove a range of software outcomes, can also be used in a cybersecurity context and are viable even in complex environments like space.

[46] https://fmeurope.org/
[47] https://fmics.inria.fr/

While formal methods have been studied for decades, their deployment remains limited; further innovation in approaches to make formal methods widely accessible is vital to accelerate broad adoption. Doing so enables formal methods to serve as another powerful tool to give software developers greater assurance that entire classes of vulnerabilities, even beyond memory safety bugs, are absent." This report highlights *static analysis* and *model checkers* as specific examples of types of formal methods.

To conclude, creating correct software is an engineering problem and software development should therefore be an engineering discipline. Mastering the complexity of software systems is a formidable intellectual challenge and Computer Science graduates need to understand the powerful formal methods and tools that are available. Then they can choose the right technique and tool for each task and with suitable humility deserve the title of *Software Engineer*.

## Acknowledgments

Sadly, our co-author Rance Cleaveland passed away unexpectedly on March 27, 2024, during the revision of this paper. Rance has written most of the Introduction and he was very passionate about this paper:

"I wanted to, on the one hand, give a sense of the history of formal methods but also tie the rest of the paper together. I also wanted to make a pedagogical point, which I don't see right now in the paper: that students who learn formal methods are better developers, because they learn to think about correctness while they are building systems."

"As to whether the point that FM is only for safety-critical system comes out in the rest of the paper, I would say the safety-critical aspects still seem to dominate a bit, although other industrial applications also are apparent. However, having said that, I think the ACM CS curricula committee's position is, frankly, silly. There are lots of things we teach in CS curricula that don't have "broad industrial relevance": functional programming, computer architecture / organization, O(-) notation, etc. And yet these topics are not controversial, because there is an agreement that students who learn these topics are better computer scientists. Even if FM is *only* for safety-critical systems, there is a great case to make for its inclusion in CS curricula."

We honor Rance with this paper and acknowledge his seminal contributions to formal methods and concurrency [284], in particular the development and application of process-algebraic modelling and verification techniques and tools, like the Concurrency Workbenches [91–93].

Several domain experts provided us with useful information or feedback. June Andronick from Proofcraft (Australia) checked the references to the formal verification of the seL4 general-purpose commercial microkernel. Gunnar Smith from Prover Technology (Sweden) provided many meaningful experiences from an engineer's perspective and successful stories of Prover using formal methods. Finally, we thank Manfred Broy and the four anonymous reviewers for their valuable comments on earlier versions of this paper.



## References

[1] Robert Abo and Laurent Voisin. 2013. Formal Implementation of Data Validation for Railway Safety-Related Systems with OVADO. In *Revised Selected Papers of the SEFM 2013 Collocated Workshops: BEAT2, WS-FMDS, FM-RAIL-Bok, MoKMaSD, and OpenCert (LNCS, Vol. 8368)*, Steve Counsell and Manuel Núñez (Eds.). Springer, Germany, 221–236. https://doi.org/10.1007/978-3-319-05032-4_17

[2] Jean-Raymond Abrial. 1996. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, UK. https://doi.org/10.1017/CBO9780511624162

[3] Jean-Raymond Abrial. 2010. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, UK. https://doi.org/10.1017/CBO9781139195881

[4] Aeronautical Radio Inc. (ARINC) Airlines Electronic Engineering Committee. 2015. ARINC 653: Avionics Application Software Standard Interface, Part 1 — Required Services. https://www.sae.org/standards/content/arinc653p1-4/

[5] Gul Agha and Karl Palmskog. 2018. A Survey of Statistical Model Checking. *ACM Trans. Model. Comput. Simul.* 28, 1 (2018), 6:1–6:39. https://doi.org/10.1145/3158668

[6] Ehsan Ahmad, Yunwei Dong, Brian R. Larson, Jidong Lü, Tao Tang, and Naijun Zhan. 2015. Behavior modeling and verification of movement authority scenario of Chinese Train Control System using AADL. *Sci. China Inf. Sci.* 58, 11 (2015), 1–20. https:

//doi.org/10.1007/s11432-015-5346-2

[7] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. 2011. An Overview of Formal Methods Tools and Techniques. In *Rigorous Software Development: An Introduction to Program Verification*. Springer, Germany, 15–44. https://doi.org/10.1007/978-0-85729-018-2_2

[8] Pedro R.G. Antonino, Marcel Medeiros Oliveira, Augusto C.A. Sampaio, Klaus E. Kristensen, and Jeremy W. Bryans. 2014. Leadership Election: An Industrial SoS Application of Compositional Deadlock Verification. In *Proceedings of the 6th International NASA Formal Methods Symposium (NFM'14) (LNCS, Vol. 8430)*, Julia M. Badger and Kristin Y. Rozier (Eds.). Springer, Germany, 31–45. https://doi.org/10.1007/978-3-319-06200-6_3

[9] Muhammad Atif and Jan Friso Groote. 2023. *Understanding Behaviour of Distributed Systems Using mCRL2*. Studies in Systems, Decision and Control, Vol. 458. Springer, Germany. https://doi.org/10.1007/978-3-031-23008-0

[10] Ralph-Johan R. Back. 1980. *Correctness Preserving Program Refinements: Proof Theory and Applications*. Mathematical Centre Tracts, Vol. 131. Mathematisch Centrum, The Netherlands.

[11] John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek, Kasper Søe Luckow, Neha Rungta, Oksana Tkachuk, and Carsten Varming. 2018. Semantic-based Automated Reasoning for AWS Access Policies using SMT. In *Proceedings of the 18th Conference on Formal Methods in Computer-Aided Design (FMCAD'18)*, Nikolaj S. Bjørner and Arie Gurfinkel (Eds.). IEEE, USA, 1–9. https://doi.org/10.23919/FMCAD.2018.8602994

[12] John Backes, Pauline Bolignano, Byron Cook, Andrew Gacek, Kasper Søe Luckow, Neha Rungta, Martin Schäf, Cole Schlesinger, Rima Tanash, Carsten Varming, and Michael W. Whalen. 2019. One-Click Formal Methods. *IEEE Softw.* 36, 6 (2019), 61–65. https://doi.org/10.1109/MS.2019.2930609

[13] Frédéric Badeau, Julien Chappelin, and Joris Lamare. 2022. Generating and Verifying Configuration Data with OVADO. In *Proceedings of the 4th International Conference on Reliability, Safety, and Security of Railway Systems: Modelling, Analysis, Verification, and Certification (RSSRail'22) (LNCS, Vol. 13294)*, Simon Collart-Dutilleul, Anne E. Haxthausen, and Thierry Lecomte (Eds.). Springer, Germany, 143–148. https://doi.org/10.1007/978-3-031-05814-1_10

[14] Tom Badgett, Corey Sandler, and Glenford J. Myers. 2015. *The Art of Software Testing*. Wiley, UK. https://www.wiley.com/en-gb/The+Art+of+Software+Testing%2C+3rd+Edition-p-x000565567

[15] Jos C. M. Baeten and W. Peter Weijland. 1990. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science, Vol. 18. Cambridge University Press, UK. https://doi.org/10.1017/CBO9780511624193

[16] Christel Baier, Luca de Alfaro, Vojtech Forejt, and Marta Kwiatkowska. 2018. Model Checking Probabilistic Systems. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, Germany, 963–999. https://doi.org/10.1007/978-3-319-10575-8_28

[17] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press, USA. https://mitpress.mit.edu/9780262026499/principles-of-model-checking/

[18] Thomas Ball, Ella Bounimova, Rahul Kumar, and Vladimir Levin. 2010. SLAM2: Static Driver Verification with Under 4% False Alarms. In *Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design (FMCAD'10)*, Roderick Bloem and Natasha Sharygina (Eds.). IEEE, USA, 35–42. https://ieeexplore.ieee.org/document/5770931/

[19] Thomas Ball, Ella Bounimova, Vladimir Levin, Rahul Kumar, and Jakob Lichtenberg. 2010. The Static Driver Verifier Research Platform. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10) (LNCS, Vol. 6174)*, Tayssir Touili, Byron Cook, and Paul B. Jackson (Eds.). Springer, Germany, 119–122. https://doi.org/10.1007/978-3-642-14295-6_11

[20] Thomas Ball, Sagar Chaki, and Sriram K. Rajamani. 2001. Parameterized Verification of Multithreaded Software Libraries. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01) (LNCS, Vol. 2031)*, Tiziana Margaria and Wang Yi (Eds.). Springer, Germany, 158–173. https://doi.org/10.1007/3-540-45319-9_12

[21] Thomas Ball, Vladimir Levin, and Sriram K. Rajamani. 2011. A Decade of Software Model Checking with SLAM. *Commun. ACM* 54, 7 (2011), 68–76. https://doi.org/10.1145/1965724.1965743

[22] Fabrizio Banci Buonamici, Gina Belmonte, Vincenzo Ciancia, Diego Latella, and Mieke Massink. 2020. Spatial logics and model checking for medical imaging. *Int. J. Softw. Tools Technol. Transf.* 22, 2 (2020), 195–217. https://doi.org/10.1007/s10009-019-00511-9

[23] Mário Baptista, Susanne Graf, Jean-Luc Richier, Luís E. T. Rodrigues, Carlos Rodriguez, Paulo Veríssimo, and Jacques Voiron. 1991. Formal Specification and Verification of a Network Independent Atomic Multicast Protocol. In *Proceedings of the 3rd IFIP TC6/WG6.1 International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'90)*, Juan Quemada, José A. Mañas, and Enrique Vázquez (Eds.). North-Holland, The Netherlands, 345–352.

[24] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'22) (LNCS, Vol. 13243)*, Dana Fisman and Grigore Rosu (Eds.). Springer, Germany, 415–442. https://doi.org/10.1007/978-3-030-99524-9_24

[25] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11) (LNCS, Vol. 6806)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, Germany, 171–177. https://doi.org/10.1007/978-3-642-22110-1_14

[26] Gilles Barthe, Sandrine Blazy, Benjamin Grégoire, Rémi Hutin, Vincent Laporte, David Pichardie, and Alix Trieu. 2020. Formal verification of a constant-time preserving C compiler. *Proc. ACM Program. Lang.* 4, POPL (2020), 7:1–7:30. https://doi.org/10.1145/3371075

[27] Ezio Bartocci and Yliès Falcone (Eds.). 2018. *Lectures on Runtime Verification: Introductory and Advanced Topics.* LNCS, Vol. 10457. Springer, Germany. https://doi.org/10.1007/978-3-319-75632-5

[28] Davide Basile, Alessandro Fantechi, Luigi Rucher, and Gianluca Mandò. 2021. Analysing an autonomous tramway positioning system with the Uppaal Statistical Model Checker. *Form. Asp. Comput.* 33, 6 (2021), 957–987. https://doi.org/10.1007/s00165-021-00556-1

[29] Davide Basile, Maurice H. ter Beek, Alessandro Fantechi, Stefania Gnesi, Franco Mazzanti, Andrea Piattino, Daniele Trentini, and Alessio Ferrari. 2018. On the Industrial Uptake of Formal Methods in the Railway Domain. In *Proceedings of the 14th International Conference on Integrated Formal Methods (iFM'18) (LNCS, Vol. 11023)*, Carlo A. Furia and Kirsten Winter (Eds.). Springer, Germany, 20–29. https://doi.org/10.1007/978-3-319-98938-9_2

[30] Davide Basile, Maurice H. ter Beek, Alessio Ferrari, and Axel Legay. 2022. Exploring the ERTMS/ETCS full moving block specification: An experience with formal methods. *Int. J. Softw. Tools Technol. Transf.* 24, 3 (2022), 351–370. https://doi.org/10.1007/s10009-022-00653-3

[31] Jan Baumeister, Bernd Finkbeiner, Sebastian Schirmer, Maximilian Schwenger, and Christoph Torens. 2020. RTLola Cleared for Take-Off: Monitoring Autonomous Aircraft. In *Proceedings of the 32nd International Conference on Computer Aided Verification (CAV'20) (LNCS, Vol. 12225)*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, Germany, 28–39. https://doi.org/10.1007/978-3-030-53291-8_3

[32] Maurice H. ter Beek. 2024. Formal Methods and Tools Applied in the Railway Domain. In *Proceedings of the 10th International Conference on Rigorous State Based Methods (ABZ'24) (LNCS, Vol. 14759)*, Silvia Bonfanti, Angelo Gargantini, Michael Leuschel, Elvinia Riccobene, and Patrizia Scandurra (Eds.). Springer, Germany, 3–21. https://doi.org/10.1007/978-3-031-63790-2_1

[33] Maurice H. ter Beek, Arne Borälv, Alessandro Fantechi, Alessio Ferrari, Stefania Gnesi, Christer Löfving, and Franco Mazzanti. 2019. Adopting Formal Methods in an Industrial Setting: The Railways Case. In *Proceedings of the 3rd World Congress on Formal Methods: The Next 30 Years (FM'19) (LNCS, Vol. 11800)*, Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira (Eds.). Springer, Germany, 762–772. https://doi.org/10.1007/978-3-030-30942-8_46

[34] Maurice H. ter Beek, Manfred Broy, and Brijesh Dongol. 2024. CS2023: The Role of Formal Methods in Computer Science Education. *ACM InRoads* (2024).

[35] Maurice H. ter Beek and Alessio Ferrari. 2022. Empirical Formal Methods: Guidelines for Performing Empirical Studies on Formal Methods. *Softw.* 1, 4 (2022), 381–416. https://doi.org/10.3390/software1040017

[36] Maurice H. ter Beek, Stefania Gnesi, and Alexander Knapp. 2018. Formal methods for transport systems. *Int. J. Softw. Tools Technol. Transf.* 20, 3 (2018), 355–358. https://doi.org/10.1007/s10009-018-0487-4

[37] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. 1999. Météor: A Successful Application of B in a Large Project. In *Proceedings of the 1st World Congress on Formal Methods in the Development of Computing Systems (FM'99) (LNCS, Vol. 1708)*, Jeannette M. Wing, Jim Woodcock, and Jim Davies (Eds.). Springer, Germany, 369–387. https://doi.org/10.1007/3-540-48119-2_22

[38] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. 2007. UPPAAL-Tiga: Time for Playing Games!. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07) (LNCS, Vol. 4590)*, Werner Damm and Holger Hermanns (Eds.). Springer, Germany, 121–125. https://doi.org/10.1007/978-3-540-73368-3_14

[39] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. 2006. UPPAAL 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems (QEST'06).* IEEE, USA, 125–126. https://doi.org/10.1109/QEST.2006.59

[40] Dimitri Belli, Alessandro Fantechi, Stefania Gnesi, Laura Masullo, Frando Mazzanti, Lisa Quadrini, Daniele Trentini, and Carlo Vaghi. 2023. The 4SECURail Case Study on Rigorous Standard Interface Specifications. In *Proceedings of the 28th International Conference on Formal Methods for Industrial Critical Systems (FMICS'23) (LNCS, Vol. 14290)*, Alessandro Cimatti and Laura Titolo (Eds.). Springer, Germany, 22–39. https://doi.org/10.1007/978-3-031-43681-9_2

[41] Gina Belmonte, Giovanna Broccia, Vincenzo Ciancia, Diego Latella, and Mieke Massink. 2021. Feasibility of Spatial Model Checking for Nevus Segmentation. In *Proceedings of the 9th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormaliSE'21).* IEEE, USA, 1–12. https://doi.org/10.1109/FormaliSE52586.2021.00007

[42] Gina Belmonte, Vincenzo Ciancia, Diego Latella, Mieke Massink, Michelangelo Biondi, Gianmarco De Otto, Valerio Nardone, Giovanni Rubino, Eleonora Vanzi, and Fabrizio Banci Buonamici. 2017. A topological method for automatic segmentation of glioblastoma in MR FLAIR for radiotherapy. *Magn. Reson. Mater. Phys. Biol. Med.* 30, Suppl 1 (2017), S437–S438. https://doi.org/10.1007/s10334-017-0634-z Proceedings of the 34th Annual Scientific Meeting of the European Society for Magnetic Resonance in Medicine and Biology (ESMRMB'17).

[43] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. 2017. *Formal Methods for Discrete-Time Dynamical Systems.* Springer, Germany. https://doi.org/10.1007/978-3-319-50763-7

[44] Marco Bernardo and Alessandro Cimatti (Eds.). 2006. *Formal Methods for Hardware Verification: Advanced Lectures of the 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'06)*. LNCS, Vol. 3965. Springer, Germany. https://doi.org/10.1007/11757283

[45] Marco Bernardo, Vittorio Cortellessa, and Alfonso Pierantonio (Eds.). 2012. *Formal Methods for Executable Software Models: Advanced Lectures of the 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'12)*. LNCS, Vol. 7320. Springer, Germany. https://doi.org/10.1007/978-3-642-30982-3

[46] Marco Bernardo, Ferruccio Damiani, Reiner Hähnle, Einar Broch Johnsen, and Ina Schaefer (Eds.). 2014. *Formal Methods for Executable Software Models: Advanced Lectures of the 14th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'14)*. LNCS, Vol. 8483. Springer, Germany. https://doi.org/10.1007/978-3-319-07317-0

[47] Marco Bernardo and Jane Hillston (Eds.). 2007. *Formal Methods for Performance Evaluation: Advanced Lectures of the 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'07)*. LNCS, Vol. 4486. Springer, Germany. https://doi.org/10.1007/978-3-540-72522-0

[48] Marco Bernardo, Rocco De Nicola, and Jane Hillston (Eds.). 2016. *Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems: Advanced Lectures of the 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'16)*. LNCS, Vol. 9700. Springer, Germany. https://doi.org/10.1007/978-3-319-34096-8

[49] Gérard Berry, Amar Bouali, Xavier Fornari, Emmanuel Ledinot, Eric Nassor, and Robert de Simone. 2000. ESTEREL: a formal method applied to avionic software development. *Sci. Comput. Program.* 36, 1 (2000), 5–25. https://doi.org/10.1016/S0167-6423(99)00015-5

[50] Antonia Bertolino. 2007. Software Testing Research: Achievements, Challenges, Dreams. In *Proceedings of the ICSE 2007 Workshop on the Future of Software Engineering (FoSE'07)*, Lionel C. Briand and Alexander L. Wolf (Eds.). IEEE, USA, 85–103. https://doi.org/10.1109/FOSE.2007.25

[51] Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development — Coq'Art: The Calculus of Inductive Constructions*. Springer, Germany. https://doi.org/10.1007/978-3-662-07964-5

[52] Aysu Betin Can, Tevfik Bultan, Mikael Lindvall, Benjamin Lux, and Stefan Topp. 2007. Eliminating synchronization faults in air traffic control software via design for verification with concurrency controllers. *Autom. Softw. Eng.* 14, 2 (2007), 129–178. https://doi.org/10.1007/s10515-007-0008-2

[53] Machiel van der Bijl, Arend Rensink, and Jan Tretmans. 2003. Compositional Testing with ioco. In *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES'03) (LNCS, Vol. 2931)*, Alexandre Petrenko and Andreas Ulrich (Eds.). Springer, Germany, 86–100. https://doi.org/10.1007/978-3-540-24617-6_7

[54] Lewis Binns. 2023. By computers, for computers: Improving scanner metrology software with generated code. https://www.linkedin.com/pulse/computers-improving-scanner-metrology-software-code-lewis/

[55] Sandrine Blazy, Zaynah Dargaye, and Xavier Leroy. 2006. Formal Verification of a C Compiler Front-End. In *Proceedings on the 14th International Symposium on Formal Methods (FM'06) (LNCS, Vol. 4085)*, Jayadev Misra, Tobias Nipkow, and Emil Sekerinski (Eds.). Springer, Germany, 460–475. https://doi.org/10.1007/11813040_31

[56] Hans Blom, Henrik Lönn, Frank Hagl, Yiannis Papadopoulos, Mark-Oliver Reiser, Carl-Johan Sjöstedt, De-Jiu Chen, and Ramin Tavakoli Kolagari. 2013. EAST-ADL – An Architecture Description Language for Automotive Software-Intensive Systems. White Paper. https://maenad.eu/public/conceptpresentations/EAST-ADL_WhitePaper_M2.1.12.pdf

[57] Tommaso Bolognesi and Ed Brinksma. 1987. Introduction to the ISO Specification Language LOTOS. *Comput. Netw.* 14 (1987), 25–59. https://doi.org/10.1016/0169-7552(87)90085-7

[58] Paulo Borba, Ana Cavalcanti, Augusto Sampaio, and Jim Woodcock (Eds.). 2010. *Testing Techniques in Software Engineering: Revised Lectures of the 2nd Pernambuco Summer School on Software Engineering (PSSE'07)*. LNCS, Vol. 6153. Springer, Germany. https://doi.org/10.1007/978-3-642-14335-9

[59] Alexander Borgida, Maurizio Lenzerini, and Riccardo Rosati. 2007. Description Logics for Databases. In *The Description Logic Handbook: Theory, Implementation, and Applications*, Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). Cambridge University Press, UK, Chapter 16, 500–524. https://doi.org/10.1017/CBO9780511711787.018

[60] Aymane Bouzafour, Marc Renaudin, Hubert Garavel, Radu Mateescu, and Wendelin Serwe. 2018. Model-Checking Synthesizable SystemVerilog Descriptions of Asynchronous Circuits. In *24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'18)*. IEEE, USA, 34–42. https://doi.org/10.1109/ASYNC.2018.00021

[61] Jonathan P. Bowen, Ricky W. Butler, David L. Dill, Robert L. Glass, David Gries, Anthony Hall, Michael G. Hinchey, C. Michael Holloway, Daniel Jackson, Cliff B. Jones, Michael J. Lutz, David L. Parnas, John M. Rushby, Jeannette M. Wing, and Pamela Zave. 1996. An Invitation to Formal Methods. *IEEE Comput.* 29, 4 (1996), 16–30. https://doi.org/10.1109/MC.1996.488298

[62] Jonathan P. Bowen and Michael G. Hinchey. 1995. Seven More Myths of Formal Methods. *IEEE Softw.* 12, 4 (1995), 34–41. https://doi.org/10.1109/52.391826

[63] Jonathan P. Bowen and Michael G. Hinchey. 1995. Ten Commandments of Formal Methods. *IEEE Comput.* 28, 4 (1995), 56–63. https://doi.org/10.1109/2.375178

[64] Jonathan P. Bowen and Michael G. Hinchey. 2006. Ten Commandments of Formal Methods ...Ten Years Later. *IEEE Comput.* 39, 1 (2006), 40–48. https://doi.org/10.1109/MC.2006.35

[65] Jonathan P. Bowen and Victoria Stavridou. 1993. The Industrial Take-up of Formal Methods in Safety-Critical and Other Areas: A Perspective. In *Proceedings of the 1st International Symposium of Formal Methods Europe (FME'93) (LNCS, Vol. 670)*, Jim Woodcock and Peter Gorm Larsen (Eds.). Springer, Germany, 183–195. https://doi.org/10.1007/BFb0024646

[66] Robert S. Boyer and J. Strother Moore. 1979. *A Computational Logic Handbook*. Perspectives in Computing, Vol. 23. Academic Press, USA.

[67] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. 2009. The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In *Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security (SAFECOMP'09) (LNCS, Vol. 5775)*, Bettina Buth, Gerd Rabe, and Till Seyfarth (Eds.). Springer, Germany, 173–186. https://doi.org/10.1007/978-3-642-04468-7_15

[68] Marco Bozzano, Alessandro Cimatti, Anthony Fernandes Pires, David Jones, Greg Kimberly, Tyler Petri, Richard Robinson, and Stefano Tonetta. 2015. Formal Design and Safety Analysis of AIR6110 Wheel Brake System. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV'15) (LNCS, Vol. 9206)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, Germany, 518–535. https://doi.org/10.1007/978-3-319-21690-4_36

[69] Robert K. Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10) (LNCS, Vol. 6174)*, Tayssir Touili, Byron Cook, and Paul B. Jackson (Eds.). Springer, Germany, 24–40. https://doi.org/10.1007/978-3-642-14295-6_5

[70] Manfred Broy, Achim Brucker, Alessandro Fantechi, Mario Gleirscher, Klaus Havelund, Markus Alexander Kuppe, Alexandra Mendes, André Platzer, Jan Ringert, and Allison Sullivan. 2024. Does Every Computer Scientist Need to Know Formal Methods? *Form. Asp. Comput.* (2024). https://doi.org/10.1145/36707

[71] Achim D. Brucker and Amy Stell. 2023. Verifying Feedforward Neural Networks for Classification in Isabelle/HOL. In *Proceedings of the 25th International Symposium on Formal Methods (FM'23) (LNCS, Vol. 14000)*, Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker (Eds.). Springer, Germany, 427–444. https://doi.org/10.1007/978-3-031-27481-7_24

[72] Jean-Marie Brunet. 2023. A Systematic Approach to Verification & Validation Using Hardware-Assisted Verification. Global Semiconductor Alliance. https://www.gsaglobal.org/forums/a-systematic-approach-to-verification-validation-using-hardware-assisted-verification

[73] Antonio Bucchiarone, Jordi Cabot, Richard F. Paige, and Alfonso Pierantonio. 2020. Grand challenges in model-driven engineering: an analysis of the state of the research. *Softw. Syst. Model.* 19, 1 (2020), 5–13. https://doi.org/10.1007/S10270-019-00773-6

[74] Christopher Budd. 2013. Ten Years of Patch Tuesdays: Why It's Time to Move On. GeekWire. https://www.geekwire.com/2013/ten-years-patch-tuesdays-time-move/

[75] Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. 2019. The mCRL2 Toolset for Analysing Concurrent Systems: Improvements in Expressivity and Usability. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'19) (LNCS, Vol. 11428)*, T. Vojnar and L. Zhang (Eds.). Springer, Germany, 21–39. https://doi.org/10.1007/978-3-030-17465-1_2

[76] Michael Butler, Thai Son Hoang, Alexander Raschke, and Klaus Reichl. 2020. Introduction to special section on the ABZ 2018 case study: Hybrid ERTMS/ETCS Level 3. *Int. J. Softw. Tools Technol. Transf.* 22, 3 (2020), 249–255. https://doi.org/10.1007/s10009-020-00562-3

[77] Michael Butler, Philipp Körner, Sebastian Krings, Thierry Lecomte, Michael Leuschel, Luis-Fernando Mejia, and Laurent Voisin. 2020. The First Twenty-Five Years of Industrial Use of the B-Method. In *Proceedings of the 25th International Conference on Formal Methods for Industrial Critical Systems (FMICS'20) (LNCS, Vol. 12327)*, Maurice H. ter Beek and Dejan Ničković (Eds.). Springer, Germany, 189–209. https://doi.org/10.1007/978-3-030-58298-2_8

[78] Ricky W. Butler, George E. Hagen, and Jeffrey M. Maddalon. 2013. *The Chorus Conflict and Loss of Separation Resolution Algorithms*. Technical Report NASA/TM–2013-218030. NASA. https://ntrs.nasa.gov/citations/20140001006

[79] Gustavo Carvalho, Ana Cavalcanti, and Augusto Sampaio. 2016. Modelling timed reactive systems from natural-language requirements. *Form. Asp. Comput.* 28, 5 (2016), 725–765. https://doi.org/10.1007/S00165-016-0387-X

[80] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. 2014. The nuXmv Symbolic Model Checker. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV'14) (LNCS, Vol. 8559)*, Armin Biere and Roderick Bloem (Eds.). Springer, Germany, 334–342. https://doi.org/10.1007/978-3-319-08867-9_22

[81] François Cerisier. 2023. How to build the future verification engineers? Verification Futures Conference (VF'23). https://www.tessolve.com/wp-content/uploads/2023/06/2-Francois-Cerisier-2023-How-to-build-the-future-verification-engineers-Francois-Cerisier-AEDVICES-V1-1.pdf

[82] Antonio Cerone, Markus Roggenbach, James Davenport, Casey Denner, Marie Farrell, Magne Haveraaen, Faron Moller, Philipp Körner, Sebastian Krings, Peter Csaba Ölveczky, Bernd-Holger Schlingloff, Nikolay Shilov, and Rustam Zhumagambetov. 2021. Rooting Formal Methods Within Higher Education Curricula for Computer Science and Software Engineering – A White Paper. In *Revised Selected*

*Papers of the 1st International Workshop on Formal Methods – Fun for Everybody (FMFun'19) (CCIS, Vol. 1301)*, Antonio Cerone and Markus Roggenbach (Eds.). Springer, Germany, 1–26. https://doi.org/10.1007/978-3-030-71374-4_1

[83] William Chan, Richard J. Anderson, Paul Beame, Steve Burns, Francesmary Modugno, David Notkin, and Jon Damon Reese. 1998. Model Checking Large Software Specifications. *IEEE Trans. Softw. Eng.* 24, 7 (1998), 498–520. https://doi.org/10.1109/32.708566

[84] Roderick Chapman and Florian Schanda. 2014. Are We There Yet? 20 Years of Industrial Theorem Proving with SPARK. In *Proceedings of the 5th International Conference on Interactive Theorem Proving (ITP'14) (LNCS, Vol. 8558)*, Gerwin Klein and Ruben Gamboa (Eds.). Springer, Germany, 17–26. https://doi.org/10.1007/978-3-319-08970-6_2

[85] Angelo Chiappini, Alessandro Cimatti, Luca Macchi, Oscar Rebollo, Marco Roveri, Angelo Susi, Stefano Tonetta, and Berardino Vittorini. 2010. Formalization and validation of a subset of the European Train Control System. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE'10)*. ACM, USA, 109–118. https://doi.org/10.1145/1810295.1810312

[86] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02) (LNCS, Vol. 2404)*, Ed Brinksma and Kim G. Larsen (Eds.). Springer, Germany, 359–364. https://doi.org/10.1007/3-540-45657-0_29

[87] Edmund M. Clarke and E. Allen Emerson. 1981. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Proceedings of the 1981 Workshop on Logics of Programs (LNCS, Vol. 131)*, Dexter Kozen (Ed.). Springer, Germany, 52–71. https://doi.org/10.1007/BFB0025774

[88] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). 2018. *Handbook of Model Checking*. Springer, Germany. https://doi.org/10.1007/978-3-319-10575-8

[89] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. 2004. A Tool for Checking ANSI-C Programs. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04) (LNCS, Vol. 2988)*, Kurt Jensen and Andreas Podelski (Eds.). Springer, Germany, 168–176. https://doi.org/10.1007/978-3-540-24730-2_15

[90] Edmund M. Clarke, Jeannette M. Wing, et al. 1996. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.* 28, 4 (1996), 626–643. https://doi.org/10.1145/242223.242257

[91] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. 1993. The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems. *ACM Trans. Program. Lang. Syst.* 15, 1 (1993), 36–72. https://doi.org/10.1145/151646.151648

[92] Rance Cleaveland, A. W. (Bill) Roscoe, and Scott A. Smolka. 2018. Process Algebra and Model Checking. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, Germany, Chapter 32, 1149–1195. https://doi.org/10.1007/978-3-319-10575-8_32

[93] Rance Cleaveland and Steve Sims. 1996. The NCSU Concurrency Workbench. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96) (LNCS, Vol. 1102)*, Rajeev Alur and Thomas A. Henzinger (Eds.). Springer, Germany, 394–397. https://doi.org/10.1007/3-540-61474-5_87

[94] Christian Colombo and Gordon J. Pace. 2022. *Runtime Verification*. Springer, Germany. https://doi.org/10.1007/978-3-031-09268-8

[95] Mathieu Comptier, Michael Leuschel, Luis-Fernando Mejia, Julien Molinero Perez, and Mareike Mutz. 2019. Property-Based Modelling and Validation of a CBTC Zone Controller in Event-B. In *Proceedings of the 3rd International Conference on Reliability, Safety, and Security of Railway Systems: Modelling, Analysis, Verification, and Certification (RSSRail'19) (LNCS, Vol. 11495)*, Simon Collart-Dutilleul, Thierry Lecomte, and Alexander B. Romanovsky (Eds.). Springer, Germany, 202–212. https://doi.org/10.1007/978-3-030-18744-6_13

[96] Byron Cook. 2018. Formal Reasoning About the Security of Amazon Web Services. In *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18) (LNCS, Vol. 10981)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer, Germany, 38–47. https://doi.org/10.1007/978-3-319-96145-3_3

[97] Byron Cook. 2022. Automated reasoning's scientific frontiers. https://www.amazon.science/blog/automated-reasonings-scientific-frontiers

[98] Byron Cook, Kareem Khazem, Daniel Kroening, Serdar Tasiran, Michael Tautschnig, and Mark R. Tuttle. 2018. Model Checking Boot Code from AWS Data Centers. In *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18) (LNCS, Vol. 10982)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer, Germany, 467–486. https://doi.org/10.1007/978-3-319-96142-2_28

[99] Jodie Cook. 2023. Shiny Object Syndrome: The Biggest Problem For Today's Entrepreneurs. Forbes. https://www.forbes.com/sites/jodiecook/2023/02/20/shiny-object-syndrome-the-biggest-problem-for-todays-entrepreneurs/?sh=5a90cb4b6709

[100] Thierry Coquand and Gérard P. Huet. 1985. Constructions: A Higher Order Proof System for Mechanizing Mathematics. In *Proceedings of the European Conference on Computer Algebra (EUROCAL'85) (LNCS, Vol. 203)*, Bruno Buchberger (Ed.). Springer, Germany, 151–184. https://doi.org/10.1007/3-540-15983-5_13

[101] Nicolas Coste, Holger Hermanns, Etienne Lantreibecq, and Wendelin Serwe. 2009. Towards Performance Prediction of Compositional Models in Industrial GALS Designs. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09) (LNCS, Vol. 5643)*, Ahmed Bouajjani and Oded Maler (Eds.). Springer, Germany, 204–218. https://doi.org/10.1007/978-3-642-02658-4_18

[102] Jean-Pierre Courtiat, Piotr Dembinski, Gerard J. Holzmann, Luigi Logrippo, Harry Rudin, and Pamela Zave. 1996. Formal methods after 15 years: Status and trends. *Comput. Netw. ISDN Syst.* 28, 13 (1996), 1845–1855. https://doi.org/10.1016/0169-7552(96)00083-9

[103] Patrick Cousot. 2021. *Principles of Abstract Interpretation.* MIT Press, USA. https://mitpress.mit.edu/9780262044905/principles-of-abstract-interpretation/

[104] Russ Cox, Robert Griesemer, Rob Pike, Ian Lance Taylor, and Ken Thompson. 2022. The Go programming language and environment. *Commun. ACM* 65, 5 (2022), 70–78. https://doi.org/10.1145/3488716

[105] Braz Araujo da Silva Junior, Simone André da Costa Cavalheiro, Luciana Foss, and Júlia Veiga da Silva. 2023. Formal Specification in Basic Education: What Does It Take?. In *Proceedings of the 53rd IEEE/ASEE International Conference on Frontiers in Education (FIE'23)*. IEEE, USA, 1–9. https://doi.org/10.1109/FIE58773.2023.10343074

[106] James B. Dabney, Julia M. Badger, and Pavan Rajagopal. 2021. Adding a Verification View for an Autonomous Real-Time System Architecture. In *Proceedings of the 2021 AIAA SciTech Forum.* AIAA, USA, Article 0566, 12 pages. https://doi.org/10.2514/6.2021-0566

[107] James B. Dabney, Julia M. Badger, and Pavan Rajagopal. 2023. Trustworthy Autonomy for Gateway Vehicle System Manager. In *Proceedings of the 14th IEEE Space Computing Conference (SCC'23).* IEEE, USA, 57–62. https://doi.org/10.1109/SCC57168.2023.00018

[108] Clara DaSilva, Babak Dehbonei, and Fernando Mejia. 1992. Formal specification in the development of industrial applications: Subway speed control system. In *Proceedings of the IFIP TC6/WG6.1 5th International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'92) (IFIP Transactions, Vol. C-10)*, Michel Diaz and Roland Groz (Eds.). North-Holland, The Netherlands, 199–213.

[109] Alexandre David, Peter G. Jensen, Kim G. Larsen, Marius Mikucionis, and Jakob H. Taankvist. 2015. Uppaal Stratego. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15) (LNCS, Vol. 9035)*, Christel Baier and Cesare Tinelli (Eds.). Springer, Germany, 206–211. https://doi.org/10.1007/978-3-662-46681-0_16

[110] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny B. Poulsen. 2015. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* 17, 4 (2015), 397–415. https://doi.org/10.1007/S10009-014-0361-Y

[111] Julien Delange. 2017. *AADL In Practice: Design and Validate the Architecture of Critical Systems.* Reblochon, France.

[112] David Delmas and Jean Souyris. 2007. Astrée: From Research to Industry. In *Proceedings of the 14th International Symposium on Static Analysis (SAS'07) (LNCS, Vol. 4634)*, Hanne Riis Nielson and Gilberto Filé (Eds.). Springer, Germany, 437–451. https://doi.org/10.1007/978-3-540-74061-2_27

[113] Edsger W. Dijkstra. 1968. A constructive approach to the problem of program correctness. *BIT Numer. Math.* 8, 3 (1968), 174–186. https://doi.org/10.1007/BF01933419

[114] Edsger W. Dijkstra. 1976. *A Discipline of Programming.* Prentice-Hall, USA.

[115] David L. Dill. 1996. The Mur*phi* Verification System. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96) (LNCS, Vol. 1102)*, Rajeev Alur and Thomas A. Henzinger (Eds.). Springer, Germany, 390–393. https://doi.org/10.1007/3-540-61474-5_86

[116] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling Static Analyses at Facebook. *Commun. ACM* 62, 8 (2019), 62–70. https://doi.org/10.1145/3338112

[117] Daniel Dollé, Didier Essamé, and Jérôme Falampin. 2003. B dans le transport ferroviaire: L'expérience de Siemens Transportation Systems. *Tech. Sci. Inform.* 22, 1 (2003), 11–32. https://doi.org/10.3166/tsi.22.11-32

[118] Brijesh Dongol, Catherine Dubois, Stefan Hallerstede, Eric Hehner, Carroll Morgan, Peter Müller, Leila Ribeiro, Alexandra Silva, Graeme Smith, and Erik de Vink. 2024. On Formal Methods Thinking in Computer Science Education. *Form. Asp. Comput.* (2024). https://doi.org/10.1145/36704

[119] Parasara Sridhar Duggirala, Le Wang, Sayan Mitra, Mahesh Viswanathan, and César A. Muñoz. 2014. Temporal Precedence Checking for Switched Models and Its Application to a Parallel Landing Protocol. In *Proceedings of the 19th International Symposium on Formal Methods (FM'14) (LNCS, Vol. 8442)*, Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun (Eds.). Springer, Germany, 215–229. https://doi.org/10.1007/978-3-319-06410-9_16

[120] Bruno Dutertre. 2011. *Probabilistic Analysis of Distributed Fault-Tolerant Systems.* Technical Report NASA/CR–2011-217090. NASA. https://ntrs.nasa.gov/citations/20110011564

[121] Didier Essamé and Daniel Dollé. 2007. B in Large Scale Projects: The Canarsie Line CBTC Experience. In *Proceedings of the 7th International Conference of B Users (B'07) (LNCS, Vol. 4355)*, Jacques Julliand and Olga Kouchnarenko (Eds.). Springer, Germany, 252–254. https://doi.org/10.1007/11955757_21

[122] European Committee for Electrotechnical Standardization. 2011. CENELEC EN 50128: Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems. https://standards.globalspec.com/std/1678027/cenelec-en-50128

[123] Marie Farrell, Matt Luckcuck, Oisín Sheridan, and Rosemary Monahan. 2022. FRETting About Requirements: Formalised Requirements for an Aircraft Engine Controller. In *Proceedings of the 28th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'22) (LNCS, Vol. 13216)*, Vincenzo Gervasi and Andreas Vogelsang (Eds.). Springer, Germany, 96–111. https://doi.org/10.1007/978-3-030-98464-9_9

[124] Andrew Ferraiuolo, Rui Xu, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. 2017. Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis. In *Proceedings of the 22nd International Conference on Architectural Support for*

*Programming Languages and Operating Systems (ASPLOS'17)*, Yunji Chen, Olivier Temam, and John Carter (Eds.). ACM, USA, 555–568. https://doi.org/10.1145/3037697.3037739

[125] Alessio Ferrari, Alessandro Fantechi, Gianluca Magnani, Daniele Grasso, and Matteo Tempestini. 2013. The Metrô Rio case study. *Sci. Comput. Program.* 78, 7 (2013), 828–842. https://doi.org/10.1016/j.scico.2012.04.003

[126] Alessio Ferrari, Franco Mazzanti, Davide Basile, and Maurice H. ter Beek. 2022. Systematic Evaluation and Usability Analysis of Formal Methods Tools for Railway Signaling System Design. *IEEE Trans. Softw. Eng.* 48, 11 (2022), 4675–4691. https://doi.org/10.1109/TSE.2021.3124677

[127] Alessio Ferrari, Franco Mazzanti, Davide Basile, Maurice H. ter Beek, and Alessandro Fantechi. 2020. Comparing Formal Tools for System Design: a Judgment Study. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE'20)*. ACM, USA, 62–74. https://doi.org/10.1145/3377811.3380373

[128] Alessio Ferrari and Maurice H. ter Beek. 2023. Formal Methods in Railways: a Systematic Mapping Study. *ACM Comput. Surv.* 55, 4 (2023), 69:1–69:37. https://doi.org/10.1145/3520480

[129] Felype Ferreira, Laís Neves, Michelle Silva, and Paulo Borba. 2010. TaRGeT: a Model Based Product Line Testing Tool. Tools Session of the 1st Brazilian Conference on Software: Theory and Practice (CBSoft'10). https://twiki.cin.ufpe.br/twiki/pub/SPG/SoftwareEstimationModels/TargetCBSOFT.pdf

[130] Larissa Ferreira, Sidney C. Nogueira, Lucas Lima, Liliane Fonseca, and Waldemar Ferreira. 2019. Initial findings on the evaluation of a model-based testing tool in the test design process. In *Proceedings of the 13th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'19)*. IEEE, USA, 1–6. https://doi.org/10.1109/ESEM.2019.8870140

[131] Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz, and Henrik Lönn. 2016. Simulink to UPPAAL Statistical Model Checker: Analyzing Automotive Industrial Systems. In *Proceedings of the 21st International Symposium on Formal Methods (FM'16) (LNCS, Vol. 9995)*, John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou (Eds.). Springer, Germany, 748–756. https://doi.org/10.1007/978-3-319-48989-6_46

[132] Michael Fisher, Viviana Mascardi, Kristin Y. Rozier, Bernd-Holger Schlingloff, Michael Winikoff, and Neil Yorke-Smith. 2021. Towards a framework for certification of reliable autonomous systems. *Auton. Agents Multi Agent Syst.* 35, 1 (2021), 8:1–8:65. https://doi.org/10.1007/s10458-020-09487-2

[133] Robert W. Floyd. 1967. Assigning Meanings to Programs. In *Proceedings of Symposia in Applied Mathematics (Mathematical Aspects of Computer Science, Vol. 19)*, J. T. Schwartz (Ed.). American Mathematical Society, USA, 19–32.

[134] Song Gao, Bohua Zhan, Depeng Liu, Xuechao Sun, Yanan Zhi, David N. Jansen, and Lijun Zhang. 2021. Formal Verification of Consensus in the Taurus Distributed Database. In *Proceedings of the 24th International Symposium on Formal Methods (FM'21) (LNCS, Vol. 13047)*, Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan (Eds.). Springer, Germany, 741–751. https://doi.org/10.1007/978-3-030-90870-6_42

[135] Hubert Garavel and Susanne Graf. 2013. *Formal Methods for Safe and Secure Computer Systems.* BSI Study 875. Bundesamt für Sicherheit in der Informationstechnik. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/formal_methods_study_875/formal_methods_study_875.html

[136] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. 2013. CADP 2011: a toolbox for the construction and analysis of distributed processes. *Int. J. Softw. Tools Technol. Transf.* 15, 2 (2013), 89–107. https://doi.org/10.1007/s10009-012-0244-z

[137] Hubert Garavel, Frédéric Lang, and Wendelin Serwe. 2017. From LOTOS to LNT. In *ModelEd, TestEd, TrustEd (LNCS, Vol. 10500)*, Joost-Pieter Katoen, Rom Langerak, and Arend Rensink (Eds.). Springer, Germany, 3–26. https://doi.org/10.1007/978-3-319-68270-9_1

[138] Hubert Garavel, Maurice H. ter Beek, and Jaco van de Pol. 2020. The 2020 Expert Survey on Formal Methods. In *Proceedings of the 25th International Conference on Formal Methods for Industrial Critical Systems (FMICS'20) (LNCS, Vol. 12327)*, Maurice H. ter Beek and Dejan Ničković (Eds.). Springer, Germany, 3–69. https://doi.org/10.1007/978-3-030-58298-2_1

[139] Marco Gario, Alessandro Cimatti, Cristian Mattarei, Stefano Tonetta, and Kristin Y. Rozier. 2016. Model Checking at Scale: Automated Air Traffic Control Design Space Exploration. In *Proceedings of the 28th International Conference on Computer Aided Verification (CAV'16) (LNCS, Vol. 9780)*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer, Germany, 3–22. https://doi.org/10.1007/978-3-319-41540-6_1

[140] Stephen J. Garland and John V. Guttag. 1988. LP: The Larch Prover. In *Proceedings of the 9th International Conference on Automated Deduction (CADE'88) (LNCS, Vol. 310)*, Ewing L. Lusk and Ross A. Overbeek (Eds.). Springer, Germany, 748–749. https://doi.org/10.1007/BFB0012879

[141] Dimitra Giannakopoulou, Falk Howar, Malte Isberner, Todd Lauderdale, Zvonimir Rakamaric, and Vishwanath Raman. 2014. Taming Test Inputs for Separation Assurance. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14)*. ACM, USA, 373–384. https://doi.org/10.1145/2642937.2642940

[142] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Julian Rhein, Johann Schumann, and Nija Shi. 2020. Formal Requirements Elicitation with FRET. In *Joint Proceedings of the Co-Located Events of the 26th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-JP'20) (CEUR Proceedings, Vol. 2584)*, Mehrdad Sabetzadeh, Andreas Vogelsang, Sallam Abualhaija, Markus Borg, Fabiano Dalpiaz, Maya Daneva, Nelly Condori-Fernández, Xavier Franch, Davide Fucci, Vincenzo Gervasi, Eduard C. Groen, Renata S. S. Guizzardi, Andrea Herrmann, Jennifer Horkoff, Luisa Mich, Anna Perini, and Angelo Susi (Eds.).

CEUR-WS.org, Germany, 6 pages. https://ceur-ws.org/Vol-2584/PT-paper4.pdf
[143] Thomas Gibson-Robinson, Philip J. Armstrong, Alexandre Boulgakov, and A. W. Roscoe. 2014. FDR3 – A Modern Refinement Checker for CSP. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14) (LNCS, Vol. 8413)*, Erika Ábrahám and Klaus Havelund (Eds.). Springer, Germany, 187–201. https://doi.org/10.1007/978-3-642-54862-8_13
[144] Mario Gleirscher and Diego Marmsoler. 2020. Formal Methods in Dependable Systems Engineering: A Survey of Professionals from Europe and North America. *Empir. Softw. Eng.* 25, 6 (2020), 4473–4546. https://doi.org/10.1007/s10664-020-09836-5
[145] Stefania Gnesi and Tiziana Margaria (Eds.). 2013. *Formal Methods for Industrial Critical Systems: A Survey of Applications.* Wiley, UK. https://doi.org/10.1002/9781118459898
[146] Patrice Godefroid. 2020. Fuzzing: Hack, Art, and Science. *Commun. ACM* 63, 2 (2020), 70–76. https://doi.org/10.1145/3363824
[147] Patrice Godefroid, Jonathan de Halleux, Aditya V. Nori, Sriram K. Rajamani, Wolfram Schulte, Nikolai Tillmann, and Michael Y. Levin. 2008. Automating Software Testing Using Program Analysis. *IEEE Softw.* 25, 5 (2008), 30–37. https://doi.org/10.1109/MS.2008.109
[148] Patrice Godefroid, Shuvendu K. Lahiri, and Cindy Rubio-González. 2011. Statically Validating Must Summaries for Incremental Compositional Dynamic Test Generation. In *Proceedings of the 18th International Symposium on Static Analysis (SAS'11) (LNCS, Vol. 6887)*, Eran Yahav (Ed.). Springer, Germany, 112–128. https://doi.org/10.1007/978-3-642-23702-7_12
[149] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. 2008. Automated Whitebox Fuzz Testing. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS'08)*. The Internet Society, USA, 16 pages. https://www.ndss-symposium.org/ndss2008/automated-whitebox-fuzz-testing/
[150] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. 2012. SAGE: Whitebox Fuzzing for Security Testing. *Commun. ACM* 55, 3 (2012), 40–44. https://doi.org/10.1145/2093548.2093564
[151] Adriano Gomes, Alexandre Mota, Augusto Sampaio, Felipe Ferri, and Edson Watanabe. 2012. Constructive model-based analysis for safety assessment. *Int. J. Softw. Tools Technol. Transf.* 14, 6 (2012), 673–702. https://doi.org/10.1007/S10009-012-0238-X
[152] Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. 1979. *Edinburgh LCF: A Mechanised Logic of Computation.* LNCS, Vol. 78. Springer, Germany. https://doi.org/10.1007/3-540-09724-4
[153] Stijn de Gouw, Frank S. de Boer, Richard Bubel, Reiner Hähnle, Jurriaan Rot, and Dominic Steinhöfel. 2019. Verifying OpenJDK's Sort Method for Generic Collections. *J. Autom. Reason.* 62, 1 (2019), 93–126. https://doi.org/10.1007/S10817-017-9426-4
[154] Stijn de Gouw, Jurriaan Rot, Frank S. de Boer, Richard Bubel, and Reiner Hähnle. 2015. OpenJDK's Java.utils.Collection.sort() Is Broken: The Good, the Bad and the Worst Case. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV'15) (LNCS, Vol. 9206)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, Germany, 273–289. https://doi.org/10.1007/978-3-319-21690-4_16
[155] David Greve, Matthew Wilding, and W. Mark Vanfleet. 2005. High Assurance Formal Security Policy Modeling. In *Proceedings of the 17th Systems and Software Technology Conference (SSTC'05)*. IEEE, USA.
[156] Alex Groce, Klaus Havelund, Gerard J. Holzmann, Rajeev Joshi, and Ru-Gang Xu. 2014. Establishing flight software reliability: Testing, model checking, constraint-solving, monitoring and learning. *Ann. Math. Artif. Intell.* 70, 4 (2014), 315–349. https://doi.org/10.1007/s10472-014-9408-8
[157] Jan Friso Groote and Mohammad Reza Mousavi. 2014. *Modeling and Analysis of Communicating Systems.* MIT Press, USA. https://mitpress.mit.edu/9780262547871/modeling-and-analysis-of-communicating-systems/
[158] Rong Gu, Raluca Marinescu, Cristina Seceleanu, and Kristina Lundqvist. 2019. Towards a Two-Layer Framework for Verifying Autonomous Vehicles. In *Proceedings of the 11th International NASA Formal Methods Symposium (NFM'19) (LNCS, Vol. 11460)*, Julia M. Badger and Kristin Yvonne Rozier (Eds.). Springer, Germany, 186–203. https://doi.org/10.1007/978-3-030-20652-9_12
[159] Gérard Guiho and Claude Hennebert. 1990. SACEM Software Validation. In *Proceedings of the 12th International Conference on Software Engineering (ICSE'90)*. IEEE, USA, 186–191.
[160] George Hagen, Ricky Butler, and Jeffrey Maddalon. 2011. Stratway: A modular approach to strategic conflict resolution. In *Proceedings of the 11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*. AIAA, USA, Article 6892, 13 pages. https://doi.org/10.2514/6.2011-6892
[161] Anthony Hall. 1990. Seven Myths of Formal Methods. *IEEE Softw.* 7, 5 (1990), 11–19. https://doi.org/10.1109/52.57887
[162] Anthony Hall. 2007. Realising the Benefits of Formal Methods. *J. Univers. Comput. Sci.* 13, 5 (2007), 669–678. https://doi.org/10.3217/jucs-013-05-0669
[163] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. 2004. *Guide to Elliptic Curve Cryptography.* Springer, Germany. https://doi.org/10.1007/b97644
[164] Dominik Hansen, Michael Leuschel, Philipp Körner, Sebastian Krings, Thomas Naulin, Nader Nayeri, David Schneider, and Frank Skowron. 2020. Validation and real-life demonstration of ETCS hybrid level 3 principles using a formal B model. *Int. J. Softw. Tools Technol. Transf.* 22, 3 (2020), 315–332. https://doi.org/10.1007/s10009-020-00551-6
[165] David S. Hardin, Eric W. Smith, and William D. Young. 2006. A Robust Machine Code Proof Framework for Highly Secure Applications. In *Proceedings of the 6th International Workshop on the ACL2 Prover and its Applications (ACL2'06)*. ACM, USA, 11–20. https://doi.org/10.1145/1217975.1217978

[166] David Harel. 1987. Statecharts: a visual formalism for complex systems. *Sci. Comput. Program.* 8, 3 (1987), 231–274. https://doi.org/10.1016/0167-6423(87)90035-9
[167] John Harrison. 2023. s2n-bignum GitHub repository. https://github.com/awslabs/s2n-bignum
[168] Hans-Dieter A. Hiep, Olaf Maathuis, Jinting Bian, Frank S. de Boer, and Stijn de Gouw. 2022. Verifying OpenJDK's `LinkedList` using KeY. *Int. J. Softw. Tools Technol. Transf.* 24, 5 (2022), 783–802. https://doi.org/10.1007/s10009-022-00679-7
[169] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul J. Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy A. Vilkomir, Martin R. Woodward, and Hussein Zedan. 2009. Using formal specifications to support testing. *ACM Comput. Surv.* 41, 2 (2009), 9:1–9:76. https://doi.org/10.1145/1459352.1459354
[170] C. A. R. (Tony) Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (1969), 576–580. https://doi.org/10.1145/363235.363259
[171] C. A. R. (Tony) Hoare. 1985. *Communicating Sequential Processes.* Prentice Hall, USA.
[172] Gerard J. Holzmann. 2003. *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley, USA.
[173] Gerard J. Holzmann. 2014. Mars Code. *Commun. ACM* 57I, 2 (2014), 64–73. https://doi.org/10.1145/2560217.2560218
[174] Marieke Huisman, Dilian Gurov, and Alexander Malkis. 2020. Formal Methods: From Academia to Industrial Practice – A Travel Guide. arXiv:2002.07279
[175] Marieke Huisman and Anton Wijs. 2023. *Concise Guide to Software Verification: From Model Checking to Annotation Checking.* Springer, Germany. https://doi.org/10.1007/978-3-031-30167-4
[176] Intel. 2011. Intel Identifies Chipset Design Error, Implementing Solution. Press release. https://intc.com/news-events/press-releases/detail/688/intel-identifies-chipset-design-error-implementing-solution
[177] International Electrotechnical Commission. 2023. IEC TC 107: Process management for avionics. https://www.iec.ch/dyn/www/f?p=103:7:0::::FSP_ORG_ID:1304
[178] International Electrotechnical Commission. 2023. IEC TC 97: Electrical installations for lighting and beaconing of aerodromes. https://www.iec.ch/dyn/www/f?p=103:7:0::::FSP_ORG_ID:1294
[179] International Organization for Standardization. 2018. ISO 26262: Road vehicles — Functional safety — Parts 1–12. https://www.iso.org/standard/68383.html
[180] International Organization for Standardization and International Electrotechnical Commission. 2017. ISO/IEC 19514 - Information technology – Object management group systems modeling language (OMG SysML). https://www.iso.org/standard/65231.html
[181] Raj Jain. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley, UK. https://www.wiley.com/en-us/The+Art+of+Computer+Systems+Performance+Analysis%3A+Techniques+for+Experimental+Design%2C+Measurement%2C+Simulation%2C+and+Modeling-p-9780471503361
[182] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan W. Gardner, Aurora C. Schmidt, Erik Zawadzki, and André Platzer. 2015. A Formally Verified Hybrid System for the Next-Generation Airborne Collision Avoidance System. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15) (LNCS, Vol. 9035)*, Christel Baier and Cesare Tinelli (Eds.). Springer, Germany, 21–36. https://doi.org/10.1007/978-3-662-46681-0_2
[183] Hanru Jiang, Hongjin Liang, Siyang Xiao, Junpeng Zha, and Xinyu Feng. 2019. Towards Certified Separate Compilation for Concurrent Programs. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'19).* ACM, USA, 111–125. https://doi.org/10.1145/3314221.3314595
[184] He Jifeng. 1994. From CSP to Hybrid Systems. In *A Classical Mind: Essays in Honour of C. A. R. Hoare*, A. W. (Bill) Roscoe (Ed.). Prentice Hall, UK, 171–189.
[185] Chris Johannsen, Phillip H. Jones, Brian Kempa, Kristin Y. Rozier, and Pei Zhang. 2023. R2U2 Version 3.0: Re-Imagining a Toolchain for Specification, Resource Estimation, and Optimized Observer Generation for Runtime Verification in Hardware and Software. In *Proceedings of the 35th International Conference on Computer Aided Verification (CAV'23) (LNCS, Vol. 13966)*, Constantin Enea and Akash Lal (Eds.). Springer, Germany, 483–497. https://doi.org/10.1007/978-3-031-37709-9_23
[186] Chris Johannsen, Brian Kempa, Phillip H. Jones, Kristin Y. Rozier, and Tichakorn Wongpiromsarn. 2023. Impossible Made Possible: Encoding Intractable Specifications via Implied Domain Constraints. In *Proceedings of the 28th International Conference on Formal Methods for Industrial Critical Systems (FMICS'23) (LNCS, Vol. 14290)*, Alessandro Cimatti and Laura Titolo (Eds.). Springer, Germany, 151–169. https://doi.org/10.1007/978-3-031-43681-9_9
[187] Stephen C. Johnson. 1977. *Lint, a C program checker.* Technical Report 65. Bell Labs.
[188] Clifford B. Jones. 1991. *Systematic software development using VDM* (2 ed.). Prentice Hall, USA.
[189] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. 2021. Safe Systems Programming in Rust. *Commun. ACM* 64, 4 (2021), 144–152. https://doi.org/10.1145/3418295
[190] Matt Kaufmann, Panagiotis Manolios, and J. Strother Moore. 2000. *Computer-Aided Reasoning: An Approach.* Advances in Formal Methods, Vol. 3. Springer, Germany. https://doi.org/10.1007/978-1-4615-4449-4
[191] Matt Kaufmann, Panagiotis Manolios, and J. Strother Moore (Eds.). 2000. *Computer-Aided Reasoning: ACL2 Case Studies.* Advances in Formal Methods, Vol. 4. Springer, Germany. https://doi.org/10.1007/978-1-4757-3188-0

[192] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David A. Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2010. seL4: Formal Verification of an Operating-System Kernel. *Commun. ACM* 53, 6 (2010), 107–115. https://doi.org/10.1145/1743546.1743574

[193] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David A. Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2009. seL4: Formal Verification of an OS Kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP'09)*, Jeanna Neefe Matthews and Thomas E. Anderson (Eds.). ACM, USA, 207–220. https://doi.org/10.1145/1629575.1629596

[194] Nikolai Kosmatov and Julien Signoles. 2016. Frama-C, A Collaborative Framework for C Code Verification: Tutorial Synopsis. In *Proceedings of the 16th International Conference on Runtime Verification (RV'16) (LNCS, Vol. 10012)*, Yliès Falcone and César Sánchez (Eds.). Springer, Germany, 92–115. https://doi.org/10.1007/978-3-319-46982-9_7

[195] Jörg Kreiker, Andrzej Tarlecki, Moshe Y. Vardi, and Reinhard Wilhelm. 2011. Modeling, Analysis, and Verification – The Formal Methods Manifesto 2010. *Dagstuhl Manifestos* 1, 1 (2011), 21–40. https://doi.org/10.4230/DAGMAN.1.1.21

[196] Abderahman Kriouile and Wendelin Serwe. 2015. Using a Formal Model to Improve Verification of a Cache-Coherent System-on-Chip. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15) (LNCS, Vol. 9035)*, Christel Baier and Cesare Tinelli (Eds.). Springer, Germany, 708–722. https://doi.org/10.1007/978-3-662-46681-0_62

[197] Amruth N. Kumar, Rajendra K. Raj, Sherif G. Aly, Monica D. Anderson, Brett A. Becker, Richard L. Blumenthal, Eric Eaton, Susan L. Epstein, Michael Goldweber, Pankaj Jalote, Douglas Lea, Michael Oudshoorn, Marcelo Pias, Susan Reiser, Christian Servin, Rahul Simha, Titus Winters, and Qiao Xiang. 2024. *Computer Science Curricula 2023*. ACM, IEEE, and AAAI, USA. https://doi.org/10.1145/3664191

[198] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2007. Stochastic Model Checking. In *Formal Methods for Performance Evaluation: Advanced Lectures of the 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'07) (LNCS, Vol. 4486)*, Marco Bernardo and Jane Hillston (Eds.). Springer, Germany, 220–270. https://doi.org/10.1007/978-3-540-72522-0_6

[199] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11) (LNCS, Vol. 6806)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, Germany, 585–591. https://doi.org/10.1007/978-3-642-22110-1_47

[200] Marta Z. Kwiatkowska and David Parker. 2012. Advances in Probabilistic Model Checking. In *Software Safety and Security: Tools for Analysis and Verification*, Tobias Nipkow, Orna Grumberg, and Benedikt Hauptmann (Eds.). NATO Science for Peace and Security Series, Vol. 33. IOS Press, The Netherlands, 126–151. https://doi.org/10.3233/978-1-61499-028-4-126

[201] Leslie Lamport. 1980. "Sometime" is Sometimes "Not Never": On the Temporal Logic of Programs. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages (POPL'80)*. ACM, USA, 174–185. https://doi.org/10.1145/567446.567463

[202] Etienne Lantreibecq and Wendelin Serwe. 2014. Formal analysis of a hardware dynamic task dispatcher with CADP. *Sci. Comput. Program.* 80 (2014), 130–149. https://doi.org/10.1016/j.scico.2013.01.003

[203] Brian R. Larson, Patrice Chalin, and John Hatcliff. 2013. BLESS: Formal Specification and Verification of Behaviors for Embedded Systems with Software. In *Proceedings of the 5th International NASA Formal Methods Symposium (NFM'13) (LNCS, Vol. 7871)*, Guillaume Brat, Neha Rungta, and Arnaud Venet (Eds.). Springer, Germany, 276–290. https://doi.org/10.1007/978-3-642-38088-4_19

[204] Julia Lawall and Gilles Muller. 2018. Coccinelle: 10 Years of Automated Evolution in the Linux Kernel. In *Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC'18)*. USENIX Association, USA, 601–613. https://www.usenix.org/conference/atc18/presentation/lawall

[205] Julia Lawall and Gilles Muller. 2022. Automating Program Transformation with Coccinelle. In *Proceedings of the 14th International NASA Formal Methods Symposium (NFM'22) (LNCS, Vol. 13260)*, Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez (Eds.). Springer, Germany, 71–87. https://doi.org/10.1007/978-3-031-06773-0_4

[206] Thierry Lecomte, Lilian Burdy, and Michael Leuschel. 2020. Formally Checking Large Data Sets in the Railways. arXiv:1210.6815 Proceedings of the Workshop on the experience of and advances in developing dependable systems in Event-B (DS-Event-B'12).

[207] Thierry Lecomte, David Déharbe, Étienne Prun, and Erwan Mottin. 2017. Applying a Formal Method in Industry: A 25-Year Trajectory. In *Proceedings of the 20th Brazilian Symposium on Formal Methods: Foundations and Applications (SBMF'17) (LNCS, Vol. 10623)*, Simone Cavalheiro and José Fiadeiro (Eds.). Springer, Germany, 70–87. https://doi.org/10.1007/978-3-319-70848-5_6

[208] Axel Legay, Anna Lukina, Louis-Marie Traonouez, Junxing Yang, Scott A. Smolka, and Radu Grosu. 2019. Statistical Model Checking. In *Computing and Software Science: State of the Art and Perspectives*, Bernhard Steffen and Gerhard J. Woeginger (Eds.). LNCS, Vol. 10000. Springer, Germany, 478–504. https://doi.org/10.1007/978-3-319-91908-9_23

[209] K. Rustan M. Leino. 2023. *Program Proofs*. MIT Press, USA. https://mitpress.mit.edu/9780262546232/program-proofs/

[210] Valentina Lenarduzzi, Fabiano Pecorelli, Nyyti Saarimäki, Savanna Lujan, and Fabio Palomba. 2023. A critical comparison on six static analysis tools: Detection, agreement, and precision. *J. Syst. Softw.* 198 (2023), 111575. https://doi.org/10.1016/J.JSS.2022.111575

[211] Michael Leuschel and Michael J. Butler. 2008. ProB: an automated analysis toolset for the B method. *Int. J. Softw. Tools Technol. Transf.* 10, 2 (2008), 185–203. https://doi.org/10.1007/s10009-007-0063-9

[212] Michael Leuschel, Jérôme Falampin, Fabian Fritz, and Daniel Plagge. 2011. Automated property verification for large scale B models with ProB. *Form. Asp. Comput.* 23, 6 (2011), 683–709. https://doi.org/10.1007/s00165-010-0172-1
[213] Nancy G. Leveson. 2017. The Therac-25: 30 Years Later. *IEEE Comput.* 50, 11 (2017), 8–11. https://doi.org/10.1109/MC.2017.4041349
[214] Nancy G. Leveson and Clark S. Turner. 1993. An Investigation of the Therac-25 Accidents. *IEEE Comput.* 26, 7 (1993), 18–41. https://doi.org/10.1109/MC.1993.274940
[215] Henrik Lönn. 2019. Model Based Continuous Integration of Automotive Embedded Systems. In *Proceedings of the 13th MODPROD Workshop 2019: Cyber-Physical Product Development (Linköping Electronic Press Workshop and Conference Collection, Vol. 21)*. Linköping University, Sweden, 36 pages. https://wcc.ep.liu.se/index.php/MODPROD/article/view/116
[216] Nesredin Mahmud, Cristina Seceleanu, and Oscar Ljungkrantz. 2015. ReSA: An ontology-based requirement specification language tailored to automotive systems. In *Proceedings of the 10th International Symposium on Industrial Embedded Systems (SIES'15)*. IEEE, USA, 1–10. https://doi.org/10.1109/SIES.2015.7185035
[217] Nesredin Mahmud, Cristina Seceleanu, and Oscar Ljungkrantz. 2016. ReSA Tool: Structured Requirements Specification and SAT-based Consistency-checking. In *Proceedings of the 18th Federated Conference on Computer Science and Information Systems (FedCSIS'16) (ACSIS, Vol. 8)*, Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki (Eds.). IEEE, USA, 1737–1746. https://doi.org/10.15439/2016F404
[218] Raluca Marinescu, Henrik Kaijser, Marius Mikucionis, Cristina Seceleanu, Henrik Lönn, and Alexandre David. 2014. Analyzing Industrial Architectural Models by Simulation and Model-Checking. In *Revised Selected Papers of the 3rd International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'14) (CCIS, Vol. 476)*, Cyrille Artho and Peter Csaba Ölveczky (Eds.). Springer, Germany, 189–205. https://doi.org/10.1007/978-3-319-17581-2_13
[219] Nadja Marko, Eike Möhlmann, Dejan Ničković, Jürgen Niehaus, Peter Priller, and Martijn Rooker. 2021. Challenges of engineering safe and secure highly automated vehicles: Whitepaper. arXiv:2103.03544
[220] Lina Marsso, Radu Mateescu, and Wendelin Serwe. 2018. TESTOR: A Modular Tool for On-the-Fly Conformance Test Case Generation. In *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'18) (LNCS, Vol. 10806)*, Dirk Beyer and Marieke Huisman (Eds.). Springer, Germany, 211–228. https://doi.org/10.1007/978-3-319-89963-3_13
[221] Radu Mateescu, Wendelin Serwe, Aymane Bouzafour, and Marc Renaudin. 2020. Modeling an Asynchronous Circuit Dedicated to the Protection Against Physical Attacks. In *Proceedings of the 4th Workshop on Models for Formal Analysis of Real Systems (MARS'20) (EPTCS, Vol. 316)*, Ansgar Fehnker and Hubert Garavel (Eds.). Open Publishing Association, Australia, 200–239. https://doi.org/10.4204/EPTCS.316.8
[222] Cristian Mattarei, Alessandro Cimatti, Marco Gario, Stefano Tonetta, and Kristin Y. Rozier. 2015. Comparing Different Functional Allocations in Automated Air Traffic Control Design. In *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design (FMCAD'15)*, Roope Kaivola and Thomas Wahl (Eds.). IEEE, USA, 112–119. https://doi.org/10.1109/FMCAD.2015.7542260
[223] Franco Mazzanti, Alessio Ferrari, and Giorgio O. Spagnolo. 2018. Towards formal methods diversity in railways: an experience report with seven frameworks. *Int. J. Softw. Tools Technol. Transf.* 20, 3 (2018), 263–288. https://doi.org/10.1007/s10009-018-0488-3
[224] Peter C. Mehlitz. 2008. Trust Your Model – Verifying Aerospace System Models with Java™ Pathfinder. In *Proceedings of the 29th IEEE Aerospace Conference (AERO'08)*. IEEE, USA, 1–11. https://doi.org/10.1109/AERO.2008.4526573
[225] John M. Mellor-Crummey and Michael L. Scott. 1991. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. *ACM Trans. Comput. Syst.* 9, 1 (1991), 21–65. https://doi.org/10.1145/103727.103729
[226] Bertrand Meyer. 2008. Seven Principles of Software Testing. *IEEE Comp.* 41, 8 (2008), 99–101. https://doi.org/10.1109/MC.2008.306
[227] Steven P. Miller. 2012. Lessons from Twenty Years of Industrial Formal Methods. In *Proceedings of the 20th High Confidence Software and Systems Conference (HCSS'12)*. Cyber-Physical Systems Virtual Organization, USA, 25 pages. http://cps-vo.org/node/3434
[228] Robin Milner. 1980. *A Calculus of Communicating Systems*. LNCS, Vol. 92. Springer, Germany. https://doi.org/10.1007/3-540-10235-3
[229] Gordon E. Moore. 1965. Cramming more components onto integrated circuits. *Electronics* 38, 8 (1965), 114–117. Reprinted in *Proc. IEEE* 86, 1 (1998), 82–85. https://doi.org/10.1109/jproc.1998.658762.
[230] Carroll Morgan. 1990. *Programming from Specifications*. Prentice-Hall, USA.
[231] Joseph M. Morris. 1987. A Theoretical Basis for Stepwise Refinement and the Programming Calculus. *Sci. Comput. Program.* 9, 3 (1987), 287–306. https://doi.org/10.1016/0167-6423(87)90011-6
[232] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08) (LNCS, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, Germany, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24
[233] Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *Proceedings of the 28th International Conference on Automated Deduction (CADE'21) (LNCS, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, Germany, 625–635. https://doi.org/10.1007/978-3-030-79876-5_37
[234] Yannick Moy, Emmanuel Ledinot, Hervé Delseny, Virginie Wiels, and Benjamin Monate. 2013. Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. *IEEE Softw.* 30, 3 (2013), 50–57. https://doi.org/10.1109/MS.2013.43

[235] Cesar Munoz, Radu Siminiceanu, Victor A. Carreno, and Gilles Dowek. 2005. *KB3D Reference Manual.* Technical Report NASA/TM-2005-213769. NASA. https://ntrs.nasa.gov/citations/20050186553
[236] César A. Muñoz, Victor Carreño, and Gilles Dowek. 2006. Formal Analysis of the Operational Concept for the Small Aircraft Transportation System. In *Rigorous Development of Complex Fault-Tolerant Systems (LNCS, Vol. 4157)*, Michael J. Butler, Cliff B. Jones, Alexander B. Romanovsky, and Elena Troubitsyna (Eds.). Springer, Germany, 306–325. https://doi.org/10.1007/11916246_16
[237] Anthony Narkawicz, César A. Muñoz, and Gilles Dowek. 2012. Provably correct conflict prevention bands algorithms. *Sci. Comput. Program.* 77, 10-11 (2012), 1039–1057. https://doi.org/10.1016/j.scico.2011.07.002
[238] Monty Newborn. 2001. *Automated Theorem Proving.* Springer, Germany. https://doi.org/10.1007/978-1-4613-0089-2
[239] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon Web Services Uses Formal Methods. *Commun. ACM* 58, 4 (2015), 66–73. https://doi.org/10.1145/2699417
[240] Flemming Nielson and Hanne Riis Nielson. 2019. *Formal Methods: An Appetizer.* Springer, Germany. https://doi.org/10.1007/978-3-030-05156-3
[241] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel (Eds.). 2002. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* LNCS, Vol. 2283. Springer, Germany. https://doi.org/10.1007/3-540-45949-9
[242] Sidney Nogueira, Emanuela Cartaxo, Dante Torres, Eduardo Aranha, and Rafael Marques. 2011. Model Based Test Generation: An Industrial Experience. In *Proceedings of the 1st Brazilian Workshop on Systematic and Automated Software Testing (SAST'07).* SBC, Brazil, 6 pages.
[243] Sidney Nogueira, Augusto Sampaio, and Alexandre Mota. 2014. Test generation from state based use case models. *Form. Asp. Comput.* 26 (2014), 441–490. https://doi.org/10.1007/s00165-012-0258-z
[244] Jonas Oberhauser, Rafael Lourenco de Lima Chehab, Diogo Behrens, Ming Fu, Antonio Paolillo, Lilith Oberhauser, Koustubha Bhat, Yuzhong Wen, Haibo Chen, Jaeho Kim, and Viktor Vafeiadis. 2021. VSync: Push-Button Verification and Optimization for Synchronization Primitives on Weak Memory Models. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21).* ACM, USA, 530–545. https://doi.org/10.1145/3445814.3446748
[245] Peter W. O'Hearn. 2020. Incorrectness Logic. *Proc. ACM Program. Lang.* 4, POPL (2020), 10:1–10:32. https://doi.org/10.1145/3371078
[246] Gerard O'Regan. 2017. *Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications.* Springer, Germany. https://doi.org/10.1007/978-3-319-64021-1
[247] Ammar Osaiweran, Mathijs Schuts, Jozef Hooman, Jan Friso Groote, and Bart J. van Rijnsoever. 2016. Evaluating the effect of a lightweight formal technique in industry. *Int. J. Softw. Tools Technol. Transf.* 18, 1 (2016), 93–108. https://doi.org/10.1007/S10009-015-0374-1
[248] Sam Owre, John M. Rushby, and Natarajan Shankar. 1992. PVS: A Prototype Verification System. In *Proceedings of the 11th International Conference on Automated Deduction (CADE'92) (LNCS, Vol. 607)*, Deepak Kapur (Ed.). Springer, Germany, 748–752. https://doi.org/10.1007/3-540-55602-8_217
[249] David L. Parnas. 2010. Really Rethinking 'Formal Methods'. *IEEE Comput.* 43, 1 (2010), 28–34. https://doi.org/10.1109/MC.2010.22
[250] Zhaoguang Peng, Yu Lu, Alice Miller, Chris W. Johnson, and Tingdi Zhao. 2013. A Probabilistic Model Checking Approach to Analysing Reliability, Availability, and Maintainability of a Single Satellite System. In *Proceedings of the 7th UKSim/AMSS European Modelling Symposium (EMS'13).* IEEE, USA, 611–616. https://doi.org/10.1109/EMS.2013.102
[251] Ivan Perez, Frank Dedden, and Alwyn Goodloe. 2020. *Copilot 3.* Technical Report NASA/TM–2020–220587. NASA. https://ntrs.nasa.gov/citations/20200003164
[252] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Inf. Softw. Technol.* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007
[253] Lee Pike, Alwyn Goodloe, Robin Morisset, and Sebastian Niller. 2010. Copilot: A Hard Real-Time Runtime Monitor. In *Proceedings of the 1st International Conference on Runtime Verification (RV'10) (LNCS, Vol. 6418)*, Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann (Eds.). Springer, Germany, 345–359. https://doi.org/10.1007/978-3-642-16612-9_26
[254] André Platzer. 2018. *Logical Foundations of Cyber-Physical Systems.* Springer, Germany. https://doi.org/10.1007/978-3-319-63588-0
[255] André Platzer and Jan-David Quesel. 2008. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08) (LNCS, Vol. 5195)*, Alessandro Armando, Peter Baumgartner, and Gilles Dowek (Eds.). Springer, Germany, 171–178. https://doi.org/10.1007/978-3-540-71070-7_15
[256] Amir Pnueli. 1977. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77).* IEEE, USA, 46–57. https://doi.org/10.1109/SFCS.1977.32
[257] Radio Technical Commission for Aeronautics (RTCA). 1992. DO-178B: Software Considerations in Airborne Systems and Equipment Certification. https://www.rtca.org/products/
[258] Radio Technical Commission for Aeronautics (RTCA). 2000. DO-254: Design Assurance Guidance for Airborne Electronic Hardware. https://www.rtca.org/products/
[259] Radio Technical Commission for Aeronautics (RTCA). 2011. DO-333: Formal Methods Supplement to DO-178C and DO-278A. https://www.rtca.org/content/standards-guidance-materials

[260] Radio Technical Commission for Aeronautics (RTCA). 2012. DO-178C/ED-12C: Software Considerations in Airborne Systems and Equipment Certification. https://www.rtca.org/products/

[261] Sarnath Ramnath and Stephen Walk. 2024. Structuring Formal Methods into the Undergraduate Computer Science Curriculum. In *Proceedings of the 16th International NASA Formal Methods Symposium (NFM'24) (LNCS, Vol. 14627)*, Nathaniel Benz, Divya Gopinath, and Nija Shi (Eds.). Springer, Germany, 399–405. https://doi.org/10.1007/978-3-031-60698-4_24

[262] Goutham Rao. 2022. Verification and Validation in VLSI. ChipEdge Technologies. https://chipedge.com/verification-and-validation-in-vlsi

[263] Anne Remke and Mariëlle Stoelinga (Eds.). 2014. *Stochastic Model Checking: Advanced Lectures of the International Autumn School on Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems (ROCKS'12)*. LNCS, Vol. 8453. Springer, Germany. https://doi.org/10.1007/978-3-662-45489-3

[264] Alexandre Riazanov and Andrei Voronkov. 1999. Vampire. In *Proceedings of the 16th International Conference on Automated Deduction (CADE'99) (LNCS, Vol. 1632)*, Harald Ganzinger (Ed.). Springer, Germany, 292–296. https://doi.org/10.1007/3-540-48660-7_26

[265] Xavier Rival and Kwangkeun Yi. 2020. *Introduction to Static Analysis*. MIT Press, USA. https://mitpress.mit.edu/9780262043410/introduction-to-static-analysis/

[266] J. Alan Robinson and Andrei Voronkov (Eds.). 2001. *Handbook of Automated Reasoning*. Elsevier, The Netherlands. https://www.sciencedirect.com/book/9780444508133/handbook-of-automated-reasoning

[267] Luis R. Rodriguez and Julia Lawall. 2015. Increasing Automation in the Backporting of Linux Drivers Using Coccinelle. In *Proceedings of the 11th European Dependable Computing Conference (EDCC'15)*. IEEE, USA, 132–143. https://doi.org/10.1109/EDCC.2015.23

[268] Markus Roggenbach, Antonio Cerone, Bernd-Holger Schlingloff, Gerardo Schneider, and Siraj Ahmed Shaikh. 2022. *Formal Methods for Software Engineering: Languages, Methods, Application Domains*. Springer, Germany. https://doi.org/10.1007/978-3-030-38800-3

[269] A. W. (Bill) Roscoe. 1997. *The Theory and Practice of Concurrency*. Prentice Hall, USA.

[270] Kristin Y. Rozier. 2011. Linear Temporal Logic Symbolic Model Checking. *Comput. Sci. Rev.* 5, 2 (2011), 163–203. https://doi.org/10.1016/j.cosrev.2010.06.002

[271] Kristin Y. Rozier. 2016. Specification: The Biggest Bottleneck in Formal Methods and Autonomy. In *Proceedings of the 8th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE'16) (LNCS, Vol. 9971)*, Marsha Chechik and Sandrine Blazy (Eds.). Springer, Germany, 1–19. https://doi.org/10.1007/978-3-319-48869-1_2

[272] Kristin Y. Rozier. 2019. From Simulation to Runtime Verification and Back: Connecting Single-Run Verification Techniques. In *Proceedings of the 13th Spring Simulation Conference (SpringSim'19)*. IEEE, USA, 1–10. https://doi.org/10.23919/SpringSim.2019.8732915

[273] Kristin Y. Rozier and Johann Schumann. 2017. R2U2: Tool Overview. In *Proceedings of the International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CuBES'17) (Kalpa Publications in Computing, Vol. 3)*, Giles Reger and Klaus Havelund (Eds.). EasyChair, UK, 138–156. https://doi.org/10.29007/5pch

[274] Harry Rudin, Colin H. West, and Pitro Zafiropulo. 1978. Automated Protocol Validation: One Chain of Development. *Comput. Networks* 2 (1978), 373–380. https://doi.org/10.1016/0376-5075(78)90016-8

[275] Neha Rungta. 2022. A Billion SMT Queries a Day. In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22) (LNCS, Vol. 13371)*, Sharon Shoham and Yakir Vizel (Eds.). Springer, Germany, 3–18. https://doi.org/10.1007/978-3-031-13185-1_1

[276] John Rushby. 1993. *Formal Methods and the Certification of Critical Systems*. Technical Report SRI-CSL-93-7. Computer Science Laboratory, SRI International. http://www.csl.sri.com/papers/csl-93-7/

[277] Caitlin Sadowski, Jeffrey van Gogh, Ciera Jaspan, Emma Söderberg, and Collin Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*. IEEE, USA, 598–608. https://doi.org/10.1109/ICSE.2015.76

[278] SAE International. 2022. Architecture Analysis & Design Language (AADL). https://doi.org/10.4271/AS5506D

[279] Monika Seisenberger, Maurice H. ter Beek, Xiuyi Fan, Alessio Ferrari, Anne E. Haxthausen, Phillip James, Andrew Lawrence, Bas Luttik, Jaco van de Pol, and Simon Wimmer. 2022. Safe and Secure Future AI-Driven Railway Technologies: Challenges for Formal Methods in Railway. In *Proceedings of the 11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Practice (ISoLA'22) (LNCS, Vol. 13704)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, Germany, 246–268. https://doi.org/10.1007/978-3-031-19762-8_20

[280] Koushik Sen, Darko Marinov, and Gul Agha. 2005. CUTE: A Concolic Unit Testing Engine for C. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'05)*. ACM, USA, 263–272. https://doi.org/10.1145/1081706.1081750

[281] Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. 2009. Formal Verification of Avionics Software Products. In *Proceedings of the 2nd World Congress on Formal Methods (FM'09) (LNCS, Vol. 5850)*, Ana Cavalcanti and Dennis Dams (Eds.). Springer, Germany, 532–546. https://doi.org/10.1007/978-3-642-05089-3_34

[282] J. Michael Spivey. 1988. *Understanding Z: A Specification Language and its Formal Semantics*. Cambridge Tracts in Theoretical Computer Science, Vol. 3. Cambridge University Press, UK.

[283] Tirumale Sreemani and Joanne M. Atlee. 1996. Feasibility of Model Checking Software Requirements: A Case Study. In *Proceedings of 11th Annual Conference on Computer Assurance (COMPASS'96)*. IEEE, USA, 77–88. https://doi.org/10.1109/CMPASS.1996.507877
[284] Bernhard Steffen. 2024. Rance Cleaveland: a life for formal methods. *Int. J. Softw. Tools Technol. Transf.* 26, 3 (2024), 247–248. https://doi.org/10.1007/s10009-024-00746-1
[285] Ulrich Stern and David L. Dill. 1995. Automatic Verification of the SCI Cache Coherence Protocol. In *Proceedings of the 8th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'95) (LNCS, Vol. 987)*, Paolo Camurati and Hans Eveking (Eds.). Springer, Germany, 21–34. https://doi.org/10.1007/3-540-60385-9_2
[286] Michael Sutton, Adam Greene, and Pedram Amini. 2007. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, USA.
[287] Martyn Thomas. 1990. The role of formal methods in developing safety-critical software. In *Proceedings of the IEE Colloquium on Safety Critical Software in Vehicle and Traffic Control*. IET, UK, 9/1–9/3. https://doi.org/10.1016/0141-9331(90)90127-H
[288] Muffy Thomas. 1994. A Proof of Incorrectness using LP: the Editing Problem from the Therac-25. *High Integrity Systems* 1, 1 (1994), 35–49.
[289] Muffy Thomas. 1994. The Story of the Therac-25 in LOTOS. *High Integrity Systems* 1, 1 (1994), 3–17.
[290] Jan Tretmans. 2017. On the Existence of Practical Testers. In *ModelEd, TestEd, TrustEd (LNCS, Vol. 10500)*, Joost-Pieter Katoen, Rom Langerak, and Arend Rensink (Eds.). Springer, Germany, 87–106. https://doi.org/10.1007/978-3-319-68270-9_5
[291] Kishor S. Trivedi. 2016. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley, UK. https://www.wiley.com/en-us/Probability+and+Statistics+with+Reliability%2C+Queuing%2C+and+Computer+Science+Applications%2C+2nd+Edition-p-x000204691
[292] Willem Visser, Klaus Havelund, Guillaume P. Brat, Seungjoon Park, and Flavio Lerda. 2003. Model Checking Programs. *Autom. Softw. Eng.* 10, 2 (2003), 203–232. https://doi.org/10.1023/A:1022920129859
[293] Werner Vogels. 2021. Diving Deep on S3 Consistency. https://www.allthingsdistributed.com/2021/04/s3-strong-consistency.html
[294] Christian von Essen and Dimitra Giannakopoulou. 2014. Analyzing the Next Generation Airborne Collision Avoidance System. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14) (LNCS, Vol. 8413)*, Erika Ábrahám and Klaus Havelund (Eds.). Springer, Germany, 620–635. https://doi.org/10.1007/978-3-642-54862-8_54
[295] Shuling Wang, Naijun Zhan, and Liang Zou. 2015. An Improved HHL Prover: An Interactive Theorem Prover for Hybrid Systems. In *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM'15) (LNCS, Vol. 9407)*, Michael J. Butler, Sylvain Conchon, and Fatiha Zaïdi (Eds.). Springer, Germany, 382–399. https://doi.org/10.1007/978-3-319-25423-4_25
[296] Colin H. West. 1978. General Technique for Communications Protocol Validation. *IBM J. Res. Dev.* 22, 4 (1978), 393–404. https://doi.org/10.1147/rd.224.0393
[297] The White House. 2024. *Back to the Building Blocks: A Path Toward Secure and Measurable Software*. Technical Report. White House Office of the National Cyber Director (ONCD). https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf
[298] Jeannette M. Wing. 1990. A Specifier's Introduction to Formal Methods. *IEEE Comput.* 23, 9 (1990), 8–24. https://doi.org/10.1109/2.58215
[299] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. *ACM Comput. Surv.* 41, 4 (2009), 19:1–19:36. https://doi.org/10.1145/1592434.1592436
[300] Bohua Zhan, Yi Lv, Shuling Wang, Gehang Zhao, Jifeng Hao, Hong Ye, and Bican Xia. 2022. Compositional Verification of Interacting Systems Using Event Monads. In *Proceedings of the 13th International Conference on Interactive Theorem Proving (ITP'22) (LIPIcs, Vol. 237)*, June Andronick and Leonardo de Moura (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 33:1–33:21. https://doi.org/10.4230/LIPIcs.ITP.2022.33
[301] Zhen Zhang, Wendelin Serwe, Jian Wu, Tomohiro Yoneda Hao Zheng, and Chris Myers. 2016. An improved fault-tolerant routing algorithm for a Network-on-Chip derived with formal analysis. *Sci. Comput. Program.* 118 (2016), 24–39. https://doi.org/10.1016/j.scico.2016.01.002
[302] Yang Zhao and Kristin Y. Rozier. 2014. Formal specification and verification of a coordination protocol for an automated air traffic control system. *Sci. Comput. Program.* 96 (2014), 337–353. https://doi.org/10.1016/j.scico.2014.04.002
[303] Yang Zhao and Kristin Y. Rozier. 2014. Probabilistic model checking for comparative analysis of automated air traffic control systems. In *Proceedings of the 33rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD'14)*. IEEE, USA, 690–695. https://doi.org/10.1109/ICCAD.2014.7001427

Received 19 September 2023; revised 13 July 2024; accepted 15 August 2024