PART I

INTRODUCTION AND STATE OF THE ART

CHAPTER 1

FORMAL METHODS: APPLYING {LOGICS IN, THEORETICAL} COMPUTER SCIENCE

DIEGO LATELLA
ISTI-CNR, Pisa, Italy

1.1 INTRODUCTION AND STATE OF THE ART

Giving a comprehensive definition of formal methods (FMs) is a hard and risky task, since they are still subject of ongoing research and several aspects of FMs, going from the foundational ones to those purely applicative, are in continuous change and evolution. Nevertheless, in order to fix the terms of discussion for the issues dealt with in this book, and only for the purpose of this book, by FMs we will mean all notations having a precise mathematical definition of both their syntax and semantics, together with their associated theory and analysis methods, that allow for describing and reasoning about the behavior of (computer) systems in a formal manner, assisted by automatic (software) tools. FMs play a major role in computer engineering and in the broader context of system engineering, where also the interaction of machines with humans is taken into consideration:

> All engineering disciplines make progress by employing mathematically based notations and methods. Research on 'formal methods' follows this model and attempts to identify and develop mathematical approaches that can contribute to the task of creating computer systems (both their hardware and software components). [26]

The very origins of FMs go back to mathematical logic—or, more precisely, to that branch of mathematical logic that gave rise to logics in computer science

Formal Methods for Industrial Critical Systems: A Survey of Applications, First Edition.
Edited by Stefania Gnesi and Tiziana Margaria.

(LICS) and theoretical computer science (TCS). An eminent example of these roots is the milestone work of Alan Turing [42] where key concepts and theoretical, intrinsic limitations of the algorithmic method—central to computer science in general and FMs in particular—are captured in precise terms. The above issues of computability have become one of the main subjects of scientific development in the field of automatic computation in the 30th's of the previous century. Central to the scientific discussion of that period was the clarification of the key notion of *formal system*, with a clear separation of syntax and semantics and with the notion of a formal proof calculus, defining *computational steps* [41]. Several fields of LICS and TCS have contributed to the development of the foundations of FMs, like language and automata theory, programming/specification language syntax and semantics, and program verification. More recently, solid contributions to the specification and analysis of systems—in particular concurrent/distributed systems—have been provided. These contributions range from the definition of specific notations, or classes of notations, to the development of solid mathematical and/or logic theories supporting such notations, to the development of techniques and efficient algorithms for the automatic or semiautomatic analysis of models of systems or of their requirements, and to the development of reliable automatic tools that implement such algorithms. In the following, we briefly describe some of the most relevant examples of (classes of) notations equipped with solid theories—process algebras; Petri nets; state-based approaches like VDM, Z, and B; and temporal logics—and analysis techniques—model checking and theorem proving. We will also recall abstract interpretation, which, although not bound to any particular notation, as a general theory of approximation of semantics constitutes another key contribution to FMs. We underline that what follows is *not* intended to be a comprehensive treatment of FMs, which would require much more space, and that it offers an overview of the field as of the time of writing:

- Notations and Supporting Theories
    - *Process Algebras.* Process algebra [30] is an algebraic approach to the study of concurrent processes. Its tools are algebraic languages for the specification of processes and the formulation of statements about them, together with congruence laws on which calculi are defined for the verification of these statements. Typical process algebraic operators are *sequential*, *nondeterministic*, and *parallel composition*. Some of the main process algebras are CCS, CSP, and ACP.
    - *Petri Nets.* Petri nets were originally proposed by Petri [33] for describing interacting finite-state machines and are constituted by a finite set of *places*, a finite set of *transitions*, a flow relation from places to transitions and from transitions to places, and weights associated to the flows. A Petri net can be given an appealing graphical representation, which makes specifications intuitively understandable. A state of a Petri net is given by marking its places, that is, associating a number of *tokens* to

places. Rules are defined to fire a transition by moving tokens, hence changing the state of the net. Many extensions and variations of Petri nets have been proposed, for example, adding values to tokens [18], time [19, 29], or probabilities to transitions [3].

- *VDM, Z, B.* The first widely used formal specification languages resorted to the use of traditional mathematical concepts such as sets, functions, and first-order predicate logic. VDM [5] was proposed mainly for describing the denotational semantics of programming languages. Z [38] used the same concepts for defining types, which describe the entities and the state space of the system of interest. Properties of the state space are described in Z by means of *invariant* predicates. State transitions are expressed by relations between inputs and outputs of the operations of the models. The B method [1] adds behavioral specifications by means of *abstract machines*. The language is complemented with a refinement-based development method, which includes the use of theorem proving for maintaining the consistency of refinements. Reference 2 is a nice, informal introduction to the key concepts of system modeling in a mathematical framework as above.
- *Temporal Logics.* Temporal logic [13] is a special type of modal logic, and it provides a formal system for qualitatively describing and reasoning about how the truth values of assertions change over system computations. Typical temporal logic operators include *sometimes P*, which is true now if there is a future moment in time in which *P* becomes true, and *always P*, which is true now if *P* is true at all future moments. Specific temporal logics differ for the model of time they use (e.g., linear time vs. branching time) and/or the specific set of temporal operators they provide.

- Analysis Techniques and Tools
  - *Model Checking.* Model checking is a verification technique in which efficient algorithms are used to check, in an automatic way, whether a desired property holds for a (usually finite) model of the system, typically a state-transition structure, like an automaton [4, 7]. Very powerful logics have been developed to express a great variety of system properties, and high-level languages have been designed to specify system models. Examples of the former are various variants of temporal logic, and notable examples of the latter are process algebras, imperative languages, and graphical notations. Prominent examples of model checkers are SMV [9], SPIN [22], and TLC [28].
  - *Automated Theorem Proving.* Automated theorem proving is the process of getting a computer to agree that a particular theorem is true. The theorems of interest may be in traditional mathematical domains, or they may be in other fields such as digital computer design [37, 44]. When used for system validation, the system *specification S* and its *realization R* are formulas of some appropriate logic. Checking whether the

realization satisfies the specification amounts to verify the validity of the formula $S \Rightarrow R$, and this can be done—at least partially, and sometimes completely—automatically by a computer program, the theorem prover. Classical theorem provers are PVS [39], HOL [43], and Nqthm [6].

- *Abstract Interpretation.* Abstract interpretation is a theory of the approximation of semantics of (programming or specification) languages. It applies both to data and to control. It formalizes the idea that the semantics can be more or less precise according to the considered level of observation. If the approximation is coarse enough, the abstraction of a semantics yields a less precise but computable version. Because of the corresponding loss of information, not all questions can be answered, but all answers given by the effective computation of the approximate semantics are always correct [12].

A key difference between FMs and their mother disciplines of LICS and TCS is that the former attempt to provide the (software or systems) engineer with

> concepts and techniques as thinking tools, which are clean, adequate, and convenient, to support him (or her) in describing, reasoning about, and constructing complex software and hardware systems. [41]

Emphasis is thus on construction rather than reduction and in pragmatics rather than classical issues like completeness. This shift in emphasis applies in general to what W. Thomas calls *logic for computer science* [41]—as opposed to more traditional logic *in* computer science—but it certainly applies to FMs in particular. Are FMs succeeding in their mission? Although a complete answer cannot be given yet—since we have not witnessed a complete and widespread uptake of FMs by an industry in computer engineering—there is a clear trend that justifies a tendency toward a positive answer to the above question, as it will be briefly elaborated below.

FMs have been used extensively in the past for security, fault tolerance, general consistency, object-oriented programs, compiler correctness, protocol development, hardware verification and computer-aided design, and human safety (see References 32 and 46 for extensive bibliographies on the use of FMs in the above areas). There are several IT industries, mainly larger ones like IBM, Intel, Lucent/Cadence, Motorola, and Siemens, which use FM techniques for quality assurance in their production processes [10, 31], and that FMs are making their way into software development practices is clear even from the words of Bill Gates:

> Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability. [17].

FMs and related tools are also used as a supporting technology in branches of fundamental sciences, like biology (see for instance Reference 40), physics, and others [24]. Moreover, FMs are used for the design and validation of safety critical systems, for example, in areas like aerospace* [34]. In the international standards for the engineering of safety critical systems, like those for space or transportation, more and more recommend or mandate the use of FMs. Examples of these engineering standards are the ECSS E-40-01 [15] and the CENELEC EN 50128 [14]. The need of FMs has been recognized also in other scientific communities, in particular the dependability one, as witnessed, for instance, by the organization of the specific workshop on Model Checking for Dependable Software-Intensive Systems in one edition of the International Conference on Dependable Systems and Networks, the most respectable conference in the field (for a discussion on the prominent role that FMs play—and will more and more play in the future—in the area of dependability, see also Reference 45). Finally, several commercial organizations nowadays provide FM-oriented services (some are listed in Reference 11). Unfortunately, the question on the success of FM cannot be fully answered yet also because many tools used in the industry are proprietary tools on which detailed information is very seldom publicly available; similarly, details on the specific methodologies and processes using such tools and the methodologies they support are difficult to be obtained for similar reasons†—a notable exception to this trend is Reference 21 where the AT&T Bell Laboratories NewCoRe project is described, which clearly shows how beneficial FMs can be in the software development process.

On the other hand, one can assess the success of a scientific/technological discipline also by the very advances that have been accomplished in the discipline itself, and, in the last few years, there have been tremendous advances in the field of FMs. First of all, their coverage has been broadened from purely functional aspects of behavior toward nonfunctional features of systems such as the following [27]:

- *Space and Mobility.* Several calculi and logics—and associated support tools—have been extended/developed in order to explicitly deal with notions like the (physical, discrete) *space* occupied by computing elements, their *migration* from one place to another, and its implication on *network connectivity* and the *communication structure*; thus, space and mobility—which are essential notions when developing or reasoning about large distributed networked applications—are first-class objects in these formal frameworks.

* A guide on the selection of appropriate FM can be found in the official site of *Formal Methods Europe* [16].

† Some advocates of FMs claim that this itself is a proof that FM and related tools are considered strategic in industrial contexts and this by itself can show their success.

- *Security.* Several calculi and logics—and associated support tools—have been extended/developed specifically for the modeling of *security* protocols and their desirable properties and for verifying or falsifying that the latter are satisfied by the former.
- *Bounded Resources.* Methods based on logics, types, and calculi—and associated support tools—have been developed for modeling and controlling *allocation* and *deallocation* of resources and for expressing the *trust* involved in resource allocation and its *propagation*.
- *Continuous Time (and Hybrid Systems in General).* In the last 15 years, several varieties of automata, process algebras, and logics—and associated support tools—have been extended with elements of *continuous* nature, like for instance *time*, and functions of continuous variables with their *derivatives*.
- *Stochastic Behavior.* Elements of stochastic behavior, like *probabilistic choices* and *stochastic durations*, have been introduced in models like automata and process algebra, and corresponding operators have been added to proper logics, thus providing conceptual *formal*, language-based tools for the modeling and reasoning about of *performance* and *dependability* attributes and requirements. Automatic software tools for (process algebras/automata) *discrete simulation* and for Markov chains and Markov decision processes *model checking* constitute their practical support counterpart.

An account of advances in the above-mentioned fields together with a rich bibliography can be found in Reference 31. Moreover, the field of application of FMs has been broadened too, including novel disciplines, like computational biology, to mention just one. Finally, the capabilities of the tools supporting FMs have been dramatically improved. For instance, some model checking techniques

> work especially well for concurrent software, which, as luck will have it, is the most difficult to debug and test with traditional means [22],

and nowadays, model checkers are able to deal with system models of up to 10^{100} states and, in some cases, even up to 10^{100} [4, 20]. Nevertheless, there are still several open problems in the area of FM, among which are the following:

- Most specification—and especially automatic verification—techniques do not easily *scale up* due to the infamous *state explosion* problem, which arises when the system to be modeled and verified has many independent actions that can occur in parallel.
- Although significant progress has taken place in the field of compositional specification of concurrent systems (notably by the introduction of process algebras), the same cannot be claimed for verification. In particular,

compositionality is not currently fully exploited in model checking techniques.

- Many specification paradigms and verification techniques developed independently from one another, often giving rise to maybe technically unjustified dichotomies. Lack of *full integration* between different paradigms—like state-oriented ones of some specification languages versus *action/event-oriented* ones of others—or between different verification techniques—like automated theorem proving versus model checking—is likely to be detrimental for each of such paradigms or techniques.
- FMs should find their way toward full acceptance and use in system—and particularly software—engineering. They should be smoothly embedded into more traditional industrial production processes.

1.2 FUTURE DIRECTIONS

In the medium/long-term future, the grand challenge for FMs will be large-scale industrialization and intensification of fundamental research efforts. This is necessary for tackling the challenge of computer—and in particular software—reliability. The latter will be a major grand challenge in computer science and practice [11].

On a more technical level, several research directions will require particular effort. In the following list, some of them are briefly presented. The list is necessarily incomplete and its items necessarily interrelated; the order of appearance of the items in the list does not imply any relative priority or level of importance:

- In the context of *abstract interpretation*, "general purpose, expressive and cost-effective abstractions have to be developed e.g. to handle floating point numbers, data dependencies (e.g. for parallelization), liveness properties with fairness […], timing properties for embedded software [and] probabilistic properties, etc. Present-day tools will have to be enhanced to handle higher-order compositional modular analysis and to cope with new programming paradigms involving complex data and control concepts (such as objects, concurrent threads, distributed/mobile programming, etc.), to automatically combine and locally refine abstractions in particular to cope with 'unknown' answers" [11]. Moreover, new ways for exploiting abstraction for (extended) static analysis, including the use of theorem proving, will need to be investigated [35].
- In the context of *model checking* [7, 8, 24], *abstraction* will be a major pillar. It will be of vital help especially for winning over the state explosion problem. Moreover, it will allow the extension of model checking to infinite models. Finally, it will make model checking of software *programs*

(and not only their specifications) a reality [23]. Research is needed in order to embed abstraction techniques within model checking ones. The following are some of the issues that need to be further investigated: (1) how to build—possibly automatically—an abstract model from a concrete one; (2) how to identify spurious abstract counterexamples provided by the model checking algorithms when checking an abstract model reports an error—spurious counterexamples originate from the very fact that an abstract model necessarily contains less information than the concrete one; (3) how to refine an abstract model when a spurious counterexample is detected in order to eliminate it; (4) how to derive concrete counterexamples from abstract (nonspurious) ones; and (5) how to develop methods for verifying *parameterized* systems, that is, systems with arbitrarily many identical components. Another major pillar will be *compositionality*. There is no hope to be able to model check—and in general, analyze—complex real-life systems of the future, like global ubiquitous systems, without having the possibility of exploiting their structure and architecture—that is, them being composed of simpler components. Compositional reasoning and compositional model checking will play a major role in the area of automated verification of concurrent systems. Finally, a smooth *integration* of model checking and theorem proving will result in a quantum leap in verification for example, by facilitating the modeling and verification of infinite-state systems, for which powerful induction principles will be available together with efficient proof of base cases.

- In the area of *FMs for security*, *access to resources*, and *trust*, we expect the development of security-oriented languages, directly derived from the theories and calculi for mobility and security mentioned in Section 1.1. Moreover, specific protocols for mobile agents to acquire and to manage resources need to be developed, which will base access negotiation on an evaluation of *trust*, the level of which will in turn depend on *knowledge* and *belief* about agents and on how trust is *propagated*. Such protocols will undergo serious and intense investigation and assessment including automatic verification [27].
- In the area of *hybrid* systems, the basic ideas used in the current tools for representing state sets generated by complex continuous functions in hybrid automata (including timed automata) will be further developed, and related efficient algorithms will be developed, which will facilitate automated reasoning on hybrid system models. More emphasis will be put on efforts for unification with control theory [27].
- Model checking techniques for *stochastic* and *probabilistic* models and logics will be further developed in order to scale up to real-life system sizes. Particular emphasis will be put on the problem of counterexample generation and on the tailoring of numerical analysis algorithms and techniques required for such kinds of model checking. Moreover, integration with mobile and hybrid models as above will be required in order to

model and verify essential aspects of future global ubiquitous computing [27].

- *Game semantics* is a promising approach to the formal semantics of specification/programming languages and logics, which has developed from mathematical games, logics, and combinatorial game theory and category theory. Although it lays more on the side of foundations for FMs, it will provide solid basis for advances in model checking—especially with respect to compositionality—partial specification and modeling, integration of qualitative and quantitative approaches, and developments in physics-based models of computation—for example, quantum computing [27].
- Finally, in order to properly face the "engineering challenge" of FMs, a proper *merging* of the languages of formulas and diagrams must be devised. Attempts have been made in the software engineering community, and especially in the industry, most notably with the UML—although not particularly impressive from the mathematical foundations' point of view. On the other hand, there are other, historical examples like the equivalence of Boolean formulas and ordered binary decision diagrams, or regular expressions and finite automata, which have been quite successful and which resulted in useful engineering tools. This should push research on "[t]heories which support merging diagram-based languages with term- or formula-based ones" since this "would help in designing better specification languages" [41].

Although the above list is far to be complete, we can definitely claim that all the above lines of research are essential for properly facing the grand research challenges in information systems [25], in TCS and its applications [36], and for making the potentials of LICS and TCS fully exploited in computer science and engineering as well as in other branches of science and engineering [24].

ACKNOWLEDGMENTS

The author wishes to thank Peter Neumann (SRI Int., USA), Rocco de Nicola (University of Florence, Italy), and Scott Smolka (SUNY, USA) for precious suggestions; Stefania Gnesi and Alessandro Fantechi, for their help on a number of issues; and Mieke Massink and Angelo Morzenti for reviewing the present report.

REFERENCES

1. J. R. Abrial. *The B-Book*. Cambridge University Press, 1996.
2. J. R. Abrial. Faultless systems: Yes we can. *IEEE Computer Society*, 42(9):30–36, 2009.

3. M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, ACM Press, 2(2):93–122, 1984.
4. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
5. D. Bjorner and C. Jones. *Formal Specification and Software Development*. Prentice Hall International, 1982.
6. R. Boyer. Nqthm, the Boyer-Moore prover, 2003. Available at: http://www.cs.utexas.edu/users/boyer/ftp/nqthm/index.html.
7. E. Clarke, E. Emerson, and J. Sifakis. Model checking: Algorithmic verification and debugging. Turing lecture. *Communications of the ACM*, 52(11):75–84, 2009.
8. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Progress in the state explosion problem in model checking. In R. Wilhelm, ed., *Informatics 10 Years Back 10 Years Ahead, Volume 2000 of Lectures Notes in Computer Science*, pp. 176–194. Springer-Verlag, 2000.
9. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
10. E. Clarke, J. Wing, et al. Formal methods: State of the art and future directions. *ACM Computing Surveys*, ACM Press, 28(4):626–643, 1996.
11. P. Cousot. Abstract interpretation based formal methods and future challenges. In R. Wilhelm, ed., *Informatics 10 Years Back 10 Years Ahead, Volume 2000 of Lectures Notes in Computer Science*, pp. 138–156, 2000.
12. P. Cousot. Abstract Interpretation and Semantics, September 30, 2003. Available at: http://www.di.ens.fr/~cousot/Equipeabsint-eg.shtml.
13. A. Emerson. Temporal and modal logics. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science—Vol. B: Formal Models and Semantics*, pp. 995–1072. Elsevier, 1990.
14. European Committee for Electrotechnical Standardization. CENELEC. Railway application—Communications, signalling and processing systems—Software for railway control and protection systems. CENELEC EN 50128. 2011.
15. European Cooperation for Space Standardization ECSS. Space segment software. ECSS E-40-01. 1999.
16. Formal Methods Europe. FME Home Page, September 30, 2003. Available at: http://www.fmeurope.org.
17. B. Gates. Remarks by Bill Gates. WinHEC 2002 Seattle—Washington April 18, 2002. Available at: http://www.microsoft.com/billgates/speeches/2002/04-18winhec.asp [Accessed October 6, 2003].
18. H. Genrich. Predicate/transition nets. In G. Rozenberg, ed., *Advances in Petri Nets 1986, Volume 254 of Lectures Notes in Computer Science*, pp. 207–247. Springer-Verlag, 1986.
19. C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzè. A unified high-level Petri net formalism for time-critical systems. *IEEE Transactions on Software Engineering*, 17(2):160–172, 1991.
20. L. Hoffman. Talking model-checking technology. A conversation with the 2007 ACM A. M. Turing Award winners. *Communications of the ACM*, 51(7):110–112, 2008.
21. G. Holzmann. The theory and practice of a formal method: NewCoRe. In *Proceedings of the 13th IFIP World Congress*, pp. 35–44. IFIP, 1994.

22. G. Holzmann. *The SPIN Model Checker. Primer and Reference Manual*. Addison-Wesley, 2003.
23. R. Jhala and R. Majumdar. Software model checking. *ACM Computing Surveys*, 41(4):21:2–21:54, 2009.
24. D. Johnson. Challenges for Theoretical Computer Science, 2000. Draft Report from the Workshop on Challenges for Theoretical Computer Science held in Portland on May 19, 2000. Available at: http://www2.research.att.com/~dsj/nsflist.html.
25. A. Jones, ed. Grand research challenges in information systems. Computer Research Association, 2003.
26. C. Jones. Thinking tools for the future of computing science. In R. Wilhelm, ed., *Informatics 10 Years Back 10 Years Ahead, Volume 2000 of Lectures Notes in Computer Science*, pp. 112–130, 2000.
27. M. Kwiatkowska and V. Sassone (moderators). Science for Global Ubiquitous Computing, 2003. Proposal for discussion nr. 2 in the context of the Grand Challenges for Computing Research initiative sponsored by the UK Computing Research Committee with support from EPSRC and NeSC.
28. L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.
29. P. Merlin and D. Farber. Recoverability of communication protocols. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.
30. R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science—Vol. B: Formal Models and Semantics*, pp. 1201–1242. Elsevier, 1990.
31. R. Milner, A. Gordon, V. Sassone, P. Buneman, F. Gardner, S. Abramsky, and M. Kwiatkowska. Theories for ubiquitous processes and data. Platform for a 15-year grand challenge, 2003. This paper is written as a background for Reference [27].
32. P. Neumann. Practical architectures for survivable systems and networks. Technical Report Cont. 1-732-427-5099—Final Report, SRI International, 2000. Available at: http://www.csl.sri.com/users/neumann/ [Accessed September 30, 2003].
33. W. Reisig. *Petri Nets—An Introduction, Volume 4 of EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.
34. J. Rushby. Formal methods and the certification of critical systems. Technical Report CSL-93-7, SRI International, 1993. Also issues under the title *Formal Methods and Digital Systems Validation for Airborne Systems* as NASA CR 4551.
35. K. Rustan and M. Leino. Extended static checking: a ten-year perspective. In R. Wilhelm, ed., *Informatics 10 Years Back 10 Years Ahead, Volume 2000 of Lectures Notes in Computer Science*, pp. 157–175, 2000.
36. A. Selman. Challenges for Theory of Computing, 1999. Report of an NSF-Sponsored Workshop on Research in Theoretical Computer Science held in Chicago on March 11–12, 1999. Available at: http://http://www.cse.buffalo.edu/~selman/report/.
37. N. Shankar. Automated deduction for verification. *ACM Computing Surveys*, 41(4):20:2–20:56, 2009.
38. J. Spivey. *The Z Notation—A Reference Manual*. Prentice Hall International, 1989.
39. SRI International—Computer Science Laboratory. The PVS Specification and Verification System, September 30, 2003. Available at: http://pvs.csl.sri.com/.

40. The BioSPI Project. The BioSPI Project Home Page, October 27, 2003. Available at: http:// www.wisdom.weizmann.ac.il/~biopsi.
41. W. Thomas. Logic for computer science: The engineering challenge. In R. Wilhelm, ed., *Informatics 10 Years Back 10 Years Ahead, Volume 2000 of Lectures Notes in Computer Science*, pp. 257–267, 2000.
42. A. M. Turing. On computable numbers with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.
43. University of Cambridge—Computer Laboratory. Automated Reasoning Group HOL page. Available at: http://www.cl.cam.ac.uk/research/hvg/HOL/.
44. Various Authors. Wikipedia, September 30, 2003. Available at: http://www.wikipedia.org
45. J. Woodcock (moderator). Dependable Systems Evolution. A Grand Challenge for Computer Science, 2003. Proposal for discussion nr. 6 in the context of the Grand Challenges for Computing Research initiative sponsored by the UK Computing Research Committee with support from EPSRC and NeSC.
46. J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys*, 41(4):19:2–19:36, 2009.