2018

Mathematical logic in computer science

Mathematical Logic in Computer Science

Assaf Kfoury

June 1, 2017 (last update: February 1, 2018)

1 Introduction

Others have written about the influences of mathematical logic on computer science. I single out two articles, which I have read and re-read over the years:

1. "Influences of Mathematical Logic on Computer Science," by M. Davis [29],
2. "On the Unusual Effectiveness of Logic in Computer Science," by J. Halpern, R. Harper, N. Immerman, P. Kolaitis, M. Vardi, and V. Vianu [60].

The first of these two articles takes stock of what had already become a productive cross-fertilization by the mid-1980's; it is one of several interesting articles of historical character by M. Davis [27, 28, 30] which all bring to light particular aspects of the relationship between the two fields. The second article gives an account of this relationship in five areas of computer science by the year 2000. More on the second article, denoted by the acronym UEL, in Section 3 below.

I wanted to write an addendum to the two forementioned articles, in the form of a timeline of significant moments in the history relating the two fields, from the very beginning of computer science in the mid-1950's till the present. The result is this paper. One way of judging what I produced is to first read the penultimate section entitled 'Timeline', Section 5 below, and then go back to earlier sections whenever in need of a justification for one of my inclusions or one of my omissions.

Disclaimer: This is not a comprehensive history, not even an attempt at one, of mathematical logic in computer science. This is a personal account of how I have experienced the relationship between the two fields since my days in graduate school in the early 1970's. So it is a personal perspective and I expect disagreements.

Notation and Organization: I use italics for naming areas and topics in mathematical logic and computer science, for book titles, and for website names; I do not use italics for emphasis. Single quotes are exclusively for emphasis, and double quotes are for verbatim quotations. I pushed all references and, as much as possible, all historical justifications into footnotes.

Acknowledgments: I updated the text whenever I received comments from colleagues who took time to read earlier drafts. Roger Hindley, Aki Kanamori, and Pawel Urzyczyn provided documents of which I was not aware. I corrected several wrong dates and wrong attributions, and made several adjustments, some minor and some significant, after communicating with Martin Davis, Peter Gacs, Michael Harris, Roger Hindley, Aki Kanamori, Phokion Kolaitis, Leonid Levin, Pawel Urzyczyn, and Moshe Vardi. I owe special thanks to all of them.

2 Areas of Mathematical Logic and How Far They Extend

Mathematical logic is often divided into four major areas:[1]

- *computability* or *recursion theory*,
- *model theory*,
- *proof theory*,
- *set theory*.

Of these four, *computability* has had the strongest impact on the younger discipline of computer science. Arguably, much of *computability* has been taken over by researchers in departments of computer science (or 'informatics' in Europe), though with a difference of emphasis. Computer scientists usually focus on *tractability* or *feasible computability*, while mathematical logicians usually focus on computability as a more theoretical concept with concerns such as non-computability and degrees of unsolvability. The term *theory of computation* usually refers to more practical sub-areas of *computability* – some even outside mathematical logic proper, and closer to *combinatorics*, *number theory*, and *probability theory* – which computer scientists have made their own and almost exclusively developed in recent decades, *e.g.*, *randomness in computation*, *resource-bounded computation*, *combinatorial complexity*, the *polynomial hierarchy*, and many refinements of the preceding with an eye on real-world applications.

Of the four areas of mathematical logic, it is fair to say that *set theory* has contributed the least to computer science, or that a good deal of it has played no role in computer science (*e.g.*, think of the very significant part of *set theory* that deals with *large cardinals*).[2] Nonetheless, there are parts of *set theory* as usually understood that have provided computer science with the means to reason about infinite sets and infinite sequences precisely (*e.g.*, the notions of *well-ordering*, *quasi-well-ordering*, *well-foundedness*, *non-well-foundedness*, and similar notions, or the principles of *simple induction*, *transfinite induction*, *coinduction*, and various *fixed-point* notions),[3] although these are primarily formal means for a rigorous discipline rather than results with potential applications.

It is also fair to say that *model theory* and *proof theory* have played more prominent roles in computer science than *set theory*, and increasingly so. Although in the early decades of computer science, their influence was limited and mostly theoretical, much like that of *set theory*, this is no longer the case. Over time, their importance increased considerably, mostly from the vantage point of real-world applications. Early on, *finite model theory*, particularly in its subarea *descriptive complexity*, became the purview of theoretical computer scientists, much less that of mathematicians, though it called for a theoretician's kind of expertise and interest.[4] If database management systems now form the backbone of 'big data' applications, then *finite model theory*, by way of its intimate synergistic

[1] These are, for example, the four areas identified in a handbook [11], and again in a very nice article [18]. In the latter article's first two sections, there is a discussion of the interaction between mathematical logic and computer science. The authors are four eminent mathematical logicians. It is interesting that they give the lion's share of this interaction to *computability* and *proof theory*, leaving relatively little to *model theory*, and almost nothing explicitly to *set theory*.

[2] For the purpose of a classification, I place *type theory* under *proof theory*, not *set theory*. Although the roots of *type theory* lie in *set theory*, it is today, at least in computer science, the study of logical systems that have at their core Alonzo Church's lambda calculus, which play a central role in the foundations of programming languages, computational logic, and automated proof-assistants. J.L. Bell traces the history of types and type theory from their beginnings in *set theory*, around the turn of the 20th Century, to their gradual migration to other parts of mathematical logic [12].

[3] These notions have been studied in details by D. Sangiorgi. See, in particular, his fascinating historical account [108], which relates the notions across different areas of mathematical logic and computer science. Some of this discussion is reproduced in Sangiorgi's textbook [109] and in the collection of articles [110] which he co-edited with J. Rutten.

[4] Two textbooks I am familar with, by two prominent researchers in this area: N. Immerman [72] and L. Libkin [91]. An earlier comprehensive coverage is in a textbook by H.-D. Ebbinghaus and J. Flum [35].

relationship with *database theory*, acquires a practical dimension well beyond its intrinsic theoretical importance.[5]

Likewise, the early impact of *proof theory* on computer science was mostly theoretical, *e.g.*, the *lambda calculus*, *type theory*, and *substructural logics* came to play a central role in the foundations of programming languages.[6] Later, they provided the foundations for most of the successful general-purpose automated proof-assistants.[7] And later also, various *deductive systems* became the essential background for the development of a wide variety of automated and semi-automated tools for formal verification and satisfiability, with enormous practical consequences.[8]

It is a pointless exercise to try to demarcate precisely the boundaries of these areas within mathematical logic, or the boundaries between any of these areas and other parts of mathematics, if only because of the many overlaps. Nonetheless, even though there are no sharp boundaries, we do not hesitate to say, after reading an article, that it is "more about model theory than about proof theory," or "more about logic than about topology," or the other way around, etc., and so the distinctions are indeed meaningful.

For the ambiguities of where to draw these boundaries, consider the case of *category theory* and how it relates to computer science. Initial concepts of *category theory* were invented in the 1940's and 1950's by mathematicians outside logic (homological algebra and closely related areas in topology). Many years later, *category theory* found its way into logic, and now there is an area called *categorical logic* with many applications in computer science.[9] So, where does *categorical logic* fit in the traditional four-area division of mathematical logic?

If we stick to the four-area division in the opening paragraph, we may have to place (or perhaps 'shoehorn') *categorical logic* somewhere in *proof theory* or *set theory* because of its extensive use of types and topos.[10] But there are also other aspects that make *categorical logic* intersect with *model theory* in a larger sense (*e.g.*, the model theory of typed lambda calculi takes us into the study of *cartesian closed categories*). In fact, *categorical logic* cuts across *proof theory*, *set theory*, and *model theory* – and even *computability theory*.[11] With its recognizably distinct concepts and conventions, it would be easier to identify *categorical logic* as another area of mathematical logic, separate from the

[5] A collection of articles that span *finite model theory*, from its theoretical foundations to its applications, is [54].

[6] I include several topics under this heading, although not always presented in a proof-theoretic framework, but more often in the context of the semantics of programming languages and, more specifically, *functional* programming languages. Some of these topics are in the textbooks by B.C. Pierce [102], C.A. Gunter [57], and J.C. Mitchell [96], in a collection of articles edited by C.A. Gunter and J.C. Mitchell [58], and in another collection edited by B.C. Pierce [103].

[7] A comprehensive account of these proof systems based on *typed lambda calculi* is in a recent textbook by R. Nederpelt and H. Geuvers [98].

[8] Two recent book accounts of methods used in SAT and SMT solvers are: one by D. Kroening and O. Strichman [86] and one by U. Schöning and J. Torán [111]. Implementation details of two highly successful SAT/SMT solvers, Z3 and CVC4, can be collected from their respective websites.

[9] Some even think the future of *categorical logic* cannot be dissociated from computer science: "The fate of categorical logic is presently intimately tied to theoretical computer science," from the last paragraph in a survey by two prominent categorical logicians [93].

[10] Consider, for example, the table of contents in Bart Jacobs' book, *Categorical Logic and Type Theory* [73]. The topics in the last six, and more advanced, chapters of the book include: effective topos, internal categories, polymorphic type theory, advanced fibred categories, first-order dependent type theory, higher-order dependent type theory – which give a sense of how deeply ingrained types, topos, and related notions, are in *categorical logic*.

[11] In the four-part *Handbook of Mathematical Logic* [11], there are two survey articles related to *categorical logic*: "Doctrines in Categorical Logic" by A. Kock and G.E. Reyes [84], which is placed in Part A: Model Theory, and "The Logic of Topoi" by M. Fourman [40], which is placed in Part D: Proof Theory. In the textbook *Category Theory for Computing Science* by M. Barr and C. Wells [10], several sections in different chapters include a categorical treatment of functional programs and computable functions.

four major ones, and to consider its impact on computer science separately.[12]

For another example of ambiguities, now the result of shifting boundaries over time, consider *automata theory*. Its early pioneers in the 1950's and 1960's were all mathematical logicians.[13] While it barely registered a mention among logicians outside *modal logics* at the time,[14] *automata theory* quickly became a core part of computer science. It has remained so, though it has also gone through ebbs and flows of relevance (and, I may add, popularity) in different parts of the field. In the early years, it was squarely placed in *theory of computation*; in later years, through its counterpart *formal-language theory*, it became part of the essential background in the study of programming languages and compiler design (mostly involving the finite automata associated with the lower levels of the Chomsky hierarchy); in later years still, it acquired a special importance in the study of *modal logics* and *temporal logics* which deeply influenced *model checking* (involving various kinds of automata on infinite objects); and now, there are a few who want to push *automata theory* completely out of *theory of computation*.[15] However, if only by way of its importance to the development of *model checking* and the latter's great successes in practice (*e.g.*, the verification, usually automated, of software properties against their formal specification), *automata theory* should occupy pride of place among works of mathematical logicians that have had deep repercussions in computer science.[16]

[12]There are many deep interactions between the four traditional areas, so that a presentation of one cannot avoid broaching elements from the other three, while *categorical logic* stands apart in that it can be omitted altogether from the presentation of any of the four. So it makes sense, for example, that many standard textbooks on mathematical logic include and mix material from all four major areas, and then leave *categorical logic* out entirely. The latter is typically treated in separate and more advanced books. That *categorical logic* stands apart is not an original observation; it is noted, critically, by categorical logicians themselves (*e.g.*, see the two last pages in the last section of [93]).

[13]Just to mention a few of the most prominent: Michael Rabin, Dana C. Scott, J. Richard Büchi, Calvin C. Elgot, Robert McNaughton, and Boris A. Trakhtenbrot, who all went on to become great contributors to *computer science* in later years.

[14]In the four-volume *Handbook of Mathematical Logic* [11], there are no chapters devoted to *modal logics* or to *automata theory* or to what relates the two. One chapter in the handbook, entitled "Decidable Theories" by M. O. Rabin, includes the author's theorem without its proof on the decidability of the second-order theory of two successor functions, though it does also mention that the proof involves an "extension of the theory of automata to cover the case of a finite automaton operating on an infinite tree" but without further explanation. In a later section in the same chapter, there is a passing mention that the forementioned theorem can be used to obtain positive decidability results for "non-classical logics" (here meaning *modal logics*). In the article "Prospects for Mathematical Logic in the Twenty-First Century" [18], also cited in footnote 1, there is a single mention of 'automata theory' in a list of over 50 possible connections between logic and computer science (Table 2 on p. 179) and a single mention of 'modal logics' earlier in the text, with no presentation of results in relation to both topics or to the deep connections between the two.

[15]This history is partly reflected in textbooks with titles containing 'theory of computation' or 'introduction to computation', or which are presented in their introductory chapters as covering such material. At least a third of the chapters in several standard texts [67, 90, 114] written up until the late 1990's (and beyond in the newer editions) are, partially or entirely, on *automata theory*. In more recent years, *automata theory* has fallen into disfavor among some people; the disfavor extends to at least the part on finite automata, not quite to automata on infinite inputs, although the former is arguably a prerequisite for the latter. Consider the following comments in a recent book by a prominent theoretical computer scientist [51, page 36]:

> "We reject the common coupling of computability theory with the theory of automata and formal languages. Although the historical links between these two theories (at least in the West) cannot be denied, this fact cannot justify coupling two fundamentally different theories (especially when such a coupling promotes a wrong perspective on computability theory). Thus, in our opinion, the study of any of the lower levels of Chomsky's Hierarchy [67, Chap. 9] should be decoupled from the study of computability theory (let alone the study of Complexity Theory)."

Is there more than meets the eye in such a categorical opinion? The book leaves out *automata theory* in all its aspects. Ignored by such an opinion is any recognition that the notion of *nondeterminism*, though fundamental in the definition of complexity classes studied in this book, was historically introduced in *automata theory* (first in [105]).

[16]A textbook survey of methods of *model checking* in practical applications is by C. Baier and J.-P. Katoen [7]. A book that deals with implementation issues of a particular model-checker (Spin) is by M. Ben-Ari [13]. A collection of papers exploring many different aspects relating mathematical logic and *automata theory* is [39], and another collection of papers

3 'On the Unusual Effectiveness of Logic in Computer Science'

The paper whose title is the title of this section gives an account of the relationship between the two fields as it stood around the year 2000.[17]

That paper (which I denote by the acronym UEL), authored by six theoretical computer scientists, surveys five areas of computer science where mathematical logic figures most prominently. The titles of the relevant sections are (here numbered according to their order in UEL):

2. Descriptive Complexity
3. Logic as a Database Query Language
4. Type Theory in Programming Language Research
5. Reasoning About Knowledge
6. Automated Verification of Semiconductor Designs

All of these five sections use a profusion of elements from *model theory* and *proof theory*, in different degrees to be sure; to a much lesser extent, elements from *computability theory*; rather little, at least explicitly, from *categorical logic*; and nothing from the deeper parts of *set theory*. Section 2 in UEL can be placed under *finite model theory* and is also perhaps the one that uses the most notions from *computability theory*; Section 3 is mostly about first-order definability in finite relational structures and can be therefore placed under *finite model theory*; Section 4 can be viewed as an explanation for the power of the Curry-Howard Isomorphism (also known as the *propositions-as-types* principle) in functional programming and is therefore closely related to *proof theory*, and can be read as the closest to *categorical logic*; Sections 5 and 6 in UEL use modal logics and their semantics (Kripke structures in Section 5, Linear Temporal Logic in Section 6) and can be construed as involving primarily elements of both *model theory* and *proof theory*.

At the very end of UEL, the authors write:

> "The effectiveness of logic in computer science is not by any means limited to the areas mentioned here. As a matter of fact, it spans a wide spectrum of areas, from artificial intelligence to software engineering. Overall, logic provides computer science with both a unifying foundational framework and a powerful tool for modeling and reasoning about aspects of computation."

That conclusion is as much in force today as it was two decades ago. However, if the aim was to select areas of computer science where mathematical logic had demonstrated its strongest impact, then there was at least one conspicuous omission in UEL: the extensive body of research in the area *logics of programs* which, in both quantity and depth, had an equal or stronger claim for showcasing the close relationship between the two fields by the year 2000.[18]

Of course, unbeknownst to the authors of UEL were the unprecedented advances yet to be experienced

on *modal logics* is [15], which includes a survey by M.Y. Vardi of results connecting automata (on infinite inputs) and modal logics [121].

[17] The title of this paper [60] is probably inspired by earlier articles on the "unusual" or "unreasonable effectiveness" of mathematics in the natural sciences. Among these, there are two (and perhaps others), one by E. Wigner [124] and one by R.W. Hamming [61], whose examination is easily redirected to be about the importance of mathematics and mathematical logic in computer science.

[18] A survey of *logics of programs* up to the late 1980's is by D. Kozen and J. Tiuryn [85], and a book-length account of *dynamic logic* (which is a part of *logics of programs*) up to the late 1990's is by D. Harel, D. Kozen, and J. Tiuryn [62].

after the year 2000. Though the results in UEL were interesting in their own right (for someone versed in mathematical logic) and significant in their respective areas, it is also fair to say their effect outside – say, in systems areas (*e.g.*, operating systems) or in application areas (*e.g.*, machine learning) – had been rather limited or nil. After UEL's publication, changes in the relationship between mathematical logic and computer science, not just incremental but some truly transformative, have taken place in rapid succession – as I try to relate below.

4 Scope of an Ever Deeper Integration

I make a selection of events that illustrate the impact of mathematical logic on the younger discipline over a longer time span, from the mid-1950's until the present. I want to stress my initial disclaimer: My aim is to record significant turning points and moments of recognition, as I see them, not to produce an exhaustive chronology.

I divide the development of computer science into three periods, each of about 20 years. This division is rather arbitrary, but it makes my presentation a little easier: the formative years (1953-1975), the consolidation years (1976-1995), and the mature or growth years (1996-2017).[19]

I omit connections that are strictly related to *computability*, including those under the more restrictive rubric *theory of computation*, the sub-areas dominated by theoretical computer scientists. At least one good reason for this omission: There are just too many *computability*-related milestones which have permeated computer science from the very beginning, perhaps to the point of crowding out other important contributions of mathematical logic in the mind of many.[20]

My focus is therefore on connections between computer science and parts of mathematical logic that are commonly considered in (using the headings of the four-part division in Section 2):

> *model theory*, *proof theory*, and *categorical logic* –
> leaving out parts that are primarily in *computability theory*.[21]

However, there are famous results about limits of *computability* that should be mentioned here (*e.g.*, *NP-completeness*) because they shed light on limits (as well as successes in bypassing these limits) in applications of *model theory* and *proof theory* decades later (*e.g.*, SAT/SMT solvers and model-checkers).[22]

[19] And what I mark as the beginnings of computer science in the 1950's is not recognized by everyone. Others like to say the birth of computer science was some two decades earlier, in the 1930's: "When Turing came to Princeton to work with Church, in the orbit of Gödel, Kleene, and von Neumann, among them they founded a field of computer science that is firmly rooted in logic" [4]. Put that way, mathematical logicians are made the true and sole progenitors of computer science – an assertion which, I suspect, will strike many (most?) computer scientists as a bit of a hyperbole. More emphatically in a similar vein, a prominent *theory-of-computation* researcher marks 1936 as the beginning: "In this extremely readable paper [120], Turing gave birth to the discipline of Computer Science, ignited the computer revolution which radically transformed society, and ..." (Chapter 2 in [123]).

[20] And others are more qualified than I to write a survey of *computability*-related milestones in computer science.

[21] The term *formal methods*, used by many computer scientists, roughly corresponds to my focus in this paper, which is also roughly the focus of Track B in the journal *Theoretical Computer Science* (TCS) and ICALP conferences of the *European Assoc. for TCS* (EATCS). Quoting from the latter website, "Track A [...] correspond to Algorithms, Automata, Complexity, and Games, while Track B to Logic, Semantics, and Theory of Programming." Following this usage, an alternative title for this paper could be 'A Perspective on Formal Methods in Computer Science' or 'A Perspective on Logic, Semantics, and Theory of Programming in Computer Science', instead of 'Mathematical Logic in Computer Science'. At the end I chose the latter title to avoid some of the limitations suggested by the former.

[22] I should add that my focus is in harmony with UEL's focus [60], as presented in Section 3 above. There is no section

4.1 First Two Decades

Figure 1 is a timeline of relevant moments during the first twenty years (which I stretched by including an event in 1953), the formative decades of computer science as a separate discipline.

During these early years, there is relatively little to relate from *model theory*, *proof theory*, and *categorical logic* – and mathematical logic in general outside *theory of computation* – to important milestones in computer science (highlighted with a gray background in Figure 1). In the column with the heading 'Milestones/Accolades', I choose to highlight four:

- The first implementation of Lisp (1959), a functional programming language influenced by Alonzo Church's lambda-calculus.
- The invention of Hoare Logic (1969), which should be considered the first precisely formulated *logic of programs*.[23]
- The programming language Pascal (1970), the first imperative and procedural PL to allow programmers to define higher-order (*i.e.*, nested to any depth level) structured datatypes and procedures.[24]
- The Cook-Levin Theorem on NP-completeness (1971). Stephen Cook directly related his theorem to the complexity of automated theorem-proving (though there was no tool at the time comparable to a modern SAT solver), while Leonid Levin formulated it in relation to search problems.[25]

---

in UEL devoted to a topic in *computability* proper, although there are many such topics that have had major impacts on computer science or that have been developed by computer scientists before the year 2000, around the time of UEL's publication (*e.g.*, the theory of *alternating Turing machines* or the many topics related to *subrecursive hierarchies*).

My focus is also in harmony with the selection of topics in the *Handbook of Logic in Computer Science* [1]. Out of 24 chapters in the first five volumes of the *Handbook*, at least 21 chapters deal primarily with issues related to first-order model theory and universal algebra, category theory and topology, domain theory and denotational semantics, types, modal logics, rewriting systems and process algebras – this information can be gathered by reading titles and introductions – which are all topics with considerable overlaps with traditional *model theory*, *proof theory*, and *categorical logic*. There is one chapter on *recursion theory* and one chapter on *complexity of logical theories*. Outside the chapter on *recursion theory*, there is arguably no chapter on a topic that can be placed mainly under *computability/recursion theory* or the narrower *theory of computation*, and no chapter on a topic that is mainly under *set theory*.

And my focus is again in harmony with the selection of topics in textbooks with titles such as *Logic for Computer Science* [106, 14, 74] and *Logic in Computer Science* [71], whose contents are mostly a mixture of *model theory* and *proof theory* (and a little of *categorical logic* when dealing with types or commutative diagrams).

[23]Others consider work by J. McCarthy [94] and R.W. Floyd [38], both preceding C.A.R. Hoare's paper [66], as the true beginning of logics of programs. And some use the name 'Floyd-Hoare Logic' instead of 'Hoare Logic'. Although these two papers suggested, if only implicitly, the seminal ideas of *loop invariants* and *correctness assertions*, these were not incorporated yet in the rules of a deductive system. Moreover, the programming formalisms used by McCarthy (recursive definitions in a functional-programming style) and Floyd (flowcharts) hampered, arguably, the adaptation of their respective approaches to other programming formalisms in later years. Not every programming formalism can be viewed as a (straightforward) adaptation of recursive definitions, or can be translated into flowcharts. From the early 1970's to the late 1990's and beyond, there was a large body of research (the theory of *program schemes*, initiated by D. Luckham, D.M.R. Park, and M.S. Paterson [92], and their collaborators and students [101, 46, 76, 79], which, among other problems, analyzed program formalisms depending on whether programs are or are not flowchartable (not all are [80]).

[24]Support of higher-order procedures was included in Pascal's language definition, but not in its implementations, which were typically limited to second order. Pascal was also meant to be strongly-typed, but wasn't quite, the most notorious hole in its type system being with variant records: "There is a large hole in the type-checking near variant records, through which some otherwise illegal type mismatches can be obtained," as pointed out early on by a Unix's co-developer, Brian W. Kernighan [75].

[25]The result is in Cook's paper [22], and published independently in Levin's paper [89] (in Russian). Articles in the 1970's, and even in the 1980's and later, often gave credit to Cook only. Though published in 1973, Levin's paper had been mentioned in talks a few years before. A detailed history is an article by Boris Trakhtenbrot [119], which includes

In the column 'Major Conferences' I highlight: ICALP (1972), CADE (1974) and POPL (1974). I highlight ICALP because of its Track B coverage.[26] CADE started as a workshop with a relatively small participation, becoming a full-fledged conference with a larger attendance in the 1980's; it has the distinction of being the first regular, annual or biennial, conference devoted to problems of automated formal reasoning.[27] Even though a significant portion of POPL articles are about pragmatics and implementation of programming languages, a good many other POPL articles cover topics based on ideas that mathematical logicians would readily recognize as coming from *model theory*, *proof theory*, or *categorical logic*. POPL was the first of several annual conferences with similar and overlapping coverages, including PLDI (first held in 1988) and ICFP (first held in 1996).[28]

In the column 'Major Conferences' I list but do not highlight STOC (1969) and FOCS (1975), the twin standard-bearer conferences in *theory of computation* and, with some qualms, outside my focus.[29]

I stretched the 'First Two Decades' by including the Cambridge Diploma in Computer Science (1953), in deference to its promoters' claim that the diploma was the "world's first". A closer look shows it was awarded after a one-year, master's level, course of studies in the use of "electronic computing-machines" in numerical analysis, which is rather different from the coverage of a master's level degree in computer science (or informatics) today.[30]

4.2 Second Two Decades

Figure 2 is a timeline of relevant moments during the second two decades. This is a period of consolidation, when many departments, schools, and colleges, of computer science are established separately, with an identity distinct from engineering and other mathematical sciences. This is also a period of greater recognition of the role of mathematical logic in computer science, when Turing Awards are given to computer scientists working from a distinctly formal-logic perspective:

- Michael O. Rabin and Dana S. Scott (1976), for their joint article "Finite Automata and Their

an annotated English translation of Levin's paper.

[26]The history of ICALP is detailed at the *EATCS* website. See my comments on EATCS's Track B in footnote 21.

[27]CADE's history can be found at its official website *Conf. on Automated Deduction*.

[28]POPL's history is at two webpages: *ACM Digital Library: POPL* and *Symp. on Principles of Prog. Lang.* More on PLDI at: *Prog. Lang. Design and Implementation*. More on ICFP at: *Int. Conf. on Functional Prog.* PLDI is the successor of several conferences with different names held in 1979 and then annually from 1982 to 1987, inclusive. ICFP is the successor of two conferences: LFP (*LISP and Functional Prog.*) and FPCA (*Functional Prog. and Comp. Architecture*); LFP was held from 1980 to 1994, inclusive, every two years; FPCA was first held in 1981 and then from 1985 to 1995, inclusive, every two years.

[29]The relation with STOC and FOCS is not clear cut. It has changed over time. In the 1970's, 1980's, and at least into the 1990's and even later, many articles appearing in STOC and FOCS, though a minority of the total, were arguably more about *formal methods* (*i.e.*, EATCS's Track B) than about *theory of computation* (*i.e.*, EATCS's Track A). These were often presented in separate sessions of STOC and FOCS that many participants would identify as the 'semantics' sessions, even though their topics were not necessarily related to semantics in any obvious way. That articles on formal methods were given relatively short shrift in STOC and FOCS in the early decades was doubtless an incentive (not the only one, of course) for the emergence of several new workshops and conferences focused on logic and formal methods, in the 1980's and 1990's and later.

[30]The Wikipedia page *Cambridge Diploma in Computer Science* says more on what is claimed to be the "world's first". For a sense of how far computer science has moved from its origins in the mid-1950's (or earlier for some, see footnote 19), at least partly because of the influence of mathematical logic, compare with the following: Every master's level student today is expected to know something about *feasible* versus *unfeasible* algorithmic problems (from a course in *theory of computation*) or about *types* (from a course on *principles of programming languages*). NP-completeness was not yet known in the 1950's, so no comparison is possible on this, but *computable functions* and *decision problems* had been studied since at least the 1930's . . . .

Decision Problems" from 1959.[31]

- C.A.R. Hoare (1980), partly in recognition of his invention of Hoare Logic.
- Edgar F. Codd (1981), in recognition of his contributions to the theory of database systems.
- S.A. Cook (1982), partly in recognition of his work on the complexity of formal proofs.
- Robin Milner (1991), in recognition of work which is arguably entirely within the space created by *model theory*, *proof theory*, and *categorical logic*, as adapted to the needs of computer science.

M.O. Rabin, D.S. Scott, and S.A. Cook, trained as mathematical logicians and their entire careers bear witness to the deep connections between the two fields. They studied under two giants of mathematical logic, Alonzo Church (doctoral advisor of both Rabin and Scott) and Hao Wang (Cook's doctoral advisor).[32] I include E.F. Codd in my list because his "introduction of the relational data model and of first-order logic as a database query language definitely changed the course of history [of the database field]." [33]

My qualification for R. Milner above may be questioned, so I quote the Turing Award citation in full, which says that the award resulted from "three distinct and complete achievements," namely:[34]

"1. LCF, the mechanization of Scott's Logic of Computable Functions, probably the first theoretically based yet practical tool for machine assisted proof construction;

2. ML, the first language to include polymorphic type inference together with a type-safe exception-handling mechanism;

3. CCS, a general theory of concurrency."

Milner's achievements 1 and 2 cannot be understood outside the background of *typed λ-calculi* (which mix *rewriting*, *types*, and *deductive systems*, all falling under *proof theory* and *categorical logic* broadly speaking), and Milner's achievement 3 is an effort to formalize a Calculus of Communicating Systems in the form of a *transition system* (again traceable back to *proof theory* and *categorical logic*) and to formalize the latter's semantics using the notion of *bisimulation* (akin and inspired by back-and-forth arguments in *model theory* and *set theory*).[35]

The 1980's and early 1990's saw the very beginnings of three important automated projects, which

[31] Just as interesting as their joint article mentioned in the Turing Award citation [105] are their separate Turing Award lectures: Rabin's lecture was about topics in *theory of computation* (EATCS's Track A) [104], Scott's lecture about topics in *semantics* and *theory of programming* (EATCS's Track B) [112]. Scott's lecture is more aligned than Rabin's lecture with my focus in this paper.

[32] To do full justice to Alonzo Church's contributions to computer science, someone else should survey not only his own accomplishments but also those of his many doctoral students. In addition to Rabin and Scott, they include (in no particular order): Alan Turing, Stephen Kleene, Hartley Rogers, Martin Davis, J. Barkley Rosser (major contributor to the lambda calculus), Peter Andrews (developer of the *TPS automated theorem prover*), John George Kemeny (designer of the *BASIC programming language*), and another two dozens distinguished logicians.

[33] My quote is from a private communication with Phokion Kolaitis. More on the influences of mathematical logic underlying Codd's work are in his citation at the *A.M. Turing Awards* website.

[34] Robin Milner's citation is found at the *A.M. Turing Awards* website.

[35] There is a vast literature by computer scientists on *bisimulation*, *coinduction*, *greatest fixpoint*, and related notions, starting with the publication of two of D.M.R. Park's papers [100, 99] in 1981 where *bisimulation* is fully defined for the first time. My sense is that this body of research deserves a highlighted entry in my timeline (if I get to produce a second, more detailed edition of the timeline) for what became a highly successful and transformative framework for the analysis of concurrency, infinite processes, and related notions. D. Sangiorgi gives a history of these notions in a long article [108], where he also discusses akin notions (sometimes with different names) in *modal logic* and *set theory*, and mentions in passing connections with Ehrenfeucht-Fraïssé games (EF games). Forms of bisimulations can be viewed as special families of partial isomorphisms, corresponding to a restricted type of EF game [118, 39]. A deeper comparison between EF games and *bisimulation* is in several other papers, including by Martin Otto and his colleagues [31, 55].

can be viewed as current standard-bearers of proof assistants (Coq and Isabelle) and model checkers (Spin), listed in Figure 2 under the column 'Milestones/Accolades':[36]

- Coq (1984).[37]
- Isabelle (1986).[38]
- Spin (1991).[39]

The 1980's and early 1990's also mark the beginnings of several academic conferences devoted to various aspects of mathematical logic in computer science, as shown under the column 'Major Conferences' in Figure 2.

Finally, I list as a major milestone the Curry-Howard Isomorphism (1980).[40] This isomorphism expresses a correspondence between two unrelated formalisms – *proof systems* and *programming formalisms* – which asserts that the two are fundamentally the same kind of mathematical objects. It turned out to be an extremely productive correspondence, the basis of a totally different approach to the design of typed programming languages, among other deep changes in both *proof theory* and *programming language theory* and in the relation between the two.[41]

4.3 Third Two Decades

Since the mid-1990's we have witnessed truly transformative changes, some would say 'paradigm shifts', in the relationship between the two fields. We can now talk as much about the impact of *mathematical logic* on *computer science* as about the converse: the impact of *computer science* on *mathematical logic*

[36] With apologies to colleagues who may feel that my singling out of Coq, Isabelle, and Spin for recognition, does not give due credit to their work on other automated systems in later years.

[37] Its history is found on the Web at *What Is Coq?*

[38] More on the *Isabelle* proof-assistant from its webpage.

[39] More on Spin from the webpage *Verifying Multi-threaded Software.*

[40] As with any concept with many threads and contributors, it is a little tricky to give due credit for how the Curry-Howard Isomorphism (CHI) and its many variations have taken shape over the years. For its earliest version, I quote from [64], page 74: "The CHI was first hinted at in print in [23] (1934), and was made explicit in [24] (1942) and in [25] (1958). But it was viewed there as no more than a curiosity." While Curry was first to notice that 'types' are 'theorems', it is probably right to say Howard in the 1960's was first to notice that 'term reduction' is 'proof normalization'. An easy-to-read historical account of the CHI is by P. Wadler [122], which includes an interesting email exchange with Howard and clarifies some of the attributions. Howard's paper was published in 1980 [69], though it had been privately circulated since 1969.

[41] Reviewing the impact of *type theory*, Robert Harper wrote around the year 2000 [60]:

> "In the 1980's and 1990's the study of programming languages was revolutionized by a remarkable confluence of ideas from mathematical and philosophical logic and theoretical computer science. Type theory emerged as a unifying conceptual framework for the design, analysis, and implementation of programming languages. Type theory helps to clarify subtle concepts such as data abstraction, polymorphism, and inheritance. It provides a foundation for developing logics of program behavior that are essential for reasoning about programs. It suggests new techniques for implementing compilers that improve the efficiency and integrity of generated code."

This "revolution" caused by *type theory*, as described by Harper, can be traced back and attributed to the Curry-Howard Isomorphism, though it did not come early enough to block the ravages caused by programming languages like PL/1 (1964, first design) – see "Section 8: Criticisms" in the webpage *PL/1* . A thorough book-length account of the Curry-Howard Isomorphism is by M.H. Sørensen and P. Urzyczyn [115]; a collection edited by Ph. de Groote [32] reproduces several of the seminal papers; and an interesting book, though mostly limited to the author's research interests in arithmetic, is by H. Simmons [113].

(or, more broadly, on *mathematics* generally).[42] To illustrate this converse, I single out five events in five different areas of mathematics (Figure 3), triggered or made possible by logic-based developments in computer science, with each event deserving the distinction of being 'first' in its respective area:

- *Boolean algebra* – a formal proof that every Robbins algebra is a Boolean algebra, using the automated theorem-prover EQP (1997).[43]
- *Graph theory* – a formal proof of the Four-Color Theorem using the automated interactive proof-assistant Coq (2008).[44]
- *Group theory* – a formal proof of the Odd-Order Theorem, also known as the Feit-Thompson Theorem, using the automated interactive proof-assistant Coq (2012).[45]
- *Three-dimensional geometry* – a formal proof of the Kepler Conjecture on dense sphere packings using the automated proof-assistants HOL Light and Isabelle (2015).[46]
- *Number theory* – a formal proof of the Pythagorean-Triple Theorem using the SAT-solver Glucose (2015).[47]

The five preceding formal proofs are not just proofs, but proofs that come with a guarantee of correctness, the result of using logic-based theorem-provers and interactive proof-assistants. In the case of two of these five theorems, the Four-Color Theorem and the Odd-Order Theorem, there were in fact earlier proofs, but always suspected of containing errors because of their length and complexity; these earlier proofs had to be partly aided by *ad hoc* computer programs, *i.e.*, each written for a specific purpose and themselves never verified to be error-free.

In the case of the three other theorems mentioned above, they had resisted all prior attempts, with

[42]The optimism expressed in this section about the reverse impact of computer science on mathematics in general is not shared by many mathematicians, perhaps by most outside the community of mathematical logicians. In particular, the idea that an interactive proof assistant is more than a 'super calculator', and can be used to search for and explore alternatives, seems antithetical to what many profess they do when they prove a theorem. Pierre Deligne, a winner of the Fields Medal, says outright, "I don't believe in a proof done by a computer." And he adds, "I believe in a proof if I understand it," thus suggesting that the use of automated tools is an obstacle to understanding a proof [68].

[43]The theorem asserting that *every Robbins algebra is a Boolean algebra* means that a set of three equations, first formulated by Herbert Robbins, are equivalent to the familiar equations of Boolean algebra that govern unions, intersections, and complements among sets. A history of the problem can be found on the Web at *Robbins algebra*. Technical details are in an article by W. McCune [95]. Simplifications and a particularly lucid presentation are in an article by B. Dahn [26]. The theorem-prover *EQP*, used in solving the Robbins-algebra problem, was derived from the automated theorem-prover *Otter* and developed by the same group, and the latter was more recently superseded by *Prover9*.

[44]The Four-Color Theorem asserts: *The regions of any simple planar map can be colored with only four colors, in such a way that any two adjacent regions have different colors.* A short presentation of the formal proof with Coq is by Georges Gonthier [52] The original proof of the Four-Color Theorem by K. Appel and W. Haken [5] used a computer program, but the correctness of that program (not the proof method) was never completly checked, namely, "the part [in that program] that is supposedly hand-checkable is extraordinarily complicated and tedious, and as far as we know, no one has verified it in its entirety," as reported by N. Robertson, D.P. Saunders, P. Seymour, and R. Thomas, *The Four Color Theorem*. More on Coq from its website *The Coq Proof Assistant*.

[45]The Feit-Thompson Theorem asserts: *Every finite group of odd order is solvable.* A presentation of the formal proof with Coq is by G. Gonthier *et al* [53]. The *Feit-Thompson theorem* webpage discusses its significance for the *Classification of finite simple groups*. The latter is said to be the longest proof in the *List of long mathematical proofs*.

[46]The Kepler Conjecture asserts: *No packing of equally-sized spheres in Euclidean three-dimensional space has density greater than that of the face-centered cubic packing.* It is more than 300 years old and considered the oldest problem in three-dimensional geometry. A history of the problem is at *Kepler conjecture* webpage and in a paper by T. Hales *et al* [59]. The latter paper explains more of the mathematical details than the webpage.

[47]The Pythagorean-Triple Theorem asserts: *It is not possible to divide the set of positive integers into two subsets A and B such that neither A nor B contains a Pythagorean triple.* A triple (a, b, c) of positive integers is Pythagorean if $a^2 + b^2 = c^2$. More on this history at the *Boolean Pythagorean triples problem* webpage. A detailed presentation of *The Glucose SAT Solver* can be found at its website.

or without the help of *ad hoc* computer programs. Their proofs were finally clinched only because of advances in the underlying theory of theorem-provers and proof-assistants (as well as, it must be stressed, improvements in the speed and power of the hardware on which they were implemented).

These five formally-proved theorems by no means exhaust the list of theorems that have been formalized and mechanically proved with a correctness guarantee. I select them here because their proofs resolved long-standing open problems in five different areas of mathematics. [48]

I also single out for inclusion in my timeline (Figure 3) the emergence of the *univalent foundations* of mathematics, largely spurred by the preceding development (of very large and complicated proofs for simply-stated theorems which, if left to humans, 'will remain incomplete or contain errors with probability one'[49]):

- *Univalent foundations* of mathematics and *homotopy type theory* (2006+).[50]

Although the early principles of *univalent foundations* were first formulated in the years from 2006 to 2009 with the specific goal of enabling the use of automated proof-assistants to verify theorems and constructions in classical mathematics, this new area has grown into a much larger body of research in the foundations of mathematics – and provides an excellent illustration for how earlier logic-based developments in computer science have subsequently triggered new and unforeseen directions in mathematics and mathematical logic.[51]

I list several Turing Award winners in Figure 3 who were strongly influenced by logic and formal methods:

- Amir Pnueli (1996), for his work in *temporal logic* and contributions to *formal verification.*
- E.M. Clarke, E.A. Emerson, and J. Sifakis (2007), for their work in *model-checking.*
- Leslie Lamport (2013), for his work in distributed and concurrent systems.

Some may question my inclusion of Leslie Lamport in this list. However, from my own reading, Lamport's work and innovations (particularly the formal specification languages TLA and TLA+, the basis of later implemented model checkers) were highly informed by ideas about *rewriting* and *transition systems* and can therefore be traced back to *proof theory* (here in the form of *temporal logics*).[52]

In Figure 3, under the column 'Milestones/Accolades', I also list:

- Alloy (1997), a model checker that has proved particularly successful in producing counter-examples.[53]

[48] A list of 100 theorems that have been proposed by researchers as benchmarks for theorem provers and proof assistants can be found on the Web at: *Formalizing 100 Theorems.* Their adaptation to two advanced automated systems can be found at: *Formalizing 100 Theorems in Coq*, and *The Top 100 Theorems in Isabelle.* Of those that have been carried out so far, the vast majority are from the years after 2000.

[49] I am paraphrasing M. Aschbacher who wrote "human beings are not capable of writing up a 10,000-page argument which is entirely free of errors. [...] the probability of an error in the proof is one" [6]. M. Aschbacher is a leading researcher in the classification of finite simple groups.

[50] See the webpages *Homotopy Type Theory and Univalent Foundations* and *Homotopy Type Theory* for more details.

[51] For an entertaining account, see Kevin Hartnett, "Will Computers Redefine the Roots of Math?" *Quanta Magazine*, 19 May 2015 (available on the Web at *Will Computers Redefine the Roots of Math?*).

[52] It is also an assessment supported by the citation for Leslie Lamport at the *A.M. Turing Awards* website.

[53] *Alloy: a language & tool for relational models.* From its webpage, "the Alloy language is a simple but expressive logic based on the notion of relations, and was inspired by the Z specification language and Tarski's relational calculus." Like many other model checkers, Alloy is implemented on top of a SAT solver, *i.e.*, Alloy works by reduction to a SAT

- EasyCrypt (2009), a tool combining automated formal reasoning about relational properties of probabilistic computations with adversarial code, which has been successfully used to verify game-based cryptographic proofs.[54]
- The CompCert project (2006), which produced a formally-certified compiler for the C programming language, using the proof-assistant Coq.[55]
- The seL4 project (2009), which verified an operating system micro-kernel with the automated proof-assistants Isabelle and HOL.[56]
- Certification of the FSCQ file system (2015), which uses proof-assistant Coq and logic-of-program Crash Hoare Logic (an extension of Hoare Logic with a 'crash' condition).[57]

As the selection of these last five items reflects my own perspective, they most certainly exclude other recent developments equally worthy of mention, but which I know only by name. I have used Alloy and compared it with other available automated tools in graduate courses, and I have covered parts of CompCert and seL4 in another graduate course. It is worth noting that the FSCQ project is only one of several which started in the last decade or so and whose focus is on producing formally verified systems software; all of them use Coq, Isabelle, or HOL, as automated proof-assistant, together sometimes with an appropriate adaptation or extension of Hoare Logic.[58]

5 Timeline

My proposed timeline is in three parts, in Figures 1, 2, and 3.

I include events that say something significant about the interaction between the two fields, as well as events that are unrelated to this interaction in order to place the former in a wider context. To distinguish the two kinds of events, I highlight those that are logic-related with a gray background. The wider context helps understand the changing character of the interaction, as many parts of computer science become more formalized over the years, mediated by more levels above the hardware (actual physical computers, circuits, ethernets, etc.), and more focused on producing higher-level abstractions and software artifacts.

I try not to 'double list' events, *e.g.*, not to list both the year of a discovery *and* the year of its presentation in a professional journal or conference, and not to list both the year of an article *and* the year (many years later) when that article's author receives public recognition. In all these cases, I choose to list the later year, not the earlier. This is, for example, the case of all the Turing Awards that are in my timeline.

Of course, there are several other awards in computer science besides the Turing Awards, and which are named to honor the greats of mathematical logic. These include the Alonzo Church Award, the Kleene Award, and the Gödel Prize.[59] To keep the timeline within bounds, however, I limit attention

solver, and is as good as the SAT solver it uses.

[54] Details from its website, *EasyCrypt: Computer-Aided Cryptographic Proofs*.

[55] An overview of the project is by its leader Xavier Leroy [87, 88]. Full details and updates are from the project website *CompCert*. More on Coq from its website *The Coq Proof Assistant*.

[56] An overview of the project is by G. Klein *et al* [83]. More on the proof-assistants Isabelle and HOL from their respective websites, *Isabelle* and *HOL*.

[57] More on FSCQ at *A Formally Certified Crash-Proof File System* and the references therein.

[58] Some of these other projects are reviewed by the principals of the FSCQ project in their joint paper [20].

[59] More information on these awards from their respective websites: the *Alonzo Church Award*, the *Kleene Award*, and the *Gödel Prize*.

to Turing Awards and, further, among the latter I only select those bearing an explicit direct influence from mathematical logic (as I see it) – and these are only a small sample of the pervasive influences between the two fields.

[a] Official name: *Diploma in Numerical Analysis and Automatic Computing*, claimed "world's first full-year taught course in CS."

[b] First use of the term in German, by Karl Steinbuch [117].

[c] First use of the term in a CACM article, by Louis Fein [37].

[d] Annual Symposium on *Switching Circuit Theory and Logical Design* (SWCT), first held in Chicago, Illinois.

[e] First use of the term in French, by Philippe Dreyfus [33]. Other terms besides 'computer science' and 'informatics' were proposed: Some survived ('computing science', 'datalogy' in Scandinavia), others disappeared ('comptology', 'hypology', 'computology') .

[f] Promoted by the French government's *Plan Calcul*. More on this at a Wikipedia entry (in French).

[g] Annual Symposium on *Switching and Automata Theory*, first held (as SWAT) in Berkeley, California.

[h] *Symposium on the Theory of Computing* (STOC), first held in Marina del Rey, California.

[i] Pascal is 'almost' but not quite strongly-typed, and the first PL allowing higher-order (limited to second order in actual implementations) structured datatypes and procedures. See the critique by B.W. Kernighan [75] who tends to minimize Pascal's innovations.

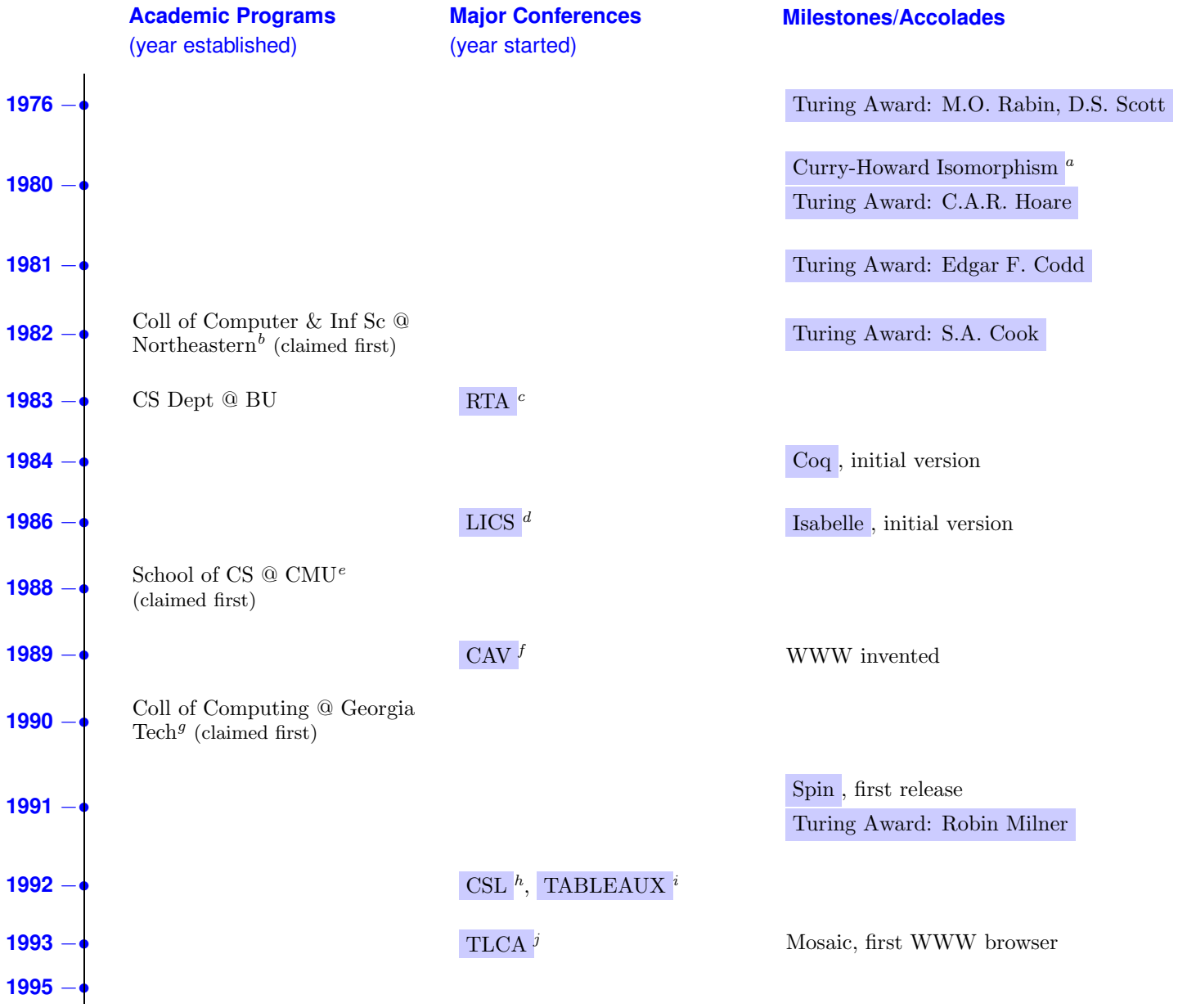
[j] *Int'l Colloquium on Automata, Languages, and Programming* (ICALP), first held in Paris, France. ICALP was held in 1972 first, in 1974 a second time, and annually since 1976. ICALP is highlighted, along with CADE and POPL, because of its Track B coverage.

[k] Mostly Biennial *Conference on Automated Deduction*, first held in Argonne National Lab, Illinois.

[l] Annual Symposium on *Principles of Programming Languages* (POPL), first held in Boston, Massachusetts.

[m] Annual Symposium on *Foundations of Computer Science*, first held (as FOCS) in Berkeley, California.

Figure 1: The First Two Decades (significant logic-related moments highlighted).



[a] Curry published the Curry-Howard Isomorphism in 1958 in his [25], Section 9E, pp. 312-314. Howard's manuscript was written in 1969, published in 1980 [69].

[b] More at *College of Computer and Information Science: Our History*.

[c] Annual Conference on *Rewriting Techniques and Applications*, first held in Dijon, France. More at *RTA Home Page*.

[d] Annual Symposium on *Logic in Computer Science*, first held in Boston, Massachusetts.

[e] More at *Computer Science at Carnegie Mellon: Mission and History*. According to *CMU School of CS: 25th Anniversary*, it was "the first college devoted solely to computer science in the United States, and a model for others that followed."

[f] Annual Conference on *Computer Aided Verification*, first held in Grenoble, France. More at *CAV homepage*.

[g] More at *History of GT Computing*.

[h] *Computer Science Logic* annual conferences, organized by the European Association for CSL.

[i] Int'l Conf on Automated Reasoning with Analytic *Tableaux* and Related Methods, first held in Karlsruhe, Germany.

[j] Biennial Annual Conference on *Typed Lambda Calculi and Applications*.

Figure 2: The Second Two Decades (significant logic-related moments highlighted).

[a] The *Federated Logic Conference* (FLoC) is held roughly every four years. FLoC is now the premier international conglomeration of several mathematical logic and computer science conferences that deal with the intersection of the two fields.

[b] Java JDK 1.0 was developed by J. Gosling, M. Sheridan, and P. Naughton, since 1991, released in January 1996.

[c] The *Int'l Joint Conference on Automated Reasoning* (IJCAR) is held semi-regularly every two-to-four years. IJCAR is a conglomeration of several conferences: CADE, FTP, TABLEAUX, and others. More information at *Home – IJCAR*.

[d] There is a debate about who was the first to coin the expression and when. However, in the current sense of a paradigm in which users increasingly access software and computer power over the Web instead of on their desktops, it seems the expression was first used on August 9, 2006, when then Google CEO Eric Schmidt introduced it to an industry conference.

[e] Int'l Conference on *Formal Structures* for *Computation* and *Deduction*.

Figure 3: The Third Two Decades (significant logic-related moments highlighted).

6 Concluding Remarks

Stressing only the positive in past sections, I may have presented a permanent picture of harmony and close collaboration between mathematical logic and computer science. As a matter of fact, it has not been always so. From time to time, a few blemishes have marred this picture – or maybe they are just a reflection of a far-flung fast-growing field.

When computer scientists do not know what logicians did already. I mention five examples in no particular order, a sample from the earlier years of computer science, more than three decades ago (purposely). There are doubtless many, old and new, of which I am ignorant.

1. The *polymorphic lambda calculus*, also known as *System F*: It was first formulated (and many of its deep properties established in relation to second-order logic) by the logician Jean-Yves Girard in the years 1970-72. It was independently re-formulated, with different syntactic conventions, by the computer scientist John Reynolds and published in 1974.[60]
2. The *functions definable in the polymorphic lambda-calculus are exactly the recursive functions provably total in second-order arithmetic*: This result was first proved by Jean-Yves Girard and published in 1971. It was proved again independently by Richard Statman and reported in 1981, spurred by other computer scientists' earlier inconclusive attempts.[61]
3. The *pebble game*, an ubiquitous concept in many parts of computer science, which has undergone many variations and extensions over the years: The original simplest version is usually credited to Michael Paterson and Carl Hewitt, who defined it in 1970, unaware of the logician Harvey Friedman's earlier formulation of the same idea (in a long and highly technical article). Friedman completed and published his article in 1969.[62]
4. The *algorithm to decide typability of terms in the simply-typed lambda-calculus*, often called the Hindley-Milner or Damas-Hindley-Milner algorithm: A version was defined and proved correct by Roger Hindley in the late 1960's, a related version was independently defined by Robin Milner in the late 1970's, and the latter was re-written and proved correct by Luis Damas in 1984. This history was upended by Hindley in 2005 when he discovered that Max Newman had been the first to develop an algorithm for the problem, and to prove it correct, in the early 1940's.[63]

[60] Girard's formulation and results appeared in print in [49], Reynolds' formulation appeared in [107]. In a later article in which he expanded on his results about System F from the early 1970's, Girard observed that "the proofs of these results have been often redone in the current [computer science] literature" [50].

[61] Pawel Urzyczyn pointed me to this discrepancy, which had been also noted by others, for example, by Gérard Huet in his lecture notes [70] (end of Section 10.3.3). Girard's paper is [48], Statman's paper is [116], a lucid presentation of the result is by Sørensen and Urzyczyn [115] (Chapter 12).

[62] Though he did not call it the 'pebble game', Friedman's formulation was in a report for the Logic Colloquium, held in Manchester in August 1969, and included in its proceedings [41]. I ran into Friedman's formulation by chance, while preparing for an article where I used the pebble game in several proofs [77]. Paterson's and Hewitt's original report was dated November 1970 [101]. Later, I wrote a follow-up article [78] where I made explicit the correspondence between the Friedman version and the Paterson-Hewitt version, showing that the two were also used for the same purpose (informally, at the simplest level: programs accessing $n+1$ storage locations can do computations that programs accessing only n storage locations cannot do, unless pairing functions are available). There are still today competing claims about the origins of *pebbling* and the *pebble game* which ought to be sorted out; see, for example, the websites *Pebble Game* and *Graph Pebbling*, neither of which, incidentally, mention Friedman's earlier work or Paterson's and Hewitt's. Part of the complication, it seems, is that *pebbling* and the *pebble game* and derived concepts are now used in separate areas of computer science, each with its own motivations and research community.

[63] The historical facts are recounted in an article by Roger Hindley [65], including all the pertinent references (his own paper, Milner's paper, Damas' paper, and Newman's paper). I am indebted to Pawel Urzyczyn for alerting me to Hindley's revised history of the typability algorithm. A complementary article, with additional discussion of Newman's

5. What is known as *Newman's Lemma*, a fundamental result widely used by computer scientists dealing with combinatory reduction systems, including the lambda calculus: It states that 'local confluence' of a notion of reduction, say $\triangleright$, implies 'confluence' of $\triangleright$ if all $\triangleright$-reduction sequences are finite. Max Newman proved the lemma in 1942. Though stated a little differently, the lemma was in essence anticipated and proved by Axel Thue some three decades earlier![64]

But these examples are just innocent misattributions or delayed attributions, causing no more damage than some duplication of effort. Far more serious is the situation with computer algebra systems which were developed, since their beginnings in the mid-1960's, outside logic and formal-methods concerns.

The situation with *computer algebra systems* (CAS's). CAS's are another category of automated general-purpose systems developed by computer scientists. They provide integrated environments for the manipulation of mathematical expressions in algebra (*e.g.*, various kinds of optimizations), number theory (*e.g.*, numerical computations and series operations), analysis (*e.g.*, differentiation and integration), and other deeper areas of mathematics – all very useful in applications.[65]

In contrast to automated theorem provers and interactive proof assistants – to which I gave the lion's share of accolades for the third period in my timeline (Figure 3) – popular commercially-available CAS's are not built on principles of formal logic. They do not carry out calculations according to formalized proofs or satisfying formally-specified guarantees of correctness. With no formal safeguards available to the user, they have sometimes produced obscure errors, difficult to trace and difficult to rectify.[66] And yet, despite their "notorious unsoundness," CAS's are "in widespread use in mathematics, and it is not always so easy to explain away the lack of concern about their unsoundness."[67]

Be that as it may, there is now an increased awareness for the need to build CAS's on stronger formal foundations.[68] This effort goes beyond earlier work of augmenting pre-existing CAS's with logic-based functionalities or, conversely, augmenting pre-existing theorem provers and proof assistants with CAS functionalities.[69] The new effort is more principled in that it aims to combine the two sides – proof search and interactive proof-assistance on the one hand, algebraic domain-specific computation on the other – in an integrated bottom-up formal design.[70] This activity is still limited to a few research

version of the algorithm and its history, is by H. Geuvers and R. Krebbers [47].

[64] I am indebted to Roger Hindley who directed me to the history of Newman's Lemma, reported in his history of the lambda calculus, co-authored with Felice Cardone [19]; see in particular Section 5.2 on page 738 in that chapter, which includes all the pertinent references (Newman's article of 1942, Axel Thue's article of 1910, and an article by M. Steinby and W. Thomas from 2000 that summarizes Thue's paper in English).

[65] Perhaps the current most popular CAS's are Mathematica and Maple. A comprehensive *List of CAS's* is available on the Web. A valuable historical overview of CAS's is by Joel Moses, the lead developer of the first CAS, Macsyma [97].

[66] A disturbing example involving Mathematica was reported in a recent article [34]. The authors accidentally discovered an error by comparing Mathematica's calculations with those of Maple. To determine which of the two CAS's was at fault, they had to run both on multiple randomly generated inputs; they did not have at their disposal formally specified conditions under which the CAS's can be safely used and return outputs with correctness guarantees. More disturbing still than an error whose source could not be identified or located (Mathematica and Maple are not open-source) was the fact that an earlier release (Mathematica 7) did not show the error, while later releases (Mathematica 9 and 10) did.

[67] I am quoting from an article by Alan Bundy [17], a prominent advocate for the use of automated theorem-provers.

[68] An approximate but useful distinction between the two categories is that, while automated theorem provers and proof assistants are 'super search engines' (of formal proofs, built from axioms and deduction rules), CAS's are 'super calculators' (mostly of numbers, derived from equations and formulas).

[69] This earlier work is exemplified by various add-ons and interfaces, to connect the two sides without fundamentally re-designing either. Examples: Analytica with Mathematica [21], Isabelle with Maple [8], Isabelle with the computer algebra library Sumit [9], Theorema with Mathematica [16], PVS with Maple and Mathematica [3, 2].

[70] In some ways, this more recent effort is akin to the earlier development of SMT solvers as extensions of SAT solvers, which was also an integrated development based on logic and formal methods. What is different now is that it aims to

groups, but it gives an inkling of what may yet become a new big frontier in the interaction between mathematical logic and computer science.

Do pure mathematicians agree or care? The optimism expressed in earlier sections about a growing mutual dependence between computer science and mathematical logic – and mathematics in general – is not shared by everyone. Many view the good effects going in one direction only: That mathematics and its formalisms underlie (or should underlie) much of computer science is taken for granted, but that computer science may have (or will have) an equally important impact of a different kind on mathematics is taken as a dubious claim. In fact, there appears to be an inbred indifference or even resistance among many pure mathematicians, notably many in the core traditional areas, to anything involving computers in their own work beyond routine pedestrian tasks (Google search, email, typesetting with LaTeX).[71]

A particularly damning remark was once made by Alexandre Grothendieck, an eminent Fields Medalist and algebraic geometer. He objected to the "purported proof [of the Four-Color Theorem], whose validity is no longer based on a firm belief that derives from the understanding of a mathematical situation, but rather on the trust that one is willing to put in a machine that lacks the capacity to understand."[72] This view was echoed by Pierre Deligne, another Fields Medalist: "I don't believe in a proof done by a computer [...]. In a way, I am very egocentric. I believe in a proof if I understand it, if it's clear."[73] But these were reactions to *ad hoc* computer programs, written for specific exhaustive searches (enormous beyond human ability), not to the more recent breakthroughs resulting from the use of automated logic-based systems (surveyed in Section 4.3). Nevertheless, the idea that the latter can become instruments of mathematical progress is still a minority view, rejected by many (most?) pure mathematicians.[74]

But change is coming. Dissenting views have been expressed by eminent members of the pure mathematical community itself.[75] Most notable are Vladimir Voevodsky's, who made contributions to core

extend or combine in a single design more features and functionalities of advanced systems (for proof search, interactive proof-assistance, and domain-specific algebraic computation). I include the following projects as examples of the more recent effort: the *LEAN Theorem Prover* and its publications (available from its website), the *FoCaLiZe* project and its publications (available from its website), as well as individual contributions by others (*e.g.*, the work of M.T. Khan and W. Schreiner [81, 82] and some of Sicun Gao's recent work with his collaborators [42, 43, 44, 45]).

[71] That attitude was more entrenched prior to the great breakthroughs of automated theorem provers and interactive proof assistants (Section 4.3), with good reasons perhaps, given the checkered history of CAS's and the emergence of what is called *experimental mathematics* since the early 1990's, which owes its existence to computers and carries a bad name among the traditionalists. David Mumford, a prominent mathematician and Fields Medalist, started his career in algebraic geometry before converting to an applied area of computer science (vision) in the 1980's. Having known practitioners on both sides of the divide, Mumford could write from experience that "the pure mathematical community by and large still regards computers as invaders, despoilers of the sacred ground" (quoted in [68]). More than two decades later, that divide and the debates it provokes persist, though less sharply. Consider, for example, what eminent number-theorist and algebraist Michael Harris has to say on this divide [63].

[72] This is a loose translation of the original French: "une 'démonstration' qui ne se trouve plus fondée dans l'intime conviction provenant de la compréhension d'une situation mathématique, mais dans le crédit qu'on fait à une machine dénuée de la faculté de comprendre," taken from the footnote on page 137 of Grothendieck's unpublished manuscript [56].

[73] Quoted in [68]. Deligne's statement, as well as Grothendieck's, precede the great advances of theorem provers and interactive proof assistants since the late 1990's.

[74] Consider, for example, what another pure mathematician says dismissively about logical formalisms and, by implication, automated logic-based systems in mathematics: "My mathematics colleagues almost never think about mathematical logic. [...] They simply learn not to make certain moves that lead to trouble (as long as the referee doesn't complain, what, me worry?). [...] So mathematicians work informally and have always done so; there is almost no trace of mathematical logic in most of the history of modern mathematics." [36]

[75] Some are expressed in Michael Harris' blog on the *Univalent Foundations* program. There is a clear separation between mathematicians (mostly against) and computer scientists (all in favor); some of the former are wavering and a

areas of pure mathematics (*e.g.*, *motivic homology* and *cohomology*, for which he received the Fields Medal) and, since around 2005 and until his untimely death in 2017, to the foundations of automated interactive proof-assistants (*univalent foundations* and *homotopy type theory*).

few topologists even express strong support for the program.

REFERENCES

My list of references is not a bibliography. It is limited to references I used to justify my timeline. I make no claim of fairness in my selection. Except for a handful of historical character which I consulted for this article, and another handful supplied by colleagues who read earlier drafts, all the other citations are from books, articles, and webpages, that I have referenced or required in courses I have taught over nearly four decades. (And, no, I didn't read them all from cover to cover! In most of the books, I only read very, very few sections of particular interest to me.)

[1] Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum. *Handbook of Logic in Computer Science*. Oxford University Press, Inc., 1992. 22

[2] Andrew Adams, Martin Dunstan, Hanne Gottliebsen, Tom Kelsey, Ursula Martin, and Sam Owre. Computer Algebra Meets Automated Theorem Proving: Integrating Maple and PVS. In *Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '01, pages 27–42, London, UK, UK, 2001. Springer-Verlag. 69

[3] Andrew Adams, Hanne Gottliebsen, Steve A. Linton, and Ursula Martin. Automated Theorem Proving in Support of Computer Algebra: Symbolic Definite Integration As a Case Study. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, ISSAC '99, pages 253–260, New York, NY, USA, 1999. ACM. 69

[4] Andrew W. Appel. The Birth of Computer Science at Princeton in the 1930's. In Alan M. Turing and Andrew W. Appel, editors, *Alan Turing's Systems of Logic: The Princeton Thesis*. Princeton University Press, Princeton, N.J., 2012. 19

[5] Kenneth Appel and Wolfgang Haken. The Solution of the Four-Color-Map Problem. *Scientific American*, 237:108–121, 1977. 44

[6] Michael Aschbacher. Highly Complex Proofs and Implications of Such Proofs. In A. Bundy, M. Atiyah, A. Macintyre, and D. Mackenzie, editors, *The Nature of Mathematical Proof*, pages 2401–2406. Philosophical Transactions of the Royal Society, 2005. 49

[7] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2007. 16

[8] Clemens Ballarin, Karsten Homann, and Jacques Calmet. Theorems and Algorithms: An Interface Between Isabelle and Maple. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, ISSAC '95, pages 150–157, New York, NY, USA, 1995. ACM. 69

[9] Clemens Ballarin and Lawrence C. Paulson. Reasoning about Coding Theory: The Benefits We Get from Computer Algebra. In Jacques Calmet and Jan Plaza, editors, *Artificial Intelligence and Symbolic Computation: International Conference AISC'98, Proceedings*, pages 55–66, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. 69

[10] Michael Barr and Charles Wells. *Category Theory for Computing Science, Third Edition*. Prentice-Hall Int. Series in Computer Science, 1998. 11

[11] J. Barwise, H.J. Keisler, K. Kunen, Y.N. Moschovakis, and A.S. Troelstra. *Handbook of Mathematical Logic*. Elsevier (North Holland), 1977. 1, 11, 14, 6

[12] John L. Bell. Types, Sets, and Categories. In Dov Gabbay, Akihiro Kanamori, and John Woods, editors, *Handbook of the History of Logic*, volume 6, pages 633–683. Elsevier (North Holland), 2012. 2

[13] Mordechai Ben-Ari. *Principles of the Spin Model Checker*. Texts in Theoretical Computer Science. Springer, 2008. 16

[14] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer, 2012. 22

[15] Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2006. 16

[16] Bruno Buchberger, Tudor Jebelean, Temur Kutsia, Alexander Maletzky, and Wolfgang Windsteiger. Theorema 2.0: Computer-Assisted Natural-Style Mathematics. *Journal of Formal Reasoning*, 9(1):149–185, 2016. 69

[17] Alan Bundy. Automated Theorem Provers: A Practical Tool for the Working Mathematician? *Annals of Mathematics and Artificial Intelligence*, 61(1):3–14, January 2011. 67

[18] Samuel R. Buss, Alexander S. Kechris, Anand Pillay, and Richard A. Shore. The Prospects for Mathematical Logic in the Twenty-First Century. *The Bulletin of Symbolic Logic*, 7(2):169–196, June 2001. 1, 14

[19] Felice Cardone and J. Roger Hindley. Lambda-Calculus and Combinators in the 20th Century. In Dov Gabbay and John Woods, editors, *Handbook of the History of Logic (Logic from Russell to Church)*, volume 5, pages 723–818. Elsevier (North Holland), 2009. 64

[20] Haogang Chen, Daniel Ziegler, Tej Chajed, Adam Chlipala, M. Frans Kaashoek, and Nickolai Zeldovich. Using Crash Hoare Logic for Certifying the FSCQ File System. In Ethan L. Miller and Steven Hand, editors, *Proceedings of 25th ACM Symp. on Operating Systems Principles*, pages 18–37, Monterey, California, October 2015. 58

[21] Edmund Clarke and Xudong Zhao. Analytica - A Theorem Prover for Mathematica. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1992. 69

[22] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proc. of Third Annual ACM Symp. on Theory of Computing*, pages 151–158. ACM, 1971. 25

[23] Haskell B. Curry. Functionality in Combinatory Logic. *Proceedings of the National Academy of Science, USA*, 20:584–590, 1934. 40

[24] Haskell B. Curry. The Combinatory Foundations of Mathematical Logic. *Journal of Symbolic Logic*, 7:49–64, 1942. 40

[25] Haskell B. Curry and Robert Feys. *Combinatory Logic, Volume I.* Studies in Logic and the Foundations of Mathematics. Noth Holland, 1958. 40, *a*

[26] Bernd I. Dahn. McCune's Computer-Generated Solution of Robbins Problem. *Journal of Algebra*, 208:526–532, October 1998. 43

[27] Martin Davis. Why Gödel Didn't Have Church's Thesis. *Information and Control*, 54:3–24, 1982. 1

[28] Martin Davis. Mathematical Logic and the Origin of Modern Computers. In Esther R. Phillips, editor, *Studies in the History of Mathematics*, pages 137–167. Mathematical Association of America, Washington, DC, USA, 1987. 1

[29] Martin Davis. Influences of Mathematical Logic on Computer Science. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 315–326. Oxford University Press, 1988. 1

[30] Martin Davis. The Early History of Automated Deduction. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 1, pages 4–15. Elsevier, 2001. 1

[31] Anuj Dawar and Martin Otto. Modal Charactrization Theorems over Special Classes of Frames. *Annals of Pure and Applied Logic*, 161:1–42, 2009. 35

[32] Philippe de Groote, editor. *The Curry-Howard Isomorphism.* Cahiers du Centre de Logique, 8. Academia, Louvain-la-Neuve, 1995. 41

[33] Philippe Dreyfus. *L'Informatique.* Gestion, Paris, 1962. *e*

[34] Antonio J. Durán, Mario Pérez, and Juan L. Varona. The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them? *Notices of the AMS*, 61(10):1249–1252, November 2014. 66

[35] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Perspective in Mathematical Logic. Springer, 1995. 4

[36] David A. Edwards. Response to Quinn. *Notices of the AMS*, 59(3):366, 2012. This is a letter in response to an earlier article: *A Revolution in Mathematics? What Really Happened a Century Ago and Why It Matters Today* [**?**]. 74

[37] Louis Fein. The Role of the University in Computers, Data Processing, and Related Fields. *Communications of the ACM*, 2(9):7–14, 1959. *c*

[38] Robert W. Floyd. Assigning Meanings to Programs. In J. T. Schwarz, editor, *Mathematical Aspects of Computer Science (Proceedings of a Symposium in Applied Mathematics)*, volume 19, pages 19–32. American Mathematical Society, Providence, R.I., 1967. 23

[39] Jörg Flum, Erich Grädel, and Thomas Wilke, editors. *Logic and Automata: History and Perspectives*. Texts in Logic and Games, Volume 2. Amsterdam University Press, 2008. 16, 35

[40] Michael P. Fourman. The Logic of Topoi. *Studies in Logic and the Foundations of Mathematics*, 90:1053 – 1090, 1977. Reproduced in *Handbook of Mathematical Logic* [11]. 11

[41] Harvey Friedman. Algorithmic Procedures, Generalized Turing Algorithms, and Elementary Recursion Theory. In R.O. Gandy and C.M.E. Yates, editors, *Logic Colloquium '69*, pages 361–389. North-Holland, 1970. 62

[42] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, pages 208–214, 2013. 70

[43] Sicun Gao, Soonho Kong, and Edmund M. Clarke. Satisfiability Modulo ODEs. *Formal Methods in Computer-Aided Design (FMCAD)*, abs/1310.8278, 2013. Also downloadable from http://arxiv.org/abs/1310.8278. 70

[44] Sicun Gao, Soonho Kong, and Edmund M. Clarke. Proof Generation from Delta-Decisions. *Int'l Conference on Symbolic and Numerical Algorithms for Scientific Computing (SYNASC)*, abs/1409.6414, 2014. Also downloadable from http://arxiv.org/abs/1409.6414. 70

[45] Sicun Gao and Damien Zufferey. Interpolants in Nonlinear Theories Over the Reals. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Proceedings*, pages 625–641, 2016. 70

[46] Stephen J. Garland and David C. Luckham. Program Schemes, Recursion Schemes, and Formal Languages. *Journal of Computer and System Sciences*, 7(2):119 – 160, 1973. 23

[47] Herman Geuvers and Robbert Krebbers. The Correctness of Newman's Typability Algorithm and Some of its Extensions. *Theoretical Computer Science*, 412(28):3242 – 3261, 2011. Festschrift in Honour of Jan Bergstra. 63

[48] Jean-Yves Girard. Une Extension de l'Interprétation de Gödel à l'Analyse, et son Application à l'Elimination des Coupures dans l'Analyse et la Théorie des Types. *Studies in Logic and the Foundations of Mathematics*, 63:63–92, 1971. Proceedings of the Second Scandinavian Logic Symposium. 61

[49] Jean-Yves Girard. Interprétation Fonctionelle et Elimination des Coupures dans l'Arithmétique d'Ordre Supérieur, 1972. Thèse dEtat, Université Paris VII. 60

[50] Jean-Yves Girard. The System F of Variable Types, Fifteen Years Later. *Theoretical Computer Science*, 45:159–192, 1986. 60

[51] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, New York, NY, USA, 1 edition, 2008. 15

[52] George Gonthier. Formal Proof – The Four-Color Theorem. *Notices of the AMS*, 55(11):1382–1393, 2008. 44

[53] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-checked Proof of the Odd Order Theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Proceedings of the 4th International Conference on Interactive Theorem Proving*, ITP'13, pages 163–179, Berlin, Heidelberg, 2013. Springer-Verlag. 45

[54] Erich Grädel, P.G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M.Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. Springer, 2007. 5

[55] Erich Grädel and Martin Otto. The Freedoms of (Guarded) Bisimulation. In Alexandru Baltag and Sonja Smets, editors, *Johan van Benthem on Logic and Information Dynamics*, pages 3–31. Springer International Publishing, Cham, 2014. 35

[56] Alexandre Grothendieck. Récoltes et Semailles, 1986. Unpublished manuscript, downloadable from various sites on the Web, Grothendieck's non-mathematical writings, Récoltes et Semailles, among others. 72

[57] Carl A. Gunter. *Semantics of Programming Languages*. MIT Press, 1992. 6

[58] Carl A. Gunter and John C. Mitchell, editors. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*. MIT Press, 1994. 6

[59] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, and et al. A Formal Proof of the Kepler Conjecture. *Forum of Mathematics, Pi*, 5, 2017. Also downloadable from https://arxiv.org/abs/1501.02155. 46

[60] Joseph Halpern, Robert Harper, Neil Immerman, Phokion Kolaitis, Moshe Vardi, and Victor Vianu. On the Unusual Effectiveness of Logic in Computer Science. *The Bulletin of Symbolic Logic*, 7(2):213–236, March 2001. 2, 17, 22, 41

[61] Richard W. Hamming. The Unreasonable Effectiveness of Mathematics. *American Mathematical Monthly*, 87(2):81–90, February 1980. 17

[62] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000. 18

[63] Michael Harris. Mathematicians of the Future? Why it matters that computers could someday prove their own theorems , March 2015. Slate Magazine [Online; posted 23-March-2015]. 71

[64] J. Roger Hindley. *Basic Simple Type Theory*. Cambridge tracts in theoretical computer science, Vol. 42. Cambridge University Press, 1997. 40

[65] J. Roger Hindley. M. H. Newman's Typability Algorithm for Lambda-calculus. *Journal of Logic and Computation*, 18(2):229–238, 2008. 63

[66] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580, October 1969. 23

[67] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 3rd edition, 2006. 15

[68] John Horgan. The Death of Proof. *Scientific American*, 269:93–103, October 1993. 42, 71, 73

[69] William Howard. The Formulas-as-Types Notion of Construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, Boston, 1980. Original version circulated privately in 1969. 40, *a*

[70] Gérard Huet. Formal Structures for Computation and Deduction, 1986. Unpublished manuscript of lecture notes, downloadable from http://pauillac.inria.fr/ huet/PUBLIC/Formal_Structures.pdf. 61

[71] Michael Huth and Mark Ryan. *Logic in Computer Science – Modelling and Reasoning about Systems*. Cambridge University Press, 2004. 22

[72] Neil Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, 1999. 4

[73] Bart Jacobs. *Categorical Logic and Type Theory*. Elsevier, 1999. 10

[74] Jean H. Gallier. *Logic For Computer Science – Foundations of Automatic Theorem Proving (Second Edition)*. Dover, 2015. 22

[75] Brian W. Kernighan. Why Pascal is Not My Favorite Programming Language. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, April 1981. 24, *i*

[76] Assaf Kfoury. Translatability of Program Schemas over Restricted Interpretations. *Journal of Computer and System Sciences*, 8:387–408, June 1974. 23

[77] Assaf Kfoury. Definability by Programs in First-Order Structures. *Theoretical Computer Science*, 25(1):1–66, 1983. 62

[78] Assaf Kfoury. The Pebble Game and Logics of Programs. In L. Harrington, M. Morley, S. Simpson, and A. Scedrov, editors, *Harvey Friedman's Research on the Foundations of Mathematics*, pages 317–329. North-Holland, 1986. 62

[79] Assaf Kfoury and David M.R. Park. On the Termination of Program Schemas. *Information and Control*, pages 243–252, November 1975. 23

[80] Assaf Kfoury and Pawel Urzyczyn. Necessary and Sufficient Conditions for the Universality of Programming Formalisms. *Acta Informatica*, 22(4):347–377, October 1985. 23

[81] Muhammad Taimoor Khan and Wolfgang Schreiner. Towards the Formal Specification and Verification of Maple Programs. In *Proceedings of the 11th International Conference on Intelligent Computer Mathematics*, CICM'12, pages 231–247, Berlin, Heidelberg, 2012. Springer-Verlag. 70

[82] Muhammad Taimoor Khan and Wolfgang Schreiner. A Verification Framework for Minimaple Programs. *ACM Comm. Computer Algebra*, 47(3/4):98–99, 2013. 70

[83] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal Verification of an OS Kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM. 56

[84] A. Kock and G.E. Reyes. Doctrines in Categorical Logic. *Studies in Logic and the Foundations of Mathematics*, 90:283 – 313, 1977. Reproduced in *Handbook of Mathematical Logic* [11]. 11

[85] Dexter Kozen and Jerzy Tiuryn. Logics of Programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 14, pages 789–840. North Holland, Amsterdam, 1989. 18

[86] Daniel Kroening and Ofer Strichman. *Decision Procedures, An Algorithmic Point of View*. Texts in Theoretical Computer Science. Springer, 2008. 8

[87] Xavier Leroy. Formal Certification of a Compiler Back-End, or: Programming a Compiler with a Proof Assistant. In *33rd symposium Principles of Programming Languages*, pages 42–54. ACM Press, 2006. 55

[88] Xavier Leroy. Formal Verification of a Realistic Compiler. *Commun. ACM*, 52(7):107–115, July 2009. 55

[89] Leonid Levin. Universal Search Problems. *Problemy Peredaci Informacii*, 9(3):115–116 (in Russian), 1973. English translation in *Problems of Information Transmission*, Vol. 9, 265-266. 25

[90] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1997. 15

[91] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, 2004. 4

[92] David C. Luckham, David M.R. Park, and Michael S. Paterson. On Formalised Computer Programs. *Journal of Computer and System Sciences*, 4(3):220–249, 1970. 23

[93] Jean-Pierre Marquis and Gonzalo E. Reyes. The History of Categorical Logic. In Dov M. Gabbay, Akihiro Kanamori, and John Woods, editors, *Handbook of the History of Logic*, volume 6, pages 1–116. Elsevier (North Holland), 2012. 9, 12

[94] John McCarthy. A Basis for a Mathematical Theory of Computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North Holland, Amsterdam, 1963. 23

[95] William McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997. 43

[96] John C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996. 6

[97] Joel Moses. Macsyma: A Personal History. *Journal of Symbolic Computation*, 47(2):123–130, 2012. 65

[98] Rob Nederpelt and Herman Geuvers. *Type Theory and Formal Proof - An Introduction*. Cambridge University Press, 2014. 7

[99] David M. R. Park. A New Equivalence Notion for Communicating Systems. Abstract of lecture at the Second Workshop on the Semantics of Programming Languages, Bad Honnef, March 16-20, 1981. 35

[100] David M. R. Park. Concurrency on Automata and Infinite Sequences. In P. Deussen, editor, *Conference on Theoretical Computer Science*, Lecture Notes in Computer Science 104, pages 167–183. Springer, 1981. 35

[101] Michael S. Paterson and Carl E. Hewitt. Comparative Schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–128. ACM, June 1970. 23, 62

[102] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002. 6

[103] Benjamin C. Pierce, editor. *Advanced Topics in Types and Programming Languages*. MIT Press, 2005. 6

[104] Michael O. Rabin. Complexity of Computations. *Communications of the ACM*, 20(9):625–633, 1977. 31

[105] Michael O. Rabin and Dana S. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, April 1959. 15, 31

[106] Steve Reeves and Mike Clarke. *Logic for Computer Science*. Addison Wesley, 2003. 22

[107] John C. Reynolds. Towards a Theory of Type Structure. In B. Robinet, editor, *Programming Symposium*, Lecture Notes in Computer Science, Volume 19, pages 408–425. Springer-Verlag, April 1974. 60

[108] Davide Sangiorgi. On the Origins of Bisimulation and Coinduction. *ACM Trans. Program. Lang. Syst. (TOPLAS)*, 31(4):1–41, May 2009. 3, 35

[109] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012. 3

[110] Davide Sangiorgi and Jan Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge Tracts in Theoretical Computer Science, Vol. 52. Cambridge University Press, 2012. 3

[111] Uwe Schöning and Jacobo Toran. *The Satisfiability Problem: Algorithms and Analysis*. Mathematik fur Anwendungen. Lehmanns Fachbuchhandlung GmbH, 2013. 8

[112] Dana S. Scott. Logic and Programming Languages. *Communications of the ACM*, 20(9):634–641, 1977. 31

[113] Harold Simmons. *Derivation and Computation – Taking the Curry-Howard Correspondence Seriously*. Cambridge University Press, 2000. 41

[114] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2012. 15

[115] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism, Volume 149 (Studies in Logic and the Foundations of Mathematics)*. Elsevier Science Inc., New York, NY, USA, 2006. 41, 61

[116] Richard Statman. Number Theoretic Functions Computable by Polymorphic Programs (Extended Abstract). In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 279–282. IEEE Computer Society, 1981. 61

[117] Karl Steinbuch. Informatik: Automatische Informationsverarbeitung. *SEG-Nachrichten (Technische Mitteilungen der Standard Elektrik Gruppe)–Firmenzeitschrift*, 4:171, 1957. *b*

[118] Wolfgang Thomas. Languages, Automata, and Logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, pages 389–455. Springer-Verlag, Inc., New York, NY, USA, 1997. 35

[119] Boris A. Trakhtenbrot. A Survey of Russian Approaches to *Perebor* (Brute-Force Searches) Algorithms. *IEEE Annals of the History of Computing*, 6:384–400, 1984. 25

[120] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society. Second Series*, 42:230–265, 1936. This is the paper that introduced what is now called the *Universal Turing Machine*. See correction [**?**]. 19

[121] Moshe Y. Vardi. Automata-Theoretic Techniques for Temporal Reasoning. In Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic, Volume 3*, chapter 17, pages 971–986. Elsevier Science Inc., 2006. 16

[122] Philip Wadler. Propositions as Types. *Communications of the ACM*, 58(12):75–84, November 2015. 40

[123] Avi Wigderson. *Mathematics and Computation*. Princeton University Press, 2017. To appear. Draft available from https://www.math.ias.edu/avi/book. 19

[124] Eugene Wigner. The Unreasonable Effectiveness of Mathematics in the Natural Sciences. *Comm. in Pure and Applied Mathematics*, 13(1):1–14, February 1960. 17