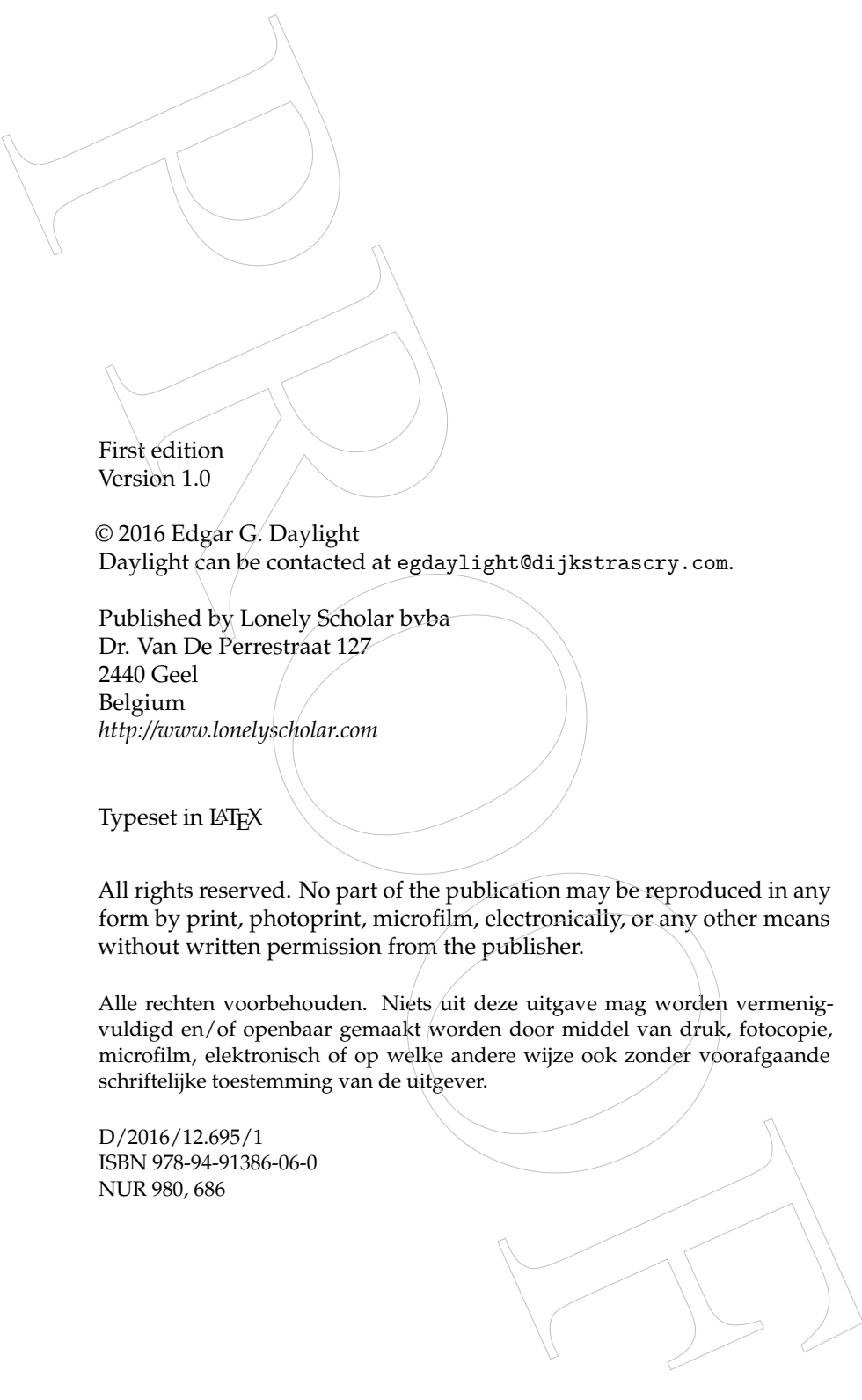


**Turing Tales**

**Edgar G. Daylight**  
**Contributions by Arthur C. Fleck and Raymond T. Boute**

Edited by Kurt De Grave.



First edition  
Version 1.0

© 2016 Edgar G. Daylight  
Daylight can be contacted at [egdaylight@dijkstrascry.com](mailto:egdaylight@dijkstrascry.com).

Published by Lonely Scholar bvba  
Dr. Van De Perrestraat 127  
2440 Geel  
Belgium  
<http://www.lonelyscholar.com>

Typeset in L<sup>A</sup>T<sub>E</sub>X

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronically, or any other means without written permission from the publisher.

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

D/2016/12.695/1  
ISBN 978-94-91386-06-0  
NUR 980, 686

# 1. Introduction

Researchers from computer science, linguistics, physics, and other disciplines have a tendency to speak about their mathematical models as if they coincide with reality. The following statement, for example, is not uncommon in physics:

Kurt Gödel proved that time travel is possible.

Careful researchers, by contrast, will stress at least once in their writings that:

Kurt Gödel proved that time travel is in principle possible *according to his mathematical model.*

More sentences of the first kind are presented in this book with regard to computer science. They show that researchers frequently fuse their mathematical models with their engineered systems.

Mistaking the category of mathematical objects for the category of concrete physical objects amounts to making a *category mistake*. Some category mistakes are less innocent than others, as Chapter 6 — A Titanic Turing Tale — will reveal. For example, it is easy to find computer science papers in which the authors explicitly and incorrectly state that they have mathematically *proved* that certain *systems* cannot be *engineered*. While many people would like computer science *theory* to take precedence over software *engineering*, nobody wants this ideal if the price to pay is faulty reasoning. Rectifications are required and as a brief indication of where I am heading, Sections 1.2 and 1.3 in the present introduction contain my scrutiny of statements made by Michael Hicks and referees of computer science's flagship journal, the Communications of the ACM.

Category mistakes are discussed at length in the second part of this book, while the first part reveals another tendency in computer science: that of constructing a set of foundations in which ideas that had not been

understood as connected when they were put forward are retrospectively integrated. Later generations of computer scientists then assume that these things have always been related. The connection between universal Turing machines and computers is one notable example. Many computer scientists who have given invited talks about the history of their discipline have made statements of the following kind:

During the 1950s, a universal Turing machine became widely accepted as a conceptual abstraction of a computer.

Other scholars, by contrast, will refrain from making such general and appealing claims. Instead, they will focus on specific actors and write the following, for example:

By 1955, Saul Gorn viewed a universal Turing machine as a conceptual abstraction of his computer.

The subject of the first sentence is not a historical actor, unlike the subject of the second sentence. If you happen to prefer the first sentence and tend to write sentences of this kind, then it is likely that your exposition will, at best, capture a development of ideas that is detached from the people who shaped the discipline under investigation. As a result, neither you nor your readership will realize that a universal Turing machine had different meanings for different actors, nor will it become apparent that the meaning of a Turing machine changed over time for each individual actor.

We will learn more about Saul Gorn and Turing machines in Chapter 2: A Tatty Turing Tale. The word “tatty” is a synonym for “worn out” and it is indeed a worn-out tale — at least for historians today — that Turing supposedly invented the modern computer. It is likely that you know somebody who proclaims that Turing is the inventor of the computer because his 1936 *theory* was a prerequisite for later breakthroughs in *engineering*. Chapter 2 reveals that this romantic view of scientific progress is misleading to say the least. Chapter 3 illustrates that there is a Dutch exception to the rule.<sup>1</sup>

The overarching theme of the present book, then, is the relationship between theory and engineering. Two specific topics are the historiographical mistakes and category mistakes made by researchers of standing in their quest for a utopian world, a world in which the role of mathematics in engineering is intended to be that of a queen rather than a humble servant.<sup>2</sup>

My meta-challenge is to convince the reader that thorough historical and philosophical investigations into computer science can advance computer science itself. As I will argue in this chapter, there is a marked difference between the computer scientist of today and the computer scientist of tomorrow.

## 1.1 John Reynolds and Turing's 1936 Paper

In order to adhere to academic standards, Gerard Alberts, a historian of mathematics and computing, has advised me to avoid using technological and theoretical concepts such as 'program,' 'compiler,' and 'universal Turing machine,' as the subjects of my sentences. Disregarding his advice often amounts to writing expositions in which one line of thought dominates the entire story. For example, if I choose as a subject matter Turing's 1936 paper [320], or more specifically, the connection between Turing's 1936 universal machine and modern computers, then it becomes very tempting to view the history of computing in terms of those few people who grasped that connection early on. Turing, for example, already saw a connection around 1944. This observation alone, however, does not make him an influential actor in the history of science and technology. By following Alberts's advice, the scholar loses the inclination and the temptation to paint the history of computing as a continuous stream from Turing's 1936 paper to the stored-program computer. The scholar will fail in explaining every advancement in terms of Turing machines, and this is to the good.

The late Michael Mahoney warned his fellow historians not to fall into the trap of viewing everything with the same glasses. The computer, Mahoney said specifically, is not one thing but many different things. He continued as follows:

[T]he same holds true of computing. There is about both terms a descriptive singularity to which we fall victim when, as is now common, we prematurely unite its multiple historical sources into a single stream, treating Charles Babbage's analytical engine and George Boole's algebra of thought as if they were conceptually related by something other than twentieth-century hindsight. [234, p.25-26]

In my own words, then, the multitude of computer-building styles, programming habits, receptions of Turing's work, interpretations of the terms "recursion" and "computer program," and so on should be placed

front and center by computer scientists and historians alike. Terms such as “general purpose” and “Turing universal” had different meanings for different actors, and their usage as synonyms is only justified if this conforms with the historical context. I take Mahoney’s challenge to mean that we need to handle each and every term with great care.

Unfortunately, Mahoney “never took up his own invitation,” says Thomas Haigh in the introduction of Mahoney’s collected works [234, p.5,8]. Men like John McCarthy, Dana Scott, Christopher Strachey, Turing, and John von Neumann appear in almost every chapter in Mahoney’s collected works, but, as Haigh continued,

[W]e never really learn who these people are, how their ideas were formed and around what kind of practice, what their broader agendas were, or whether anything they said had a direct influence on the subsequent development of programming work. [234, p.8]

Mahoney did occasionally provide some insights about the aforementioned men. Coincidentally or not, the rare passages in which he did also comply with the writing style advocated by Alberts, as the following excerpt from Mahoney’s oeuvre illustrates.

Christopher Strachey had learned about the [lambda]-calculus from Roger Penrose in 1958 and had engaged Peter J. Landin to look into its application to formal semantics. [234, p.172]

Words such as these inform the reader about several historical actors, their social networks, and their research objectives.

In general, however, Mahoney much preferred technological concepts over historical actors.<sup>3</sup> One instructive example comes from his 2002 work, in which he stated that:

The idea of *programs that write programs* is inherent in the concept of the universal Turing machine, set forth by Alan M. Turing in 1936. [234, p.78-79, my emphasis]

This statement, which has an idea as subject matter, is anachronistic. Turing’s 1936 paper was not at all about programs that write programs. Turing’s universal machine did not modify its own instructions, nor did it modify the instructions of the other machines that it simulated.

Instead of alluding to compilers, as Mahoney did, one could perhaps refer to interpreters when attempting to document the importance of

Turing's 1936 paper. After all, interpreters emulate programs — loosely speaking. The following sentence is technically more accurate:

[Turing's] universal machine in particular is the first example of an interpretative program. [88, p.165]

These words come from the eminent logician Martin Davis who has made great strides in explaining to the layman the importance of logic in computer science. The subject in Davis's sentence is Turing's universal machine, not a historical actor. From Alberts's perspective, then, it is tempting to rewrite, and in my opinion improve, the wording. My first suggestion is to write:

During the early 1960s, John McCarthy viewed a universal Turing machine as an interpretative program.

This sentence limits the scope of Turing's influence by focusing on McCarthy and the years in which he thought about interpretative programs (in the modern sense or in *a* more modern sense of the word).

My second, more elaborate, suggestion is to write:

In the 1950s, when interpreters were built, leading computer programmers like McCarthy did not initially view Turing's universal machine as an interpretative program in a practical sense. Although McCarthy had by 1960 already written a paper [238] in which he had connected the universal Turing machine to his LISP programming system, he did not see the practical implication of LISP's universal function *eval*. It was his student Steve Russell who insisted on implementing it. After having done so, an interpreter for LISP "surprisingly" and "suddenly appeared".<sup>4</sup>

The previous fragment informs the reader about McCarthy and his research team, partially capturing how Turing's 1936 universal machine eventually led to McCarthy's interpreter for LISP. The passage does not disregard the possibility that others had already made some kind of a connection between a universal Turing machine and programming technology in earlier years. However, if one wants to make a general claim about Turing's legacy, then he or she should first do the hard work of finding several specific actors for which the claim holds.

Taken at face value then, Mahoney's oeuvre gives the false impression that several influential historical actors, along with himself, thoroughly

understood Turing's 1936 paper and the logic literature in general. "Computer science," Mahoney postulated in 1997, "had come around full circle to the mathematical logic in which it had originated" [234, p.144]. However, this claim does not sit well with his own appeal to avoid falling into the trap of viewing everything with the same — and in this case, logical — glasses time and again. Nor does it sit well with several primary sources and oral histories. For example, according to an expert in Separation Logic, the late John Reynolds, understanding the logical literature is "taxing."

[Reynolds:] I'd better admit that I haven't read Turing's 1936 paper. I probably avoid old papers less than most computer scientists, but I wasn't trained in logic and thus find the subject taxing to read (indeed, more to read than to write). [288]

Men of the stature of Tony Hoare and Peter Naur had difficulty studying Turing's 1936 paper [98]. In addition, Davis's remark at the end of Christos Papadimitriou's talk at Princeton in 2012 clearly shows that prominent computer scientists, such as Papadimitriou, still do not fully comprehend Turing's 1936 paper either [270].

In his monumental book *A Science of Operations* [280], Mark Priestley documented the interaction between theory and practice in the development of computing, starting from the early work of Charles Babbage and ending with the programming language `Smalltalk`, thereby providing a coverage that has yet to be matched by fellow historians. Based on his work and on some of my own research, I now list seven influential publications of the 1950s-1960s. Each publication played an important role in transferring ideas from logic to computing.

- 1950: Turing's *Computing machinery and intelligence* [322]
- 1950: Paul Rosenbloom's *Elements of Mathematical Logic* [294]
- 1952: Stephen Kleene's *Introduction to Metamathematics* [202]
- 1954: Andrey Markov Jr.'s *Theory of Algorithms* [197]
- 1958: Davis's *Computability and Unsolvability* [86]
- 1958: Haskell Curry & Robert Feys's *Combinatory Logic, Vol. 1* [78]
- ...



- 1967: Marvin Minsky's *Computation: Finite and Infinite Machines* [245]

The influence exerted by most of the publications listed above has yet to be thoroughly investigated in future work. Turing's scholarly legacy in computer science, not to mention that of Emil Post, Alonzo Church and others, has yet to be described.

Priestley has discussed the influence of the first publication at length in 'Logic and the Invention of the Computer' [280, Ch.6]. Priestley argued that Turing had little influence in the 1940s (and in computer building in particular) but that his 1950 paper, 'Computing machinery and intelligence,' was "a turning point in the characterization of the computer as a universal machine" [280, p.153]. "After 1950," he wrote, "it became common to describe electronic digital computers as being instantiations of Turing's concept of a universal machine" [280, p.124]. Furthermore, he writes:

Following 1950, [Turing's 1950] paper was widely cited, and his characterization accepted and put into circulation. [280, p.153]

These words by Priestley give the unfounded impression that most computer practitioners became acquainted with Turing's work during the 1950s. If I now name people who did not grasp, read, or come across Turing's work, Priestley's readership will think that these are exceptions. By the late 1950s, Turing's work had indeed become increasingly popular in some niches of computing (see Chapters 2 and 3). However, if anything at all can be stated about the majority of computer practitioners, then it is that they either did not yet understand the all-purpose nature of the computer, or they did but *without* having resorted to (a re-cast version of) Turing's work [98]. Ideally, and following Alberts, neither Priestley nor I should be making claims about the majority of computer practitioners in the first place. Instead of taking Turing's 1950 paper as a subject, as Priestley has done in the above passage, a historical actor or a well-defined group of historical actors could be chosen instead, thereby limiting the scope of Priestley's claim regarding Turing's influence.<sup>5</sup>

To conclude, then, I urge computer scientists and historians to respect the various receptions of Turing's work. Furthermore, let us view our past not solely as an application of Turing's 1936 paper but also as a history of struggling to understand what Turing's 1936 paper has to offer to some, perhaps many, but definitely not all of us. This brings me to my final introductory remarks about Chapters 2 and 3, chapters which

I wrote with the previously explained methodology and motivation in mind. Both chapters cover the 1950s and the role that Turing's 1936 paper played in the emerging discipline now called computer science. Chapter 2 focuses on Anglo-Saxon developments and Chapter 3 zooms in on the Dutch cities Delft, The Hague, and Amsterdam. It was in Delft and later in The Hague where Turing's 1936 paper *did* influence computer *building* to an exceptional extent, both on an academic level and in an industrial European-wide context due to the extraordinary work of Willem van der Poel.

My main interests in Chapters 2 and 3 are, again, the relationship between modern logic and engineering and the way in which this relationship is erroneously portrayed today, albeit often unintentionally, in order to advance a particular research agenda.

## 1.2 Michael Hicks and Formal Verification

Viewing a universal Turing machine and a stored-program computer as one and the same amounts to making a trivial, yet common, category mistake. Jack Copeland is perhaps the most renowned scholar who insists that Alan Turing came up with "the stored-program universal computer" as a "single invention" in 1936. Andrew Hodges, Martin Davis, and others have, in turn, complained about Copeland's viewpoint and I suspect that Turing would do the same if he were still alive today.<sup>6</sup>

Why would Turing, of all people, mistakenly view his mathematical *model* of computation, now called a Turing machine, as a concrete physical object such as a stored-program computer? A Turing machine can be practically realized in multiple ways, and a stored-program computer is just one of them. Furthermore, and as I will highlight repeatedly, a Turing machine is only one possible model of a computer.

Category mistakes are not limited to digital systems, however. Engineers in all areas use models and can potentially fall into the trap of mistaking the model for reality. Dave Parnas told me that one of his favorite examples in this regard is Ohm's Law, which states that  $V = I \times R$ . In reality, the ratio between current ( $I$ ) and voltage ( $V$ ) varies, i.e., the relationship is not actually linear. Engineers can use this law to analyze a circuit and verify that it has the desired behavior only to find that the circuit fails in certain conditions. Parnas gives the following example:

The circuit may overheat and the resistance ( $R$ ) will change.  
Ohm's Law can be made an accurate description of the device

by adding conditions, e.g.  $V - I \times R < \delta$  if  $a < I < b$  etc. The result will be much more complex, but it will give conditions under which the model is accurate. Even this is a lie, as it makes assumptions about ambient temperature and the aging of the components that are not stated.<sup>7</sup>

Parnas's main point is that engineers must be aware of the limitations of models as descriptions. More on Parnas's views can be found in the proceedings of *The Future of Software Engineering*, a workshop held in ETH Zurich in November 2010 [103, 255]. Parnas concludes by noting that this awareness is completely missing from the UML (Unified Modeling Language) and MDE (Model-Driven Engineering) literature. Can the same be said about the programming language literature? The full answer to this question is given in Chapter 6.

Of course, one could argue that researchers frequently choose to craft their sentences with brevity in mind and that they are well aware of the fact that they are, strictly speaking, making small category mistakes; surely, they must know that these are models. This is precisely the kind of response I received from a POPL referee a few months ago with regard to my rejected paper, entitled: 'Category Mistakes in Computer Science at Large.' (POPL is an abbreviation for Principles of Programming Languages and the contents of the aforementioned paper constitute Chapter 6.) When I gave this kind of response to the colleague who brought to my attention the statement about Gödel and time travel, he insisted that some physicists really do believe that Gödel proved that time travel is possible. As this book will reveal to the neutral scholar, the same can be said about a large number of computer scientists.

Similar remarks were made by the mathematician Vladimir I. Arnold in 1997:

The mathematical technique of modeling consists of [...] speaking about your deductive model in such a way as if it coincided with reality. The fact that this path, which is obviously incorrect from the point of view of natural science, often leads to useful results in physics is called "the inconceivable effectiveness of mathematics in natural sciences" (or "the Wigner principle"). [15, my emphasis]

The mathematical technique of modeling is undoubtedly useful in practice. However, society will fare even better when the vices of modeling are also placed front and center every now and then.

Coming to formal verification, then, several prominent computer scientists — some of whom I have met and talked to many times — actually believe the following statement:

... full formal verification, the end result of which is a proof that the code will always behave as it should. Such an approach is being increasingly viewed as viable. [176]

These words come from Michael Hicks, who said in 2014 that we can have a mathematical *proof* of the behavior of an engineered *system*. The best we can actually do — as already pointed out by Brian Cantwell Smith [303], James Fetzer [125], and Timothy Colburn [72] in previous decades and in a technically more refined way in the present book — is to prove certain *mathematical properties* of one or more *mathematical models* of the running *system*. This small nuance will presumably be readily accepted by many readers, but I hope to bring more conceptual clarity to the table in the sequel.<sup>8</sup>

## Rice's Theorem Under Attack

Let us consider some of the alleged practical implications of Rice's theorem. Here, my own thoughts become more visible, since the writings of Fetzer and the other two aforementioned philosophers do not cover computability theory.

For the uninitiated, Figure 1.1 presents Wikipedia's description of Rice's theorem. For both the uninitiated and the initiated, Figure 1.1 comes very close to illustrating a category mistake, and therefore exemplifies a lack of conceptual clarity that I hope to remedy in the present book. I will also return to Figure 1.1 in the final chapter.

Stepping away from Wikipedia for now and coming to Hicks's discussion on software bugs and Rice's theorem, this is how Hicks defines a "sound" mathematical analysis:

A *sound* analysis is one that, if there exists an execution that manifests a bug at run-time, then the analysis will report the bug. [176]

Note that the first occurrence of the word "bug" refers to an event in the real, physical world. The second instance of the same word refers to a mathematical model of a bug (and only indirectly to the manifested bug in the real world). For the sake of clarity — and at the expense of some

In computability theory, Rice's theorem states that all non-trivial, semantic properties of programs are undecidable. A semantic property is one about the program's behavior (for instance, does the program terminate for all inputs), unlike a syntactic property (for instance, does the program contain an if-then-else statement). A property is non-trivial if it is neither true for every program, nor for no program.

We can also put Rice's theorem in terms of functions: for any non-trivial property of partial functions, no general and effective method can decide whether an algorithm computes a partial function with that property. Here, a property of partial functions is called trivial if it holds for all partial computable functions or for none, and an effective decision method is called general if it decides correctly for every algorithm.

Figure 1.1: Wikipedia's description of Rice's theorem portrays rather accurately how it is taught in academia today in the sense that the first paragraph comes very close to illustrating a category mistake. (How close depends on how careless one answers the question: What, precisely, is a program?) This mistake is discussed at length in the present book. The second paragraph is less problematic but is not very accessible to the average computer scientist for it requires prior knowledge of partially computable functions and computability theory in general. For the record: I accessed the Wikipedia site on October 26, 2016.

brevity — it is worthwhile to make this categorical distinction explicit *at least once*, as I am doing here and will do throughout most of this book.

Another important remark is that if the first occurrence of “bug” indeed refers to an event in the real world, then so does the word “execution.” However, the mathematical analysis has to rely on a mathematical notion of execution. A distinction is therefore required between a ‘real execution’ and a ‘mathematical execution.’ The former belongs to the category of concrete physical objects, while the latter belongs to the category of abstract objects.

Furthermore, Hicks's ‘mathematical execution’ denotes a mathematical object in conformance with his chosen model of computation. Somebody else — following, say, Edsger Dijkstra's or Parnas's schools of thought, described below — might choose *another* model of computation. So it would be clearer if Hicks would also make his chosen model of computation explicit in his definition.

In other words, a sound analysis says something about a mathematical model and only *indirectly* something about the computer program that is being modeled. Furthermore, since someone else can choose another model for the same computer program, **it is misleading to suggest that there is a one-to-one interdependence between the computer program and the chosen mathematical model.**

Similar remarks can be made about Hicks's definition of a "complete" analysis, which is as follows:

On the flip side, a *complete* analysis is one that, if it reports a bug, then that bug will surely manifest at run-time. [176]

Again, a mathematically modeled bug (which the analysis "reports") is categorically distinct from a bug encountered during the "run-time" execution of a computer program. They are not the same thing.

Am I right to conclude that Hicks and other programming language specialists often think they are *directly* referring to *both* their mathematical model *and* the actual computer program? (Chapters 5 and 6 will delve into these matters further.) Later on, Hicks provides the following statement, which I, as a 'POPL outsider,' have genuine difficulty in comprehending:

Ideally, we would have an analysis that is both sound and complete [Yes], so that it reports all true bugs, and nothing else [No!]. [176]

This statement can only be correct if the category of abstract objects coincides with the category of concrete physical objects. However, since a bug encountered during program execution is categorically different from a mathematical bug, what does Hicks's sound and complete analysis *actually* report? Does it report

1. bugs manifested during program execution,
2. mathematical bugs, or
3. both?

Again, I claim that the correct answer is 2. Hicks's analysis reports about mathematical bugs, that is, mathematically modeled bugs. Does Hicks think the answer is 3., in the sense that both categories coincide? Or am I wrong to question whether there is a one-to-one interdependence

between the computer program and a well-chosen mathematical model of the computer program?

To recapitulate, at least as far as my understanding goes: a perfect *mathematical* analysis (in the sense that it is both sound and complete) cannot *guarantee* something about the *real* world, including the behavior of the engineered *system* under scrutiny along with the bugs that manifest themselves in the process. This would only be possible if the mathematical object and the engineered system belong to the same category (and, *moreover*, if we can specify absolutely everything about the engineered system in a concise and useful manner). A sound and complete analysis can provide engineers with *extra confidence* that their system will behave appropriately in the real world and *nothing more*.

To continue with Hicks's views on formal verification, it should also be mentioned at this point that, contrary to Hicks, computer scientists following Dijkstra and Adriaan van Wijngaarden would mathematically model a computer program — such as a C computer program — with a Turing-incomplete model of computation and, specifically, with a finite state machine.<sup>9</sup> I therefore *also* struggle with what I take to be Hick's subsequent suggestion, which is only to use Turing-complete languages when mathematically modeling computer programs:

Unfortunately, such an [ideal] analysis [which is both sound and complete] is impossible for most properties of interest, such as whether a buffer is overrun (the root issue of Heartbleed). This impossibility is a consequence of Rice's theorem, which states that proving nontrivial properties of programs in Turing-complete languages is undecidable. So we will always be stuck dealing with either unsoundness or incompleteness. [176]

I certainly agree with Hicks that *if* we use a Turing-complete language, *then* we cannot ignore an important consequence of Rice's theorem, namely that "proving nontrivial properties" of our mathematically modeled computer programs (expressed in our Turing-complete language) "is undecidable." However, engineers such as Parnas resort to multiple models to describe buffer overflows (pertaining to, say, a C computer program), including models that are described with a Turing *incomplete* language. The bottom line is that engineers do not have a preference for sticking to a single modeling language, nor do they advocate a Turing-complete language *per se*.

Contrary to many (if not most) programming language specialists, engineers do not want to attach *precisely one meaning* to each computer

program. An engineered system can be mathematically modeled in more than one way; each model has its pros and cons. I can model my computer with both a finite state machine and a linear bounded automaton without contradicting myself. Likewise, I can mathematically model my C computer program in multiple, complementary ways, for example with a finite state machine, with primitive recursive functions, and with general recursive functions. The richness lies in the multitude of ways in which reality can be mathematically modeled, and I hope to convey this richness in the remainder of this book.

### 1.3 Abusing the Halting Problem

The analysis presented so far shows that, at the very least, a categorical distinction is required between computer programs and mathematical programs. The term “mathematical program” is used from now on as an abbreviation for “a mathematical model of a computer program.”

Another obvious distinction that is worth making explicit at least once is the distinction between computers (which include laptops and iPads) on the one hand and their mathematical models on the other hand. Strictly speaking, then, it is wrong to say that:

A computer is a finite state machine.

Once again, this is like speaking about a mathematical model (the finite state machine) as if it coincides with reality (the computer). But making this observation explicit in computer science, as I am doing now and as I have done in my rejected paper entitled ‘Category Mistakes in Computer Science at Large,’ seems to be rather unusual.

My paper on Category Mistakes was rejected by two POPL referees for some very good reasons, albeit non-technical ones. I shall have more to say about the reviews I received in Chapter 5. For now, it is important to note that my rejected POPL paper contains quoted reviews from anonymous referees of the Communications of the ACM (CACM). With regard to these CACM review comments, here is what the first POPL reviewer had to say:

I’m not sure why CACM reviewers would ignore the difference between real-world systems and their mathematical models. I don’t actually see that mistake in the quoted reviews [in your paper].



I am afraid that the reviewers of the CACM have applied faulty reasoning, and I have illustrated precisely this in my POPL paper. I shall illustrate some of this faulty reasoning in the next section, leaving the rest for Chapter 6.

## Computers vs. Mathematical Models

Strictly speaking, a computer is not a finite state machine. The former is a *concrete physical* object which can be mathematically modeled by the latter, which is an *abstract* object. Here then is the first comment that I have received from a referee of the CACM:

My laptop is a universal Turing machine, but its tape size is of course limited by the finiteness of human resources.

If you limit the tape size of a universal Turing machine, you may end up with, say, a linear bounded automaton or even an automaton that is computationally equivalent to a finite state machine. You thus end up with another *mathematical* model of computation but *not* with a laptop (i.e., a concrete physical object). To be more precise, I stress that:

You cannot use *human* resources to limit the size of a *mathematical* object, i.e., the tape. Note that the “tape” indeed denotes a mathematical object and not a physical object, contrary to what the word “tape” seems to suggest.

You can introduce mathematical restrictions to limit the size of a mathematical object; likewise, you can use human resources to limit the size of a concrete physical object, such as a laptop. However, once again:

A Turing machine *is* a mathematical object, it is not a computer. This is contrary to what the word “machine” seems to suggest.

I understand the CACM reviewer’s train of thought. I, too, was educated as a computer scientist and I used to speak about my mathematical model as if it coincided with reality. The right way to put it, once again, is as follows:

Placing finite bounds on an abstract object (Turing machine) does not make it a concrete physical object (laptop). Instead,

it results in another abstract object (e.g., a linear bounded automaton or a finite state machine) that can potentially serve as another mathematical model for the physical object at hand.

I agree that these words convey a very trivial distinction. However, missing this distinction can easily lead to faulty reasoning. Only a mathematical language can be Turing complete; it thus makes no sense to question whether your iPhone is Turing universal or not (as, for instance, did almost all my computer science students at Utrecht University in 2015). Unfortunately, statements of this kind can be found all over the place, not only in peer reviews but also in articles and in books, published by reputable publishers. I have even had discussions with colleagues who start proving on the blackboard and in the classroom that my laptop *is* a universal Turing machine after all. They really think they are giving a *mathematical* proof about my laptop. I emphasize, once again, that:

It is a mathematical model of a laptop that may or may not be Turing universal, not the laptop itself. Yet, anonymous referees of computer science's flagship journal, the Communications of the ACM, disagree with this statement and erroneously place both objects in the same category. This is where a seemingly innocent category mistake occurs.

Comparing a laptop with a Turing machine is only warranted with the proviso that we all agree we are reasoning across separate categories.

## A Big Category Mistake

Grasping the significance of seemingly obvious categorical distinctions is not easy; here is yet another response that I have received from a referee of the CACM and which I have also reported in my rejected POPL paper:

What does the undecidability proof of the halting problem for computer programs actually tell us? Like diagonalization proofs in general it may be viewed finitely as saying that, if there is a bound  $M$  on the size of accessible computer memory, or on the size of computer programs, or any other resource, then no computer program subject to the same resource bounds can solve the problem for all such computer programs.

The previous remark and the follow-up remark, presented below, are only correct if we accept the following two assumptions (both of which are wrong):

1. A computer program is a synonym for a mathematical program.
2. The mathematical program (mentioned in the previous sentence) must be equivalent to a Turing machine program and not to, say, a primitive recursive function.

The reason why the second assumption has to hold is merely because the referee is referring to the halting problem of Turing machines. Continuing with the remarks made by the anonymous referee:

If computer program  $A$  solves correctly all halting problems for computer programs respecting bound  $M$ , then the counterexample computer program  $T$  must exceed that bound, which is why  $A$  fails for  $T$ . To solve problems of computer programs, one needs an ideal program.

This quote hints at a distinction that must be made between finite and infinite objects (with the latter being labeled “ideal”); however, the categorical distinction between computer programs and mathematical programs goes completely unnoticed. This is where a big category mistake occurs. The undecidability proof of the halting problem concerns *mathematical* programs only and not *computer* programs. The diagonal argument can only be applied to mathematical objects, not engineered artefacts. Thus, while the referee thinks this is a mathematical argument, in fact this is faulty reasoning. The referee is *not* proving something about *computer* programs but something about a *\*particular\** mathematical model of a computer program! So much for mathematical rigor.<sup>10</sup>

## 1.4 Pluralism to the Rescue

Based on the above analysis it is now possible to provocatively define both a computer scientist of today and one of tomorrow. A computer scientist of today is somebody who conflates:

- a Turing machine and a computer,
- a Turing tape and a physical tape,

- a mathematically modeled bug and a bug encountered during program execution, and
- a mathematical object and a computer program.

Moreover, the mathematical object denoted in the last item must be a Turing machine or some object computationally equivalent to it. (I realize that programming language experts do not work with Turing machines *per se*; however, this is rather beside the point.)

The computer scientist of tomorrow, by contrast, is sensitive to the aforementioned categorical distinctions and, furthermore, is receptive to the *multitude* of answers to the seemingly simple question:

What is a computer program?

Today, we know that the “computer program” concept has acquired at least four meanings during the course of history. It can refer to:

1. a physical object à la Maurice Wilkes in 1950 and Dave Parnas in 2012,
2. a mathematical object of finite capacity à la Edsger Dijkstra in 1973,
3. a mathematical (Turing-machine) object of infinite size à la Christopher Strachey in 1973, and
4. a model of the real world that is not a logico-mathematical construction à la Peter Naur in 1985 and Michael Jackson today.<sup>11</sup>

Moreover, wearing my philosophical hat, I will follow Raymond Turner’s recent analysis [325] and view a computer program as a technical artefact.<sup>12</sup> I shall define and use this fifth interpretation of what a computer program entails in Chapter 6.

The multitude of interpretations of what a computer program entails is an example of epistemic pluralism. In addition, computer scientists have not consistently followed a single interpretation of a computer program in their writings. Marvin Minsky’s 1967 book, *Computation: Finite and Infinite Machines* [245], for example, uses the word “program” on page 25 to refer to data and instructions that fit in the real, finite memory of a physical computer (that is, as a physical object). On page 153, by contrast, the very same word refers to a mathematical object of “unlimited storage capacity,” akin to a Turing machine. Likewise, Tony Hoare consistently used the word “computer” in 1969 to refer to a real

physical device, while in his 1972 paper 'Incomputability' [181] the very same word sometimes refers to a *finite*, physical device and sometimes to a mathematical abstraction in which "*infinite* computations" can arise.

In sum, historical actors have not always explicitly distinguished between real artefacts and their models, let alone between all the aforementioned meanings of a "computer program." In the words of Bernadette Bensaude-Vincent, "epistemic pluralism is a major feature of emerging fields" [332] and I hope my readers will come to appreciate that computer science is still too young a field to be any different.

## 1.5 Arthur Fleck's Reflections & Raymond Boute's Scrutiny

The two cherries on the cake are Chapter 4 and Chapter 7, written by Arthur Fleck and Raymond Boute, respectively. When a software scholar receives personal reflections and detailed scrutiny from two historical actors, along with the implicit question about whether and where their narratives can be published, the scholar experiences one of his finest days.

Arthur Fleck is former chairman of the Computer Science Department of the University of Iowa. His personal and technical recollections on programming language history — on FORTRAN, ALGOL 60, EULER, APL, SNOBOL4, Smalltalk-80, FP, Miranda, and Prolog — are an absolute delight to read and I am confident that fellow historians will buy this book for Chapter 4 alone.

Raymond Boute is professor emeritus in formal methods from INTEC Department of Information Technology, Ghent University. He questions the understanding of basic concepts and the *function* concept in particular in Chapter 7. Complementary to my focus on the informal aspects of computer science, Boute advocates elementary use of symbolism to support the textual definitions at hand. I believe next generations of formal methodists can benefit greatly from reading Chapter 7.

# Endnotes

<sup>1</sup>I wrote most of Chapters 2 and 3 a few years ago and the careful scholar will observe that some statements made in the endnotes exemplify category mistakes. I think it is instructive to keep these mistakes in the first edition of the present book.

<sup>2</sup>Paraphrasing Michael Jackson [193, p. 51].

<sup>3</sup>See my blog post on Michael Mahoney [99].

<sup>4</sup>See McCarthy's recollections [239, p.191] and Herbert Stoyan's historiography [308].

<sup>5</sup>It should also be noted that the Turing machine model of computation has been over sold by computer scientists [35] and historians [99] alike. But only in future work will I attempt to rectify this situation in a more direct manner; e.g., by focusing on the legacy of Alonzo Church and the topic of types in programming languages [237], and e.g., by scrutinizing the relationship between reasoning and computing [116].

It should perhaps also be remarked that the technicalities in the present edition of this book make it less accessible for some fellow historians, which I regret and hope to remedy in the future.

<sup>6</sup>Sources: Copeland [76, p.3], Davis [90, p.147,166], and Hodges [183, p.1994]. For a further discussion, see the present author's paper 'A Turing Tale' [101].

<sup>7</sup>Paraphrasing Parnas, based on private correspondence with the present author in early August 2016.

<sup>8</sup>I have written that "several prominent computer scientists...

actually believe . . .” This statement is based on my own research and on the many examples provided in the writings of Timothy Colburn [72] and Donald MacKenzie [233].

<sup>9</sup>Details are provided in Chapter 6 along with the following definition: a mathematical language is *Turing complete* when it is able to express all partially computable functions [333, p.539].

<sup>10</sup>Three remarks are in order. First, to be more precise, the diagonal argument can be applied only to mathematical objects *if* one wants to maintain the claim that one is using the argument in a mathematical proof.

Second, no harm is done in hijacking the term “computer program” and using it as a synonym for a “mathematical program,” but only as long as one does so (a) *consistently* and (b) without claiming that the obtained result is an absolute claim about *technology*.

Third, and for the sociological record only, it should be noted that the reviews received from the CACM are much more elaborate than those from POPL. The CACM reviews convey more how each reviewer thinks and are several pages long. In retrospect, this is perhaps to be expected; the CACM is after all a journal, while POPL is an annual conference with a well-defined scope of research.

<sup>11</sup>Two remarks. First, Strachey was passionate about the lambda calculus rather than Turing machines, but this observation is less relevant for the purpose of the present book. Second, I refer to my oral histories with Naur and Jackson for more coverage on their views [96, 193].

<sup>12</sup>I consistently follow Turner in writing “artefact” (British English) instead of “artifact” (American English) even though the rest of this book is written in American English.

<sup>13</sup>I am particularly grateful to Thomas Haigh for his written and oral feedback on multiple drafts of this chapter, starting in the spring of 2013. I also thank Nancy R. Miller and J.M. Duffin of the University of Pennsylvania Archives and Jos Baeten of the ‘Centrum voor Wiskunde & Informatica’ for funding my visit to the Archives.

<sup>14</sup>The minutes of that meeting state:

Bright reported that the Program Committee recommends that the National ACM Lecture be named the Allen [sic] M. Turing Lecture.

# Bibliography

- [1] *ACM Council Meeting* (1965). Available from the “Saul Gorn Papers”, the University of Pennsylvania Archives (unprocessed collection).
- [2] *ACM Council Meeting* (1966). Available from the “Saul Gorn Papers”, the University of Pennsylvania Archives (unprocessed collection).
- [3] *ACM Council Meeting* (1966). Available from the “Saul Gorn Papers”, the University of Pennsylvania Archives (unprocessed collection).
- [4] A. Aker. “Anthony Oettinger interview: January 10–11, 2006”. In: *ACM Oral History interviews*. 2006.
- [5] A. Aker. *Calculating a Natural World: Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research*. Cambridge, Massachusetts, USA: MIT Press, 2007.
- [6] G. Alberts. “Jaren van berekening”. PhD thesis. Universiteit van Amsterdam, 1998.
- [7] G. Alberts and H.T. de Beer. “De AERA. Gedroomde machines en de praktijk van het rekenwerk aan het Mathematisch Centrum te Amsterdam”. In: *Studium* 2 (2008), pp. 101–127.
- [8] G. Alberts and E.G. Daylight. “Universality versus Locality: the Amsterdam Style of ALGOL Implementation”. In: *IEEE Annals of the History of Computing* 4 (2014), pp. 52–63.
- [9] P. Alevoor, P. Saeda, and K. Kapoor. “On the decidability and matching issues for regex languages”. In: *International Conference on Advances in Computing*. Ed. by M.A. Kumar, R. Selvarani, and T.V.S. Kumar. Springer Verlag, 2012, pp. 137–145.
- [10] F.L. Alt. *Electronic Digital Computers: Their Use in Science and Engineering*. New York, NY, USA: Academic Press, 1958.
- [11] F.L. Alt. “Archaeology of Computers — Reminiscences, 1945–1947”. In: *Communications of the ACM* 15.7 (1972), pp. 693–694.
- [12] T.M. Apostol. *Calculus, Vol. I (2nd. ed.)* Wiley, 1967.



- [13] A. Appel. The science of deep specification (<http://deepspec.org/research/>). Accessed on September 25th, 2016.
- [14] A. Appel. *Turing, Gödel, and Church at Princeton in the 1930s*. YouTube: [www.youtube.com/watch?v=kO-8RteMwfw](http://www.youtube.com/watch?v=kO-8RteMwfw). Presentation at the Turing Centennial Celebration at Princeton, 10–12 May 2012.
- [15] V.I. Arnold. *On Teaching Mathematics*. This is an extended text of the address at the discussion on teaching of mathematics in Palais de Decouverte in Paris on 7 March 1997 (<http://pauli.uni-muenster.de/~munsteg/arnold.html>). 1997.
- [16] J. Backus. “Can programming be liberated from the von Neumann style?” In: *Communications of the ACM* 21.8 (1978), pp. 613–641.
- [17] J. Backus. “The History of FORTRAN I, II, and III”. In: *ACM SIGPLAN Notices* 13 (1978), pp. 165–180.
- [18] J. Backus, J. Williams, and E. Wimmers. “An introduction to the programming language FL”. In: *Research Topics in Functional Programming*. Ed. by D.A. Turner. Addison-Wesley, 1990, pp. 219–247.
- [19] J.W. Backus. “The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM Conference”. In: *IFIP Congress*. UNESCO, Paris. 1959, pp. 120–125.
- [20] J.W. Backus, F.L. Bauer, J. Green, C. Katz, J. McCarthy, A.J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J.H. Wegstein, A. van Wijngaarden, and M. Woodger. “Report on the algorithmic language ALGOL 60”. In: *Communications of the ACM* 3.5 (1960). Editor: P. Naur, pp. 299–314.
- [21] J.W. Backus, F.L. Bauer, J. Green, C. Katz, J. McCarthy, A.J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J.H. Wegstein, A. van Wijngaarden, M. Woodger, and P. Naur. “Revised report on the algorithmic language ALGOL 60”. In: *Communications of the ACM* 6.1 (1963), pp. 1–17.
- [22] P.R. Bagley. *Letter to Mort Bernstein*. From the Charles Babbage Institute collections. Thanks to David Nofre for giving me a copy of this letter. 1960.
- [23] P.R. Bagley. *Letter to the SHARE UNCOL Committee and other interested parties, 26 May 1960*. From the Charles Babbage Institute collections. Thanks to David Nofre for giving me a copy of this letter. 1960.
- [24] T.P. Baker and A.C. Fleck. “A Note on Pascal Scopes”. In: *Pascal News* 17 (1980), p. 62.

- [25] T.P. Baker and A.C. Fleck. "Does Scope = Block in Pascal?" In: *Pascal News* 17 (1980), pp. 60–61.
- [26] Y. Bar-Hillel. *Language and Information: Selected Essays on their Theory and Application*. Reading, Massachusetts and Jerusalem, Israel: Addison-Wesley Publishing Company, Inc. and the Jerusalem Academic Press Ltd, 1964.
- [27] Y. Bar-Hillel, M. Perles, and E. Shamir. "On formal properties of simple phrase structure grammars". In: *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* 14 (1961), pp. 143–172.
- [28] R.G. Bartle. *The Elements of Real Analysis*. Wiley, 1964.
- [29] C. Gordon Bell and A. Newell. *Computer Structures: Readings and Examples*. New York, USA: McGraw Hill, 1971.
- [30] R.W. Bemer. "The Status of Automatic Programming for Scientific Problems". In: *The Fourth Annual Computer Applications Symposium, October 24–25, 1957*. Ed. by F.C. Bock. Armour Research Foundation of Illinois Institute of Technology, 1958, pp. 107–117.
- [31] E.C. Berkeley. *Giant Brains, or Machines That Think*. New York: Wiley, 1949.
- [32] E.C. Berkeley. *Memorandum for the Association for Computing Machinery — Committee on the Social Responsibilities of Computer Scientists*. Tech. rep. 1958. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 3.
- [33] E. Berkens and E.G. Daylight. *De geest van de computer: Een geschiedenis van software in Nederland*. ISBN 978-90-5345-504-3. Uitgeverij Matrijs, 2016.
- [34] R. Bird and O. de Moor. *Algebra of Programming*. Prentice Hall, 1997.
- [35] G.E. Blelloch and R. Harper. " $\lambda$ -Calculus: The Other Turing Machine". In: *CMU CSD Fiftieth Anniversary volume*. 2015.
- [36] E. Bloch. *Proofs and Fundamentals*. Springer, 2011.
- [37] J. Bloem. "Gerrit Blaauw: Van 'rekenmachines' die het doen naar computerarchitectuur". In: *Informatie* (2006).
- [38] D.G. Bobrow and B. Raphael. "A comparison of list-processing languages: including a detailed comparison of COMMIT, IPL-V, LISP 1.5, and SLIP". In: *Communications of the ACM* (1964), pp. 231–240.
- [39] A. van den Bogaard. "Stijlen van Programmeren 1952–1972". In: *Studium 2* (2008), pp. 128–144.

- [40] C. Böhm and G. Jacopini. "Flow diagrams, Turing machines, and languages with only two formation rules". In: *Communications of the ACM* 9.5 (May 1966), pp. 366–371.
- [41] A.D. Booth and K.H.V. Booth. *Automatic Digital Calculators*. second. London, UK: Butterworths Scientific Publications, 1956.
- [42] N. Bourbaki. *Théorie des ensembles*. Hermann & c<sup>ie</sup>, 1954.
- [43] R. Boute. "Concrete Generic Functionals". In: *Generic Programming*. Ed. by Jeremy Gibbons and Johan Jeuring. Kluwer, 2003, pp. 89–119.
- [44] R. Boute. *Rekindling critical thinking: heeding major errors in current Introduction to Proof type textbooks*. <http://www.funmath.be/CriTnk.pdf>. Contributed paper session, MAA Mathfest 2015. 2015.
- [45] R. Boute. "Why mathematics needs engineering". In: *Journal of Logical and Algebraic Methods in Programming* 85.5, part 2 (2016). <http://dx.doi.org/10.1016/j.jlamp.2016.01.001>, pp. 867–878.
- [46] R.T. Boute. "System Semantics and Formal Circuit Description". In: *IEEE Transactions on Circuits and Systems CAS-33.12* (1986), pp. 1219–1231.
- [47] R.T. Boute. "Systems Semantics: Principles, Applications, and Implementation". In: *ACM Transactions on Programming Languages and Systems* 10.1 (1988), pp. 118–155.
- [48] B.W. Bowden, ed. *Faster Than Thought: A Symposium on Digital Computing Machines*. London: Sir Isaac Pitman & Sons, Ltd., 1953.
- [49] J.M. Boyle and A.A. Grau. "An algorithmic semantics for ALGOL 60 identifier denotation". In: *Journal of the ACM* 17 (1970), pp. 361–382.
- [50] J. Brown and J.W. Carr III. "Automatic Programming and its Development on the MIDAC". In: *Symposium on Automatic Programming for Digital Computers*. Office of Naval Research, Department of the Navy. Washington D.C., 1954, pp. 84–97.
- [51] N.G. de Bruijn. *Verslag inzake onderzoek betreffende electronische en elektrische rekenapparatuur over het cursusjaar 1947/48*. Tech. rep. Delft, 1948.
- [52] T. Budd. *A Little Smalltalk*. Addison-Wesley, 1987.
- [53] M. Bullynck. "Programming primes (1968-1976)". In: *History and Philosophy of Logic* 36 (2015), pp. 229–241.
- [54] A.W. Burks. "Turing's Theory of Infinite Computing Machines (1936–1937) and its Relation to the Invention of Finite Electronic Computers (1939–1949)". In: *Theory and Practical Issues on Cellular*

- Automata*. Ed. by S. Bandini and T. Worsch. London Berlin Heidelberg: Springer, 2001, pp. 179–197.
- [55] A.W. Burks. “The invention of the universal electronic computer—how the Electronic Computer Revolution began”. In: *Future Generation Computer Systems* 18 (2002), pp. 871–892.
- [56] M. Campbell-Kelly. “Alan Turing’s Other Universal Machine: Reflections on the Turing ACE computer and its influence”. In: *Communications of the ACM* 55.7 (2012), pp. 31–33.
- [57] M. Campbell-Kelly and W. Aspray. *Computer: A History of the Information Machine*. New York, NY, USA: Basic Books, 1996.
- [58] C. Câmpeanu, K. Salomaa, and S. Yu. “A formal study of practical regular expressions”. In: *International Journal of Foundations of Computer Science* 14 (2003), pp. 1007–1018.
- [59] J.W. Carr. *Inaugural Presidential Address*. Presented at the meeting of the Association. 1956.
- [60] J.W. Carr. *Computing Programming and Artificial Intelligence*. Ann Arbor, University of Michigan. An intensive course for practicing scientists and engineers: lectures given at the University of Michigan. 1958.
- [61] J.W. Carr. “Programming and Coding”. In: *Handbook of Automation, Computation, and Control: Computers and Data Processing*. Ed. by E.M. Grabbe, S. Ramo, and D.E. Wooldridge. Vol. 2. New York: John Wiley and Sons, Inc., 1959. Chap. 2.
- [62] G. Chartrand, A. Polimeni, and P. Zhang. *Mathematical Proofs: A Transition to Advanced Mathematics (3rd. ed.)* Pearson, 2012.
- [63] S. Chaudhuri, A. Farzan, and Z. Kincaid. “Consistency Analysis of Decision-Making Programs”. In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2014, pp. 555–567.
- [64] S. Chaudhuri, S. Gulwani, and R. Lublinerman. “Continuity & Robustness of Programs”. In: *Communications of the ACM* 55.8 (2012), pp. 107–115.
- [65] N. Chomsky. “Transformational Analysis”. PhD thesis. University of Pennsylvania, 1955.
- [66] N. Chomsky. “Three models for the description of language”. In: *I.R.E. Trans. on Information Theory IT-2* (1956), pp. 113–123.
- [67] N. Chomsky. *Syntactic Structures*. The Hague/Paris: Mouton, 1957.
- [68] N. Chomsky. “On certain formal properties of grammars”. In: *Information and Control* 2 (1959), pp. 137–167.

- [69] A. Church. "A set of postulates for the foundation of logic". In: *Annals of Mathematics* 2 (1932-33), pp. 33–34, 346–366, 839–864.
- [70] F. Cohen. "Computer Viruses: Theory and Experiments". In: *Computers and Security* 6.1 (1987), pp. 22–35.
- [71] I. Bernard Cohen. *Howard Aiken: Portrait of a Computer Pioneer*. MIT Press, 1999.
- [72] T.R. Colburn. *Philosophy and Computer Science*. Ed. by J.H. Fetzer. M.E. Sharpe, 2000.
- [73] A. Colmerauer and P. Roussel. "The birth of Prolog". In: *History of Programming Languages – II*. Ed. by T.J. Bergin Jr. and R.G. Gibson Jr. ACM Press, 1996, pp. 331–367.
- [74] B. Cook, A. Podelski, and A. Rybalchenko. "Proving Program Termination". In: *Communications of the ACM* 54.5 (May 2011), pp. 88–98.
- [75] S.B. Cooper. "Incomputability after Alan Turing". In: *Notices of the AMS* 59.6 (2012), pp. 776–784.
- [76] B. Jack Copeland. *Turing: Pioneer of the Information Age*. Oxford, UK: Oxford University Press, 2012.
- [77] H.B. Curry. *On the Composition of Programs for Automatic Computing*. Memorandum 9806. Silver Spring, Maryland: Naval Ordnance Laboratory, 1949.
- [78] H.B. Curry and R. Feys. *Combinatory Logic. Volume I*. Amsterdam: North-Holland, 1958.
- [79] U. Daepf and P. Gorkin. *Reading, Writing and Proving: a Closer Look at Mathematics*. Springer, 2003.
- [80] U. Daepf and P. Gorkin. *Reading, Writing and Proving: a Closer Look at Mathematics (2nd. ed.)* Springer, 2011.
- [81] O.-J. Dahl, E.W. Dijkstra, and C.A.R. Hoare. *Structured Programming*. London/New York: Academic Press, 1972.
- [82] O.-J. Dahl, B. Myhrhaug, and K. Nygaard. *The SIMULA 67 common base language*. Tech. rep. Norwegian Computing Center, Oslo, 1968.
- [83] O.-J. Dahl and K. Nygaard. "SIMULA—an ALGOL-based simulation language". In: *Communications of the ACM* 9.9 (1966), pp. 671–678.
- [84] A. Dasgupta. *Set Theory*. Birkhäuser, 2014.
- [85] S. Dasgupta. *Computer Science: A Very Short Introduction*. Oxford University Press, 2016.
- [86] M. Davis. *Computability and Unsolvability*. New York, USA: McGraw-Hill, 1958.

- [87] M. Davis. "Mathematical Logic and the Origin of Modern Computers". In: *The Universal Turing Machine - A Half-Century Survey*. Ed. by R. Herken. Originally in: *Studies in the History of Mathematics*. Mathematical Association of America, 1987, pages 137-165. Wien: Springer, 1988, pp. 135–158.
- [88] M. Davis. *Engines of Logic: Mathematicians and the origins of the Computer*. first. New York NY: W.W. Norton & Company, 2000.
- [89] M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. first. Florida: Norton, 2000.
- [90] M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. second. CRC Press, 2012.
- [91] M. Davis. *Universality is Ubiquitous*. YouTube: [www.youtube.com/watch?v=ZVTgtODX0Nc](http://www.youtube.com/watch?v=ZVTgtODX0Nc). Presentation at the Turing Centennial Celebration at Princeton, 10–12 May 2012. 2012.
- [92] M. Davis, R. Sigal, and E.J. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. second. Morgan Kaufmann, 1994.
- [93] E.G. Daylight. *Interview with Van der Poel in February 2010, conducted by Gerard Alberts, David Nofre, Karel Van Oudheusden, and Jelske Schaap*. Tech. rep. 2010.
- [94] E.G. Daylight. "Dijkstra's Rallying Cry for Generalization: the Advent of the Recursive Procedure, late 1950s – early 1960s". In: *The Computer Journal* 54.11 (2011), pp. 1756–1772.
- [95] E.G. Daylight. *Interview with Blaauw on 29 November 2011, conducted by Gerard Alberts and Karel Van Oudheusden*. Tech. rep. 2011.
- [96] E.G. Daylight. *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains*. Ed. by K. De Grave. Heverlee: Lonely Scholar, 2011.
- [97] E.G. Daylight. "A Hard Look at George Dyson's Book "Turing's Cathedral: the Origins of the Digital Universe"". In: *Turing in Context II*. Lecture available on video: [www.dijkstrascry.com/presentations](http://www.dijkstrascry.com/presentations). 2012.
- [98] E.G. Daylight. *The Dawn of Software Engineering: from Turing to Dijkstra*. Ed. by K. De Grave. See: [www.lonelyscholar.com](http://www.lonelyscholar.com). Heverlee: Lonely Scholar, 2012.
- [99] E.G. Daylight. *On Mahoney's Accounts of Turing*. 2013.
- [100] E.G. Daylight. *Turing's 1936 Paper and the First Dutch Computers*. Communications of the ACM. 2013.
- [101] E.G. Daylight. "A Turing Tale". In: *Communications of the ACM* 57.10 (2014), pp. 36–38.

- [102] E.G. Daylight. "Towards a Historical Notion of "Turing — the Father of Computer Science"". In: *History and Philosophy of Logic* 36.3 (2015), pp. 205–228.
- [103] E.G. Daylight and S. Nanz, eds. *The Future of Software Engineering: Panel discussions, 22–23 November 2010, ETH Zurich*. Conversations. www.lonelyscholar.com. Heverlee: Lonely Scholar, Oct. 2011.
- [104] E.G. Daylight, A. Vandecappelle, and F. Catthoor. "The Formalism Underlying EASYMAP: a Precompiler for Refinement-Based Exploration of Hierarchical Data Organizations". In: *Science of Computer Programming* 72.3 (Aug. 2008), pp. 71–135.
- [105] T.J. Dekker, E.W. Dijkstra, and A. van Wijngaarden. *Cursus programmeren voor automatische rekenmachines*. Tech. rep. Amsterdam: MCR:CR-9, 1957.
- [106] E.W. Dijkstra. *Functionele beschrijving van de ARRA*. Tech. rep. MR 12. Mathematisch Centrum Amsterdam, 1953.
- [107] E.W. Dijkstra. *Communication with an Automatic Computer*. Academisch Proefschrift. Universiteit van Amsterdam, Oct. 1959.
- [108] E.W. Dijkstra. "An Attempt to Unify Constituent Concepts of Serial Program Execution". In: *Proceedings of the Symposium Symbolic Languages in Data Processing*. London/New York: Gordon and Breach Science Publishers, 1962, pp. 237–251.
- [109] E.W. Dijkstra. "GOTO Statement Considered Harmful". In: *Letters to the Editor, Communications of the ACM* 11 (1968), pp. 147–148.
- [110] E.W. Dijkstra. *EWD 249: Notes on structured programming*. Tech. rep. Technische Hogeschool Eindhoven, 1969.
- [111] E.W. Dijkstra. *Notes on Structured Programming*. Tech. rep. T.H.-Report 70-WSK-03. Second edition. Published as Chapter 1 of [81]. Technische Hogeschool Eindhoven, Apr. 1970.
- [112] E.W. Dijkstra. *EWD 372: A simple axiomatic basis for programming language constructs*. Tech. rep. 1973.
- [113] E.W. Dijkstra. *A Discipline of Programming*. Englewood Cliffs, N.J.: Prentice-Hall, 1976.
- [114] E.W. Dijkstra. *EWD 1166: From my Life*. Tech. rep. The University of Texas at Austin, 1993.
- [115] P.J. van Donselaar. *De ontwikkeling van elektronische rekenmachines in Nederland; een historisch overzicht van Nederlandse computers*. Tech. rep. Amsterdam: Rapport Stichting Studiecencentrum voor Administratieve Automatisering en Bestuurlijke Informatieverwerking, 1967.

- [116] G. Dowek. *Computation, Proof, Machine: Mathematics Enters a New Age*. Cambridge University Press, 2015.
- [117] G. Dyson. *Turing's Cathedral: The Origins of the Digital Universe*. London: Penguin Books, 2012.
- [118] E.A. Emerson. "25 Years of Model Checking". In: *The Beginning of Model Checking: A Personal Perspective*. Springer, 2008, pp. 27–45.
- [119] U. Erlingsson, Y. Younan, and F. Piessens. "Low-Level Software Security by Example". In: *Handbook of Information and Communication Security*. Ed. by P. Stavroulakis and M. Stamp. Springer, 2010, pp. 633–658.
- [120] G.R. Exner. *An Accompaniment to Higher Mathematics*. Springer, 1997.
- [121] A.D. Falkoff and K.E. Iverson. *APL\360: User's Manual*, IBM, 1968.
- [122] A.D. Falkoff and K.E. Iverson. "The Evolution of APL". In: *ACM SIGPLAN Notices* 13 (1978), pp. 47–57.
- [123] D.J. Farber, R.E. Griswold, and I.P. Polonsky. "SNOBOL, A String Manipulation Language". In: *Journal of the ACM* 11 (1964), pp. 21–30.
- [124] M. Fernández. *Models of Computation: An Introduction to Computability Theory*. London: Springer, 2009.
- [125] J.H. Fetzer. "Program Verification: The Very Idea". In: *Communications of the ACM* 31.9 (1988), pp. 1048–1063.
- [126] E. Filiol. *Computer Viruses: from theory to applications*. Springer, 2005.
- [127] A.C. Fleck. "Isomorphism groups of automata". In: *Journal of the ACM* 9 (1962), pp. 469–476.
- [128] A.C. Fleck. "Algebraic structure of automata". PhD thesis. Michigan State University Library, 108 286 THS, 1964.
- [129] A.C. Fleck. "On the automorphism group of an automaton". In: *Journal of the ACM* 12 (1965), pp. 566–569.
- [130] A.C. Fleck. "Towards a theory of data structures". In: *Journal of Computer and System Sciences* 5 (1971), pp. 475–488.
- [131] A.C. Fleck. "On the impossibility of content exchange through the by-name parameter transmission mechanism". In: *SIGPLAN Notices* 11 (1976), pp. 38–41.
- [132] A.C. Fleck. "Formal models for string patterns". In: *Current Trends in Programming Methodology, Volume IV: Data Structuring*. Ed. by R. Yeh. Prentice-Hall, 1978, pp. 216–240.
- [133] A.C. Fleck. "Verifying abstract data types with SNOBOL4". In: *Software – Practice and Experience* 12 (1982), pp. 627–640.



- [134] A.C. Fleck. "A proposal for the comparison of types in Pascal and associated semantic models". In: *Computer Languages* 9.2 (1984), pp. 71–87.
- [135] A.C. Fleck. "Structuring FP-style functional programs". In: *Computer Languages* 11 (1986), pp. 55–63.
- [136] A.C. Fleck. "A case study comparison of four declarative programming languages". In: *Software – Practice and Experience* 20 (1990), pp. 49–66.
- [137] A.C. Fleck. "Specifying and proving object-oriented programs". In: *2004 Hawaii International Conference on Computer Sciences*. 2004, pp. 190–206.
- [138] A.C. Fleck. "Prolog as the first programming language". In: *ACM SIGCSE Bulletin* 39 (2007), pp. 61–64.
- [139] A.C. Fleck and R.S. Limaye. "Formal semantics and abstract properties of string pattern operations and extended formal language description mechanisms". In: *SIAM Journal on Computing* 12 (1983), pp. 166–188.
- [140] T.M. Flett. *Mathematical Analysis*. McGraw-Hill, 1966.
- [141] R.W. Floyd. "The syntax of programming languages—A survey". In: *IEEE Transactions on Electronic Computers* EC-13.4 (1964), pp. 346–353.
- [142] M. Franssen, G. Lokhorst, and I. Poel. "Philosophy of technology". In: *Stanford Encyclopedia of Philosophy*. <http://plato.stanford.edu/entries/technology>, 2009.
- [143] D.D. Freydenberger. "Extended regular expressions: succinctness and decidability". In: *Theory of Computing Systems* 53 (2013), pp. 159–193.
- [144] J.E.F. Friedl. *Mastering Regular Expressions*. 3rd ed. O'Reilly Media, Inc., 2006.
- [145] R. Frigg. "Models and fiction". In: *Synthese* 172 (2009), pp. 251–268.
- [146] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [147] R. Garnier and J. Taylor. *Discrete Mathematics — Proofs, Structures and Applications*. CRC Press, 2010.
- [148] L. Gerstein. *Introduction to Mathematical Structures and Proofs (2nd ed.)* Springer, 2012.
- [149] L. Gilbert and J. Gilbert. *Elements of Modern Algebra (7th. ed.)* Cengage Learning, 2008.

- [150] S. Ginsburg. "On the reduction of superfluous states in sequential machines". In: *Journal of the ACM* 6 (1959), pp. 259–282.
- [151] S. Ginsburg and H. Gordon Rice. "Two Families of Languages Related to ALGOL". In: *Journal of the ACM* 9.3 (1962), pp. 350–371.
- [152] V.E. Giuliano and A.G. Oettinger. "Research on automatic translation at the Harvard Computation Laboratory". In: *Information processing: proceedings of the International Conference on Information Processing*. Unesco, Paris. 1959.
- [153] J.A. Goguen. "Some design principles and theory for OBJ-0, a language for expressing and executing algebraic specifications of programs". In: *Mathematical Studies of Information Processing*. Ed. by E. Blum, M. Paul, and S. Takasu. LNCS V.75. Springer-Verlag, 1979, pp. 425–473.
- [154] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. "Abstract data-types as initial algebras and correctness of data representations". In: *Computer Graphics, Pattern Recognition and Data Structure*. 1975, pp. 89–93.
- [155] A. Goldberg and D. Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley, 1983.
- [156] E.G. Goodaire and M.M. Parmenter. *Discrete Mathematics with Graph Theory (3rd. ed.)* Pearson Prentice Hall, 2006.
- [157] R. Goodman, ed. *Annual Review in Automatic Programming I: Papers read at the Working Conference on Automatic Programming of Digital Computers held at Brighton, 1–3 April 1959*. New York, USA: Pergamon Press, 1960.
- [158] S. Gorn. "Planning Universal Semi-Automatic Coding". In: *Symposium on Automatic Programming for Digital Computers*. Office of Naval Research, Department of the Navy. Washington D.C., May 1954, pp. 74–83.
- [159] S. Gorn. *Real Solutions Of Numerical Equations By High Speed Machines*. Tech. rep. 966. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection). Ballistic Research Laboratories, 1955.
- [160] S. Gorn. "Standardized Programming Methods and Universal Coding". In: *Journal of the ACM* (July 1957). Received in December 1956.
- [161] S. Gorn. *Common Programming Language Task, Final Report: Report of the work in the period 1 May 1958 to 30 June 1959*. Tech. rep. AD59UR1. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 39. 1959.

- [162] S. Gorn and W. Manheimer. *The electronic brain and what it can do*. Ed. by P.F. Brandwein. Chicago, Illinois, USA: Science Research Associates, Inc., 1956.
- [163] E.M. Grabbe, S. Ramo, and D.E. Wooldridge, eds. *Handbook of Automation, Computation, and Control: Computers and Data Processing*. Vol. 2. New York: John Wiley and Sons, Inc., 1959.
- [164] D. Gries and F.B. Schneider. *A Logical Approach to Discrete Math*. Springer, 1993.
- [165] R.E. Griswold. "History of Programming Languages". In: ACM Press, 1981. Chap. A history of the SNOBOL programming languages, pp. 601–645.
- [166] R.E. Griswold, J.F. Poage, and I.P. Polonsky. *The SNOBOL4 Programming Language*. Prentice-Hall, 1968.
- [167] R.E. Griswold, J.F. Poage, and I.P. Polonsky. *The SNOBOL4 Programming Language*. 2nd ed. Prentice-Hall, 1971.
- [168] J.V. Guttag, E. Horowitz, and D.R. Musser. "The design of data type specifications". In: *ICSE'76: 2nd International Conference on Software Engineering*. IEEE, 1976, pp. 414–430.
- [169] J.V. Guttag, E. Horowitz, and D.R. Musser. "Abstract data types and software validation". In: *Communications of the ACM* 21 (1978), pp. 1048–1063.
- [170] T. Haigh, M. Priestley, and C. Rope. *ENIAC in Action*. MIT Press, 2016, p. 360.
- [171] P.R. Halmos. "Nicolas Bourbaki". In: *Scientific American* 196.5 (1957), pp. 88–99.
- [172] P.R. Halmos. *Naïve Set Theory*. Van Nostrand Reinhold, 1960.
- [173] R. Hammack. *Book of Proof*. CC BY-ND, 2009.
- [174] D. Harel. *The Science of Computing: Exploring the Nature and Power of Algorithms*. Addison-Wesley, 1987, 1989.
- [175] I.N. Herstein. *Topics in Algebra*. Xerox College Publishing, 1964.
- [176] M. Hicks. *How did Heartbleed remain undiscovered, and what should we do about it?* This post is dated "July 1st, 2014" and I accessed it on September 19th, 2016. URL: <http://www.pl-enthusiast.net/2014/07/01/how-did-heartbleed-remain-undiscovered-and-what-should-we-do-about-it/>.
- [177] C.A.R. Hoare. "Record handling". In: *ALGOL Bulletin* 21 (1965), pp. 39–69.
- [178] C.A.R. Hoare. "An Axiomatic Basis for Computer Programming". In: *Communications of the ACM* 12.10 (1969), pp. 576–580.

- [179] C.A.R. Hoare. "Proof of correctness of data representations". In: *Acta Informatica* 1 (1972), pp. 271–281.
- [180] C.A.R. Hoare. "The Emperor's Old Clothes". In: *Communications of the ACM* 24.2 (1981), pp. 75–83.
- [181] C.A.R. Hoare and D.C.S. Allison. "Incomputability". In: *ACM Computing Surveys* 4.3 (1972), pp. 169–178.
- [182] A. Hodges. *Alan Turing: The Enigma*. London: Burnett Books, 1983.
- [183] A. Hodges. "Book Review: The Essential Turing". In: *Notices of the AMS* 53.10 (2006), pp. 1190–1199.
- [184] G. van den Hove. "On the Origin of Recursive Procedures". In: *The Computer Journal* 58.11 (2015), pp. 2892–2899.
- [185] R.K.W. Hui, K.E. Iverson, E.E. McDonnell, and A.T. Whitney. "APL?" In: *APL1990*. 1990, pp. 192–200.
- [186] Yu. I. Ianov. "On the equivalence and transformation of program schemes". In: *Doklady Akad. Nauk S.S.S.R.* 113 (1957), pp. 39–42.
- [187] IBM, *650 Magnetic Drum Data-Processing Machine Manual of Operation, Form 22-6060-1, 1955*.
- [188] IBM, *Programmer's Reference Manual (for the) Fortran Automatic Coding System for the IBM704, 1956*.
- [189] *IEEE Standard Pascal Computer Programming Language, ANSI/IEEE X3.97-1983*. American National Standards Institute, Inc., 1983.
- [190] N. Irmak. "Software is an Abstract Artifact". In: *Grazer Philosophische Studien* 86 (2012), pp. 55–72.
- [191] ISO/IEC. *Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology (ISO 80000-2)*. ISO/IEC, 2009.
- [192] K.E. Iverson. *A Programming Language*. New York: John Wiley and Sons, Inc., 1962.
- [193] M.A. Jackson and E.G. Daylight. *Formalism and Intuition in Software Development*. Ed. by K. De Grave. Geel: Lonely Scholar, 2015.
- [194] T. Jech. *Set Theory*. Springer, 2003.
- [195] K. Jensen and N. Wirth. *Pascal User Manual and Report*. Springer-Verlag, 1974.
- [196] R.T. Johnson and J.B. Morris. "Abstract data types in the MODEL programming language". In: *SIGPLAN Notices* 11 (1976), pp. 36–46.

- [197] A.A. Markov Jr. *Theory of Algorithms*. Vol. 42. Trudy Matematicheskogo Instituta imeni V. A. Steklova. Moscow/Leningrad: Academy of Sciences of the USSR, 1954.
- [198] *Jsoftware Inc.*, <http://www.jsoftware.com>.
- [199] D. Kahn. *The Codebreakers: The Story of Secret Writing*. New York: The Macmillan Company, 1967.
- [200] L.V. Kantorovich. "On a mathematical symbolism convenient for performing machine calculations". In: *Doklady Aka. Nauk S.S.S.R.* 113 (1957), pp. 738–741.
- [201] A.C. Kay. "The early history of Smalltalk". In: *History of Programming Languages – II*. Ed. by T.J. Bergin Jr. and R.G. Gibson Jr. ACM Press, 1996, pp. 511–598.
- [202] S.C. Kleene. *Introduction to Metamathematics*. Princeton, New Jersey, USA: Van Nostrand, 1952.
- [203] S.C. Kleene. "Representation of events in nerve nets and finite automata". In: *Automata Studies*. Ed. by C.E. Shannon and J. McCarthy. Princeton University Press, 1956, pp. 3–42.
- [204] G. Klein et al. "seL4: Formal Verification: of an OS Kernel". In: *SOSP*. 2009.
- [205] D.E. Knuth. "The remaining trouble spots in ALGOL 60". In: *Communications of the ACM* 10 (1967), pp. 611–618. Reprinted with corrections and an addendum in [209].
- [206] D.E. Knuth. "Semantics of Context-Free Languages". In: *Mathematical Systems Theory* 2 (1968), pp. 127–145. Reprinted with corrections and an addendum in [209].
- [207] D.E. Knuth. "Literate Programming". In: *The Computer Journal* 27 (1984), pp. 97–111. Reprinted as Chapter 4 of [208].
- [208] D.E. Knuth. *Literate Programming*. Vol. 27. CSLI Lecture Notes. Stanford, California: CSLI Publications, 1992.
- [209] D.E. Knuth. *Selected Papers on Computer Languages*. Vol. 139. CSLI Lecture Notes. Stanford, California: CSLI Publications, 2003.
- [210] D.E. Knuth and E.G. Daylight. *Algorithmic Barriers Falling: P=NP?* Ed. by K. De Grave. Geel: Lonely Scholar, 2014.
- [211] A.L. Kolmogorov and S.V. Fomin. *Introductory Real Analysis*. Dover, 1970.
- [212] E. Kranakis. "Early Computers in The Netherlands". In: *CWI-Quarterly* (), pp. 61–274.
- [213] S.G. Krantz. *Real Analysis and Foundations*. Chapman & Hall/CRC, 2005.

- [214] D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.
- [215] P. Kroes. "Engineering and the dual nature of technical artefacts". In: *Cambridge Journal of Economics* 34 (2010), pp. 51–62.
- [216] L. Lamport. *Specifying Systems — The TLA+ Language and Tools for Hardware and Software Engineers*. Pearson, 2003.
- [217] L. Lamport and L.C. Paulson. "Should your specification language be typed?" In: *ACM TOPLAS* 21.3 (1999), pp. 502–526.
- [218] S. Lang. *Undergraduate Analysis*. Springer, 1983.
- [219] R. Larson and B. Edwards. *Calculus 9e*. Brooks/Cole, 2009.
- [220] S. Lavington, ed. *Alan Turing and his Contemporaries: Building the world's first computers*. Swindon, UK: bcs, 2012.
- [221] D. Leavitt. *The Man Who Knew Too Much: Alan Turing and the Invention of the Computer*. New York: Atlas Books, 2006.
- [222] D. Leavitt. *Alan Turing, l'homme qui inventa l'informatique*. Paris: Dunod, 2007.
- [223] E.A. Lee. "Absolutely positive on time: what would it take?" In: *Computer* 38.7 (2005), pp. 85–87.
- [224] E.A. Lee and P. Varaiya. *Structure and Interpretation of Signals and Systems*. Addison Wesley / Pearson Education, 2003.
- [225] J.A.N. Lee. *Computer pioneers*. California: IEEE Computer Society Press, 1995.
- [226] H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation*. Ed. by Engelwood Cliffs. New Jersey: Prentice-Hall, 1981.
- [227] B. Liskov and S. Zilles. "Programming with abstract data types". In: *ACM SIGPLAN Notices* 9 (1974), pp. 50–59.
- [228] B.H. Liskov and S. Zilles. "Specification Techniques for Data Abstractions". In: *IEEE Transactions on Software Engineering* (1975), pp. 72–87.
- [229] K.C. Liu and A.C. Fleck. "String pattern matching in polynomial time". In: *6th ACM Symposium on Principles of Programming Languages*. 1979, pp. 222–225.
- [230] W.N. Locke and A.D. Booth, eds. *Machine Translation of Languages: Fourteen Essays*. Cambridge, MA and New York: MIT Press and John Wiley & Sons, Inc., 1955.
- [231] R.L. London. *Who Earned First Computer Science Ph.D.?* BLOG @CACM (2013), <http://cacm.acm.org/blogs/blog-cacm/159591-who-earned-first-computer-science-ph-d/fulltext>.

- [232] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.
- [233] D. MacKenzie. *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press, 2004.
- [234] M.S. Mahoney. *Histories of Computing*. Ed. by T. Haigh. Cambridge, Massachusetts/London, England: Harvard University Press, 2011.
- [235] Z. Manna and J. Vuillemin. "Fixpoint approach to the theory of computation". In: *Communications of the ACM* 15 (1972), pp. 528–536.
- [236] J. Martin-Nielsen. "'This war for men's minds': the birth of a human science in Cold War America". In: *History of the Human Sciences* 23.5 (2010), pp. 131–155.
- [237] S. Martini. "Several Types of Types in Programming Languages". In: *History and Philosophy of Computing* 2015. 2015, pp. 216–227.
- [238] J. McCarthy. "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". In: *Communications of the ACM* 3.4 (1960), pp. 184–195.
- [239] J. McCarthy. "History of Programming Languages". In: ed. by R.L. Wexelblat. New York: Academic Press, 1981. Chap. 'History of LISP' and the transcripts of: presentation, discussant's remark, question and answer session, pp. 173–195.
- [240] W.S. McCulloch and W. Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *Bull. math. Biophys.* 5 (1943), pp. 115–133.
- [241] G.H. Mealy. "A method for synthesizing sequential circuits". In: *Bell Systems Technology Journal* 34 (1955), pp. 1045–1079.
- [242] E. Mendelson. *Introduction to Mathematical Logic (3rd. ed.)* Wadsworth & Brooks / Cole, 1987.
- [243] A.R. Meyer and D.M. Ritchie. "The complexity of loop programs". In: *Proceedings of the ACM National Meeting*. 1967, pp. 465–469.
- [244] B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice Hall, 1991.
- [245] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc, 1967.
- [246] L. De Mol. "Doing mathematics on the ENIAC. Von Neumann's and Lehmer's different Visions." In: *Mathematical practice and development throughout History*. Ed. by E. Wilhelmus and I. Witzke. Logos Verlag, Berlin, 2009, pp. 149–186.
- [247] L. De Mol and M. Bullynck. "A short history of small machines". In: *CiE 2012 - How the World Computes*. 2012.

- [248] L. De Mol, M. Bullynck, and M. Carle. "Haskell before Haskell. Curry's contribution to a theory of programming." In: *Programs, Proofs, Processes, Computability in Europe 2010*. Vol. 6158. LNCS. 2010, pp. 108–117.
- [249] E.F. Moore. "Gedanken-experiments on sequential machines". In: *Automata Studies*. Princeton University Press, 1956, pp. 129–153.
- [250] H.L. Morgan and R.A. Wagner. "PL/C: the design of a high-performance compiler for PL/I". In: *AFIPS Spring Joint Computing Conference*. 1971, pp. 503–510.
- [251] P. Mounier-Kuhn. "Comment l'informatique devint une science". In: *La Recherche* 465 (2012), pp. 92–94.
- [252] P. Mounier-Kuhn. "Computer Science in French Universities: Early Entrants and Latecomers". In: *Information & Culture: A Journal of History* 47.4 (2012), pp. 414–456.
- [253] P. Mounier-Kuhn. "Logic and Computing in France: A Late Convergence". In: *AISB/IACAP World Congress 2012 — History and Philosophy of Programming*. Ed. by L. De Mol and G. Primiero. 2012, pp. 44–47.
- [254] P. Mounier-Kuhn. "Algol in France: From Universal Project to Embedded Culture". In: *IEEE Annals of the History of Computing* 36.4 (2014), pp. 6–25.
- [255] S. Nanz, ed. *The Future of Software Engineering*. Springer, 2011.
- [256] P. Naur. "The European side of the last phase of the development of ALGOL 60". In: *History of Programming Languages*. Ed. by R.L. Wexelblat. New York: Academic Press, 1981, pp. 92–139.
- [257] P. Naur. *Computing: A Human Activity*. New York: ACM Press / Addison-Wesley, 1992.
- [258] A. Newell and F.M. Tonge. "An Introduction to Information Processing Language V". In: *Communications of the ACM* 3.4 (1960), pp. 205–211.
- [259] D. Nofre. "Unraveling Algol: US, Europe, and the Creation of a Programming Language". In: *IEEE Annals of the History of Computing* 32.2 (2010), pp. 58–68.
- [260] D. Nofre. *Alan Jay Perlis*. Short bio of Perlis, available on the official ACM website: [amturing.acm.org/award\\_winners](http://amturing.acm.org/award_winners). 2012.
- [261] D. Nofre, M. Priestley, and G. Alberts. "When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950–1960". In: *Technology and Culture* 55.1 (2014), pp. 40–75.
- [262] K.V. Nori, U. Ammann, K. Jensen, H.H. Nageli, and C. Jacobi. "Pascal-P Implementation Notes". In: *Pascal – The Language and*



- its Implementation*. Ed. by D.W. Barron. John Wiley and Sons, Inc., 1981, pp. 125–170.
- [263] M.J. O'Donnell. "Computing in Systems Described by Equations". In: *Lecture Notes in Computer Science* 58 (1977).
- [264] A.G. Oettinger. "Programming a Digital Computer to Learn". In: *Philosophical Magazine* 43.347 (1952), pp. 1243–1263.
- [265] A.G. Oettinger. "Account identification for Automatic Data Processing". In: *Journal of the ACM* 4.3 (1957), pp. 245–253.
- [266] A.G. Oettinger. *Automatic Language Translation*. Cambridge, Massachusetts, USA: Harvard University Press, 1960.
- [267] A.G. Oettinger. "Automatic Syntactic Analysis and the Pushdown Store". In: *Proceedings of Symposium in Applied Mathematics*. Vol. 12. Providence: American Mathematical Society, 1961, pp. 104–129.
- [268] A.G. Oettinger. "Reminiscences of the Boss". In: *Makin' Numbers: Howard Aiken and the Computer*. Ed. by I.B. Cohen and G.W. Welch. Cambridge, Massachusetts, USA: MIT Press, 1999, pp. 203–214.
- [269] A. Olley. "Existence Precedes Essence — Meaning of the Stored-Program Concept". In: *IFIP Advances in Information and Communication Technology*. Ed. by A. Tatnall. Vol. 325. Springer, 2010, pp. 169–178.
- [270] C.H. Papadimitriou. *The Origin of Computable Numbers: A Tale of Two Classics*. YouTube: [www.youtube.com/watch?v=IZWjFCTx0OQ](http://www.youtube.com/watch?v=IZWjFCTx0OQ). Presentation at the Turing Centennial Celebration at Princeton, 10–12 May 2012. 2012.
- [271] D.L. Parnas. "On Proving Continuity of Programs". In: *Letter to the Editor of the Communications of the ACM* 55.11 (2012), p. 9.
- [272] A.J. Perlis. "Announcement". In: *Communications of the ACM* 1.1 (1958).
- [273] B.C. Pierce. *Basic Category Theory for Computer Scientists*. The MIT Press, 1991.
- [274] W. van der Poel. *Inzending 1946/47 van Van der Poel op de prijsvraag genaamd "1+1=10"*. Tech. rep. Delft, 1948.
- [275] W.L. van der Poel. "A Simple Electronic Digital Computer". In: *Appl. sci. Res.* 2 (1952), pp. 367–399.
- [276] W.L. van der Poel. "The Logical Principles of Some Simple Computers". PhD thesis. Universiteit van Amsterdam, 1956.
- [277] W.L. van der Poel. "Digitale Informationswandler". In: ed. by W. Hoffmann. Braunschweig: Vieweg, 1961. Chap. Microprogramming and trickology, pp. 269–311.
- [278] W.L. van der Poel. *Een leven met computers*. TU Delft. Oct. 1988.

- [279] *Preliminary Report – Specifications for the IBM Mathematical FORMula TRANslating System*. Tech. rep. New York: IBM, Programming Research Group, 1954.
- [280] M. Priestley. *A Science of Operations: Machines, Logic and the Invention of Programming*. Ed. by M. Campbell-Kelly. London: Springer, 2011.
- [281] P.M. Priestley. *Logic and the Development of Programming Languages, 1930-1975*. PhD thesis. University College London, May 2008.
- [282] W.V. Quine. *Set Theory and Its Logic*. The Belknap Press of Harvard University Press, 1969.
- [283] M.O. Rabin and D. Scott. "Finite Automata and their Decision Problems". In: *IBM Journal of Research and Development* 3.2 (1959), pp. 114–125.
- [284] G. Radin and H.P. Rogoway. "NPL: Highlights of a New Programming Language". In: *Communications of the ACM* 8 (1965), pp. 9–17.
- [285] B. Randell, ed. *The Origins of Digital Computers: Selected Papers*. Berlin Heidelberg New York: Springer-Verlag, 1973.
- [286] B. Randell and L.J. Russell. *ALGOL60 Implementation*. Academic Press, 1964.
- [287] S.N. Razumovskii. "On the question of automatization of programming of problems of translation from one language to another". In: *Doklady Akad. Nauk S.S.S.R.* 113 (1957), pp. 760–762.
- [288] J.C. Reynolds. *Letter to the author on 9 March 2012*.
- [289] C.J. van Rijsbergen. "Turing and the origins of digital computers". In: *Aslib Proceedings*. Vol. 37. 6/7. Paper presented at an Aslib Evening Meeting, Aslib, Information House, 27 March 1985. Emerald Backfiles, 1985, pp. 281–285.
- [290] C.E. Jr. Roberts. *Introduction to Mathematical Proofs — A Transition*. CRC Press, 2010.
- [291] J.A. Robinson. "A machine-oriented logic based on the resolution principle". In: *Journal of the ACM* 12 (1965), pp. 23–41.
- [292] J.A. Robinson. "Logic, computers, Turing, and von Neumann". In: *Machine Intelligence 13: Machine Intelligence and Inductive Learning*. Ed. by K. Furukawa, D. Michie, and S. Muggleton. Oxford, UK: Clarendon Press, 1994, pp. 1–35.
- [293] C. Rooijendijk. *Alles moest nog worden uitgevonden: De geschiedenis van de computer in Nederland*. Olympus, 2007.
- [294] P. Rosenbloom. *Elements of Mathematical Logic*. New York: Dover, 1950.
- [295] H.L. Royden. *Real Analysis*. Macmillan, 1968.

- [296] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 1964.
- [297] H. Rutishauser. "The Use of Recursive Procedures". In: *Annual Review in Automatic Programming* 3. Ed. by R. Goodman. New York: Pergamon Press, 1963, pp. 43–52.
- [298] J.G. Sanderson. "On Simple Low Redundancy Languages". In: *Communications of the ACM* 8.10 (1965). Letters to the Editor.
- [299] S.R. Sataluri and A.C. Fleck. "Semantic specification using logic programs". In: *Logic Programming, Proceedings of the North American Conference 1989*. Ed. by E.L. Lusk and R.A. Overbeek. Vol. 2. Cleveland, Ohio: MIT Press, 1989, pp. 772–794.
- [300] E.R. Scheinerman. *Mathematics — A Discrete Introduction (3rd. ed.)* Cengage Learning, 2012.
- [301] E. Shapiro. "Separating Concurrent Languages with Categories of Language Embeddings". In: *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*. 198–208. 1991.
- [302] H.S. Shuard. "Does it matter?" In: *The Mathematical Gazette* 59.407 (1976), pp. 7–15.
- [303] B.C. Smith. "The Limits of Correctness". In: *ACM SIGCAS Computers and Society* 14,15 (1985), pp. 18–26.
- [304] D. Smith, M. Eggen, and R. St Andre. *A Transition to Advanced Mathematics*. Cengage Learning, 2010.
- [305] J.M. Spivey. *The Z Notation – A Reference Manual*. <http://spivey.oriel.ox.ac.uk/~mike/zrm/>. Prentice Hall, 1989.
- [306] D. Sprecher. *Elements of Real Analysis*. Academic Press, 1970.
- [307] J.B. Stewart. *Calculus: Early Transcendentals (7th. ed.)* Cengage Learning, 2010.
- [308] H. Stoyan. "Early LISP History (1956-1959)". In: *LISP and Functional Programming*. 1984, pp. 299–310.
- [309] C. Strachey. "An impossible program". In: *Letter to the Editor of the Computer Journal* (1965), p. 313.
- [310] C. Strachey. *The Varieties of Programming Language*. Monograph PRG-10. Oxford University, Computing Laboratory, 1973.
- [311] C. Strachey and M.V. Wilkes. "Some proposals for improving the efficiency of ALGOL 60". In: *Communications of the ACM* 4.11 (1961). Also in: University Mathematical Laboratory Technical Memorandum, No. 61/5, pp. 488–491.
- [312] P. Suppes. *Axiomatic Set Theory*. Dover, 1972.
- [313] I.E. Sutherland. "Sketchpad: a man-machine graphical communication system". In: *AFIPS'63 — Spring Joint Computer Conference*. 1963, pp. 392–346.

- [314] *Symposium on Advanced Programming Methods for Digital Computers — Washington, D.C., June 28,29, 1956*. Available from the “Saul Gorn Papers” from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 43. 1956.
- [315] *Symposium on Automatic Programming for Digital Computers*. Office of Naval Research, Department of the Navy. Washington D.C., May 1954.
- [316] A. Tarski and S. Givant. *A Formalization of Set Theory Without Variables*. Reprinted with corrections 1988. The American Mathematical Society, 1987.
- [317] M. Tedre. *The Science of Computing: Shaping a Discipline*. Taylor and Francis, 2014.
- [318] K. Thompson. “Regular expression search algorithm”. In: *Communications of the ACM* 11.6 (1968), pp. 419–422.
- [319] H. Tuch, G. Klein, and M. Norrish. “Types, Bytes, and Separation Logic”. In: *Principles of Programming Languages*. 2007.
- [320] A.M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society, 2nd series* 42 (1936), pp. 230–265. Corrections provided in [321].
- [321] A.M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction”. In: *Proceedings of the London Mathematical Society, 2nd series* 43 (1937).
- [322] A.M. Turing. “Computing machinery and intelligence”. In: *Mind* 59 (1950), pp. 433–460.
- [323] D.A. Turner. “A new implementation technique for applicative languages”. In: *Software – Practice and Experience* 9 (1979), pp. 31–49.
- [324] D.A. Turner. “Miranda: a non-strict functional language with polymorphic types”. In: *IFIP International Conference on Functional Programming Languages and Computer Architecture*. Vol. 201. Lecture Notes in Computer Science. Springer-Verlag, 1985, pp. 1–16.
- [325] R. Turner. “Programming Languages as Technical Artefacts”. In: *Philosophy and Technology* 27.3 (2014). First online: 13 February 2013, pp. 377–397.
- [326] *UNCOL Committee Report*. From the Charles Babbage Institute collections. Thanks to David Nofre for giving me a copy of this letter. 1961.
- [327] *University of Pennsylvania Computer Activity Report: July 1, 1959 – December 31, 1960*. Tech. rep. Available from the “Saul Gorn Pa-

- pers" from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 39. Office of Computer Research and Education, 1960.
- [328] USA Standard FORTRAN, ANSI X3.9-1966. USA Standards Institute, Inc.
- [329] M.Y. Vardi. "Who Begat Computing?" In: *Communications of the ACM* 56.1 (2013), p. 5.
- [330] D.J. Velleman. *How To Prove It: A Structured Approach* (2nd. ed.) 5th printing. Cambridge, 2009.
- [331] C.J.D.M. Verhagen. *Rekenmachines in Delft*. Uitgave van de Commissie Rekenmachines van de Technische Hogeschool te Delft. 1960.
- [332] B. Bensaude Vincent. "Discipline-building in synthetic biology". In: *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences* 44.2 (2013), pp. 122–129.
- [333] J. Voeten. "On the fundamental limitations of transformational design". In: *ACM Transactions on Design Automation of Electronic Systems* 6.4 (2001), pp. 533–552.
- [334] W.D. Wallis. *A Beginner's Guide to Discrete Mathematics* (2nd. ed.) Birkhäuser, 2012.
- [335] R.L. Wexelblat, ed. *History of Programming Languages*. New York: Academic Press, 1981.
- [336] A. van Wijngaarden. *Switching and Programming*. Tech. rep. Report MR 50. Mathematisch Centrum Amsterdam, 1962.
- [337] A. van Wijngaarden. "Generalized ALGOL". In: *Annual Review in Automatic Programming*. Ed. by R. Goodman. Vol. 3. New York: Pergamon Press, 1963, pp. 17–26.
- [338] A. van Wijngaarden. "Numerical analysis as an independent science". In: *BIT* 6 (1966), pp. 66–81.
- [339] A. van Wijngaarden and E.W. Dijkstra. *Programmeren voor Automatische Rekenmachines*. Amsterdam, 1955.
- [340] Wikipedia. *List of programming languages*. [https://en.wikipedia.org/wiki/List\\_of\\_programming\\_languages](https://en.wikipedia.org/wiki/List_of_programming_languages).
- [341] M.V. Wilkes. "Can Machines Think?" In: *Spectator* 6424 (1951), pp. 177–178.
- [342] M.V. Wilkes. *Memories of a Computer Pioneer*. MIT Press, 1985.
- [343] N. Wirth. "The programming language Pascal". In: *Acta Informatica* 1 (1971), pp. 35–63.

- [344] N. Wirth. "Recollections about the development of Pascal". In: *History of Programming Languages*. Ed. by T.J. Bergin Jr. and R.G. Gibson Jr. Addison-Wesley, 1996, pp. 97–111.
- [345] N. Wirth and C.A.R. Hoare. "A contribution to the development of ALGOL". In: *Communications of the ACM* 9 (1966), pp. 423–432.
- [346] N. Wirth and H. Weber. "EULER: A Generalization of ALGOL, and its Formal Definition: Part I". In: *Communications of the ACM* 9.1 (Jan. 1966), pp. 13–25. Part II, *ibid.* 9.2 (1966) 89–99.
- [347] W. Wulf, R. London, and M. Shaw. "An introduction to the construction and verification of Alphard programs". In: *IEEE Transactions on Software Engineering* 2.4 (1976), pp. 253–265.
- [348] E. Zakon. *Mathematical Analysis, Vol. I*. <http://www.trillia.com/d9/zakon-analysisI-a4-one.pdf>. Trillia, 2004.