The Logic in Computer Science Column

BY

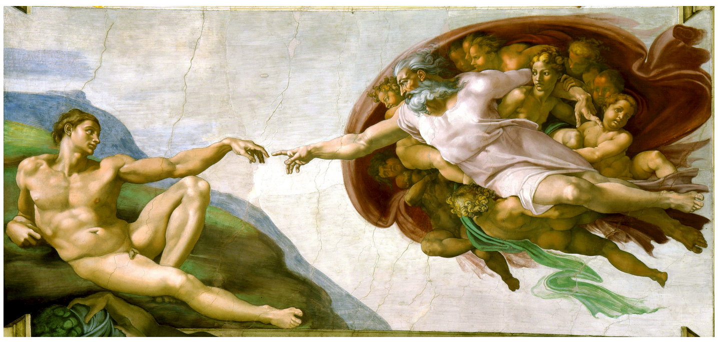
Yuri Gurevich

Computer Science & Engineering
University of Michigan, Ann Arbor, Michigan, USA
gurevich@umich.edu

Interactive classical algorithms: Preview

Yuri Gurevich

June 2025


The power of interaction

Abstract

This dialog paper offers a preview and provides a foretaste of an upcoming work on the axiomatization of interactive classical algorithms.

The modern notion of algorithm was elucidated in the 1930s–1950s. It was axiomatized a quarter of a century ago as the notion of "sequential algorithm"; we call it "classical algorithm" here. The axiomatization was used to show that for every classical algorithm there is a behaviorally equivalent abstract state machine. It was also used to prove the Church-Turing thesis as it has been understood by the logicians.

Starting from the 1960s, the notion of algorithm has expanded — probabilistic algorithms, quantum algorithms, etc. — prompting introduction of a much more ambitious version of the Church-Turing thesis commonly known as the "physical thesis." We emphasize the difference between the two versions of the Church-Turing thesis and illustrate how nondeterministic and probabilistic algorithms can be viewed as classical algorithms with appropriate oracles. The same view applies to quantum circuit algorithms and many other classes of algorithms.

[0]Partially supported by the US Army Research Office under W911NF-20-1-0297.

1 Is coin flipping algorithmic?

Q: Consider flipping a coin. Is it algorithmic?

A: Why do you ask?

Q: People speak about randomized algorithms, involving probability distributions. These distributions stem from physical processes, like flipping a coin. So a question arises: *who* flips the coin? I don't see how an algorithm could achieve that. An external agent has to perform the flip.

A: You could also ask whether a quantum measurement is algorithmic.

Q: Isn't that essentially the same question? A quantum measurement also yields a probability distribution.

A: Well, if you doubt that an algorithm can flip a coin, you might be even more skeptical about an algorithm performing a quantum measurement. That process involves Mother Nature, after all.

Q: But coin flipping involves Mother Nature as well, doesn't it?

A: You are right. The difference is that quantum measurement involves an aspect of nature that we are not accustomed to and don't fully understand, whereas coin flips have been well understood for a long time.

In any case, I stick to the traditional view that algorithms are inherently deterministic, making "nondeterministic algorithm" a contradiction in terms. Yogi Berra, an American philosopher and baseball player, once illustrated this: "When you come to a fork in the road, take it."

Q: How do you reconcile this view with the widespread use of the term "nondeterministic algorithm"?

A: This could be just a figure of speech. Nondeterministic algorithms can be viewed as deterministic algorithms that interact with their environment where someone makes the necessary choices, possibly by flipping a coin. Some authors say that a random sequence of 0's and 1's is part of the input, so that the necessary choices are made ahead of time.

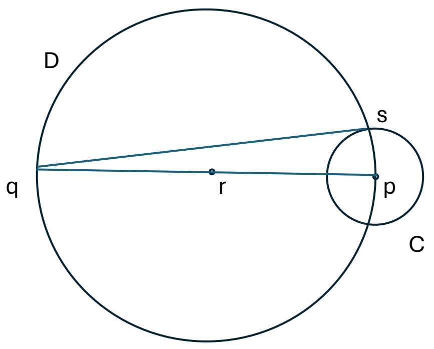
Alternatively — and quite legitimately — one can broaden the notion of algorithm, just as the notion of numbers was broadened. From positive integers all the way to real numbers, then to complex numbers and beyond.

Q: Let's consider a simple example of a nondeterministic algorithm.

A: Here's a classic ruler-and-compass algorithm with minimal nondeterminism. The setting is a fixed Euclidean plane. Given a circle C, its center p, and a point q outside of C, the algorithm constructs a tangent from q to C.

```
1. draw the midpoint $r$ between $p$ and $q$;
2. draw the circle $D$ centered at $r$ and passing through $q$;        (1)
3. choose a point $s$ where the circles $C, D$ intersect;
4. draw the line through $q$ and $s$.
```

The resulting line through q and s is the desired tangent to C.



Instruction 3 involves a nondeterministic choice. Either intersection point will work, but the choice must be made. The algorithm doesn't specify which one to choose; presumably, that's left to the executor. Aside from this step, the construction is fully deterministic.

2 Two vastly different theses

Q: Before we plunge into interactive algorithms, let me ask you, again, about the Dershowitz-Gurevich derivation of the Church-Turing thesis in [7]. Peter Shor is critical of it, but I'm interested in exploring the concerns he raises.

> "The Dershowitz-Gurevich paper says nothing about probabilistic or quantum computation. It does write down a set of axioms about computation, and prove the Church-Turing thesis assuming those axioms. However, we're left with justifying these axioms. Neither probabilistic nor quantum computation is covered by these axioms (they admit this for probabilistic computation, and do not mention quantum computation at all), so it's quite clear to me these axioms are actually false in the real world, even though the Church-Turing thesis is probably true" [21].

But let me start with this: Do you think he read the paper?

A: I'd guess that he just skimmed the abstract and then searched the text for "probabilistic" and "quantum."

Q: Is this arrogance or misperception?

A: Both, I think. The misperception would disappear if he read just a bit beyond the abstract.

There are two vastly different interpretations of the Church-Turing thesis in play. The thesis may be formulated thus:

> Church-Turing thesis. *Every effectively calculable string-to-string function is Turing computable.*

A question arises: what does "effectively calculable" mean?

One interpretation is traditional in logic. In the 1930s – 1950s, the meaning of "effectively calculable" was not in dispute. Logicians, including Church and Turing, — Turing wrote his dissertation in logic, under Church — had robust intuition about it. This intuition is elucidated in many books, for example in the influential books [15, §62], [19, §1.1], and [20, §9]. In this interpretation, effective calculation was deterministic. For example Turing writes: "The behaviour of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment" [22, §9]. Let me call this original interpretation of the thesis "classical" (though we should be careful with this term because, in quantum computing, "classical" means "not quantum").

The other interpretation is also natural in a sense, especially if you don't know the history of the subject. "Effectively calculable" can be interpreted as physically computable, which allows probabilistic and quantum computations as well as highly parallel, distributed, etc. This broader interpretation is commonly known as the "physical thesis."

Q: I see. You derive the classical thesis, and you think that Shor has in mind the physical thesis.

A: Yes, that is what I think. §1 of our paper is about effectivity. If Shor read just a little beyond the abstract, he would know that we prove the classical thesis.

Q: Why is it "quite clear" to Shor that your axioms "are actually false in the real world?"

A: I can only speculate about that. The axioms fail to cover probabilistic and quantum algorithms, so they are wrong and the proof does not establish the thesis. Nevertheless, the thesis "is probably true." Of course I think that it cannot be true [12].

`Q:` The speculation seems reasonable. But if the axioms imply the physical thesis then the thesis must have been formulated in mathematical terms, which is of independent interest. That should have occurred to Shor unless indeed he spent infinitesimal time on your paper. Anyway, is there a convincing formulation of the physical thesis?

`A:` I don't think so. Early on, Robin Gandy attempted such a formulation [6] but the attempt was not successful. In particular, it didn't cover distributed computing.

`Q:` Are your axioms true in the real world?

`A:` A better question is what algorithms satisfy the axioms?

`Q:` Do you admit that your axioms don't cover probabilistic computations?

`A:` "Admit" is the wrong word. Probabilistic algorithms are out of scope of our paper because they aren't classical. The relevant paragraph says: "Methods satisfying the Sequential Postulates include ... On the other hand, the postulates exclude ... They are also meant to exclude nondeterministic methods ..., probabilistic methods (like Rabin's algorithm for testing primality), ..." But the same paragraph includes this footnote:

> "Large classes of such non-classical algorithms are covered by the generalizations of the ASM Theorem in [1, 2, 3, 8]."

`Q:` I know from our earlier conversations that "ASM" stands for "abstract state machine." But what do you mean by the ASM theorem?

`A:` Let's take a quick dive into the history of the ASM project. It started in mid 1980s as a computer theory project — I wanted to understand what algorithms are. That led me to the notion of abstract state machines, originally called evolving algebras, and to the

> `ASM thesis.` *For every algorithm there is a behaviorally equivalent ASM.*

which I see as a natural extension of Turing's approach to the Church-Turing thesis. The project quickly became applied and practical. An ASM community emerged, and ASMs were successfully used for high-level executable specifications and related tasks. Article [14] is one example.

The ASM thesis was first formulated for classical algorithms [14] and then extended to algorithms in general [10]. Article [11] turned the ASM thesis, restricted to classical algorithms, called the Sequential ASM thesis in [11], into a theorem. That's the ASM theorem that you asked about.

Q: Why “sequential" rather than “classical" thesis?

A: At the time, it seemed to me that the thesis can’t be called classical, because it was new. Hence “sequential” as an imperfect substitute, emphasizing the difference from parallel, distributed, etc.

Q: But the ordinary meaning of sequential algorithms does not rule out probabilistic algorithms for example.

A: True. It would probably have been better to call the thesis classical — even though it was new — because it is about classical algorithms.

Q: Are readers “left with justifying” your axioms?

A: No. There are four axioms (or postulates) in [7]. The first three aim to capture all classical algorithms, whether they compute a function or not.

1. Sequential Time Postulate is self-evident, almost trivial. It says that an algorithm could be viewed as (finite or infinite) automaton with states, initial states, and a transition function.
2. Abstract State is natural, at least to logicians. The main part of it is that states can be viewed as structures in the sense of mathematical logic.
3. Bounded Exploration Postulate says that, during a step, only a bounded part of the state is explored, namely the part given by a fixed set of expressions. [1]

Besides, the work by the ASM community provided plenty of justification. There is little doubt that classical algorithms satisfy the three axioms. The surprising part is that the axioms are sufficient to capture the notion of classical algorithms mathematically.

Q: What is the fourth axiom for?

A: A string-to-string function, computed by a classical algorithm, is not necessarily Turing computable because the initial state may have too much information. The fourth axiom guarantees that the initial states are bare.

[1]The first two postulates occurred to me right away but it took me years to arrive at the third postulate. I might have arrived at it earlier if I knew at the time about Kolmogorov’s insight “An algorithmic process breaks down into separate steps of a priori bounded complexity” [16] in the minutes of Moscow Mathematical Society in Uspekhi Matematicheskikh Nauk, which is translated as Russian Mathematical Surveys, but minutes aren’t translated.

3 Spec Explorer

Q: Do you describe ASM applications in the Dershowitz-Gurevich paper?

A: We probably should have, but we didn't — mainly because we submitted the paper to the Bulletin of Symbolic Logic rather than a computer science journal.

Q: What would be a good example of ASM applications?

A: Let me tell you about Spec Explorer. The story illustrates how theoretical work on ASMs had profound practical impact.

In 1998, Microsoft Research (MSR) invited me to create a group and apply the ASM method. My first hire was Wolfram Schulte, one of the most talented people in the ASM community in Germany. Unlike me, he had industrial experience. We built a wonderful Foundations of Software Engineering group. In a few years the group built an ASM-based tool, Spec Explorer that allowed us to write high-level executable specifications and test them against programs. Upper management, including and especially Bill Gates, liked the tool. But it seemed impossible to get the developers to use the tool because it required nontrivial training. Spec Explorer was an advanced but niche tool, rich in features, beloved by testers and those who appreciated formal methods.

Then the European Union came to our rescue ☺. In the early 2000s, the EU reprimanded Microsoft—and for good reason. While outside developers were confined to official Windows interfaces, Microsoft's own products could tap into undocumented internal protocols that offered privileged access to the core of the operating system. This asymmetry made meaningful competition on the Windows platform nearly impossible[2].

The EU demanded change. Specifically, it required Microsoft to produce *high-level, executable* specifications of the internal Windows protocols that its own products used. Microsoft lacked comprehensive, precise, high-level executable specifications. Word documents and ad hoc specs weren't sufficient.

This created a moment of crisis. And then, unexpectedly, Spec Explorer was thrust into the spotlight. It could produce high-level executable specs using state machines and C# annotations. It could generate test suites and simulate behaviors. It was suitable for model checking and detecting inconsistencies.

The Windows Division picked up Spec Explorer to model protocols. The tool was "dumbed down" a bit for broader adoption — many industrial teams want minimal knobs. It was also strengthened and productized to scale to the thousands of pages of protocol documentation eventually published.

[2]The European Commission's antitrust case against Microsoft culminated in a 2004 decision (Case T-201/04).

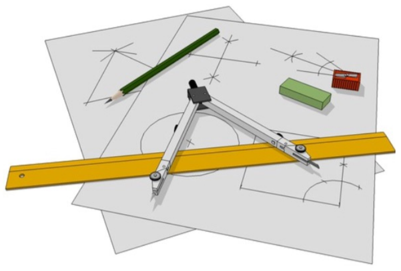
It became an industrial-strength workhorse. Teams worked tirelessly to generate models and produce the documentation needed to satisfy the EU's demands. The specifications were released as part of the Microsoft Open Specifications Promise. Microsoft paid significant fines (over €1.6 billion across multiple years). The case marked a major shift toward transparency and openness, especially regarding protocols and APIs.

Ultimately, it was, however, a Pyrrhic victory for the EU. While Microsoft did produce thousands of pages of protocol specifications and did improve transparency, the underlying technology landscape was shifting. The relevance of those protocols was already beginning to fade. Web services, cross-platform frameworks, and cloud computing were quickly rendering the entire issue obsolete.

Still, for a brief and intense period, formal modeling and executable specifications were at the center of one of the most consequential regulatory battles in software history—and Spec Explorer played a key role in bridging the gap between legal compliance and technical execution.

4 Examples

`Q:` You said that classical algorithms are deterministic, but the tangent algorithm is not. It is a ruler-and-compass algorithm that comes from antiquity; it can't be more classical.



Credit: Wikipedia [23]

`A:` This is a matter of definition. The notion of algorithm was elucidated in logic work of the 1930s–1950s. It is these algorithms that I call classical. They are deterministic. The notion of nondeterministic algorithms was formally introduced by Stephen Cook only in 1971 [5] (and independently by Levin in 197 [17], with a slightly different terminology), though Michael Rabin and Dana Scott introduced nondeterministic finite automata in 1959 [18].

By the way, in classical geometry, the tangent algorithm is not considered nondeterministic. Both solutions are considered equally valid, and the choice is left to the user.

Q: I've read Church's and Turing's thesis papers, [4] and [22], and I don't recall them using the word "algorithm."

A: You are right. The term algorithm (or algorism) existed earlier, but it referred to performing calculation using Arabic numerals. Logicians used terms like "effective method", "mechanical procedure", and "rule of calculation." It wasn't until the 1950s–60s, with the rise of computer science, that "algorithm" took on its current meaning: a finite, unambiguous, effective procedure for solving a problem.

Q: What ASM is behaviorally equivalent to the tangent algorithm?

A: Let me start from afar. Often, especially in small examples, the form of ASMs from [11] is used. It is an implicit iteration of a generic one-step rule *R* and is given just by *R*. The most common convention is that *R* is executed repeatedly until, if ever, you reach a fixed-point state *X*, so that, in state *X*, executing *R* leaves the state unchanged.

This convention is not appropriate for interactive algorithms. Executing *R* in a fixed-point state isn't innocent if *R* involves any interaction with the environment because this interaction is visible externally. Fortunately, the three axioms of [11] imply that the halting condition of a classical algorithm can be expressed by a Boolean-valued term. Accordingly, the program of a classical — in the sense of classical algorithms – ASM can be given in the form

```
do until $H$
  $R$
```

where *R* is a generic step rule and *H* a halting condition. For example, consider the following version of Euclid's algorithm for computing the greatest common divisor $d = GCD(a, b)$ of two nonnegative integers $a \geq b$.

```
while $b > 0$ do
   $(a := b) \parallel (b := a \bmod b)$;          (2)
$d := a$
```

In this case, the program of a simulating ASM could be

```
do until $d = a$
   if $b > 0$ then $(a := b) \parallel (b := a \bmod b)$
   else $d := a$
```

where it is presumed that *d* is initially undefined (or has a value, like −1, that *a* cannot possibly have).

Q: Typically, when you convert a while loop into a do-until loop, you just negate the condition.

A: Since we insist that the do-until loop is the whole program, it needs to “suck in” the assignment $d := a$ and thus becomes one step longer.

Q: Implicit iteration makes good sense in this case because the step rule makes it clear when to stop

A: Indeed, if a program (i) merely computes the value of a term t that involves no oracle queries and (ii) at the final step executes an assignment $o := t$ where o is an output variable, then the obvious halting condition is $o = t$ and the do-until loop can be made implicit. In the case of Euclid’s algorithm, the program of the simulating ASM can be given by the generic step rule

```
if $b > 0$ then $(a := b) \parallel (b := a \bmod b)$
else $d := a$
```

Q: Let’s return to the tangent algorithm. I’d expect that the program of an ASM that naturally simulates (1) would be similar to (1).

A: This is true in a sense, but the only ASM commands used to form a generic step rule in [11] are assignments, conditionals, and parallel compositions. Sequentiality within a step is handled by conditionals and sequentiality of different steps by iteration.

To deal with geometric objects, we use a restricted incidence relation Inc of type Point $\times$ Circle $\to$ Boolean, and we need some program variables and function symbols:

- program variables r, s of type `Point`, D of type `Circle`, and T of type `Line`,
- binary function symbols Cl, L, M with domain type `Point` $\times$ `Point` and range types `Circle, Line`, and `Point` respectively, and
- a binary function symbol I of domain type `Circle` $\times$ `Circle` and range type `Point`.

The intended meaning of the binary functions is as follows. Let x, y be distinct points and A, B distinct intersecting circles. Then

- $Cl(x, y)$ is the circle through y with center x.
- $L(x, y)$ is the line through x and y.
- $M(x, y)$ is the midpoint between x and y.
- I is an oracle function. $I(A, B)$ is a point in the intersection of the two circles.

Making the do-until loop implicit, we can mimic the structure of (1).

```
$r := M(p, q) \parallel$
if $r = M(p, q)$ then $D := C(r, q) \parallel$
if $D = C(r, q)$ then $s := I(C, D) \parallel$
if $\mathtt{Inc}(s, C) \wedge \mathtt{Inc}(s, D)$ then $T := L(q, s)$.
```
(3)

Q: Why isn't the guard $s = I(C, D)$ in the last line?

A: Because we cannot guarantee that the second oracle call $I(C, D)$ will give the same result as the previous one. (In [2], we have a convention that, within a step, the same queries have the same anwer.)

By the way, in the case of the tangent algorithm, we can do better:

$$T := L(q, I(A, C(M(p, q), q))).$$

Q: I'd like to see an example where you start with a probabilistic algorithm, say with the Rabin Primality Test.

A: Ok, let me first recall what it is all about.

Problem: Given an integer n determine with sufficiently high probability whether n is prime.

Algorithm: Let a, i be integer variables initialized to 1, k a positive integer constant, prime a Boolean variable initialized to true, and Random a probabilistic oracle that, given integers $b < c$, selects a number in the segment $[b, c]$ according to the uniform probability distribution.

```
do until prime = false or $i > k$
  choose a random integer $a$ such that $2 \le a \le n - 2$;
  if $a^{n-1} \ne 1$ then prime := false
  else increment $i$ by 1
```

The probability of a false positive — n is composite but prime retains value true — decreases exponentially with k. So the desired precision can be given by the value of k.

ASM. The program of a simulating ASM could be

```
do until (prime = false) $\vee$ $(i > k)$
  $a :=$ Random$(2, n - 2)$ || $i := i + 1$
  if $a^{n-1} \bmod n \ne 1$ then prime:=false
```

Q: You said that classical/sequential algorithms are not parallel, and you keep using parallel composition in the examples.

A: In [11], parallelism is subject to the following constraints.

- It is used only within a step, and
- the number of components is a priori bounded.

Q: I thought that a single step results in at most one state changing action. Parallel actions within a single step seem to contradict that.

A: Well, even Turing machines permit these two parallel actions during a single step: changing the control state and moving the head.

5 On interactive classical algorithms

Q: You say that nondeterministic algorithms are naturally interactive. Deterministic algorithms also could be interactive.

A: Deterministic algorithms are typically interactive, though interaction is often implicit. For example, algorithm (2) doesn't have code for computing the modulo function and therefore interaction is necessary. But interaction is more problematic in the case of nondeterministic algorithms, as we saw above when we discussed (3).

Q: Have you tried to extend the formalization in [11] to allow algorithms that query possibly-nondeterministic oracles?

A: We did. The results were reported in lengthy publications [2] and [3], probably too lengthy and involved for much of the intended audience.

Q: Why so lengthy?

A: Those papers included detailed motivations, discussions of alternatives, explanations, detailed proofs. Also, they explicitly allowed viewing the operating system as part of the environment and viewing our algorithm as an agent in a distributed computation.

Anyway, currently Andreas Blass and I are finishing a much shorter version of [2] with many improvements and simplifications. The axioms are the three axioms of [11] somewhat expanded.

Q: Give me a simple example of axiom expansion.

A: The simplest example is related to the Sequential Time Postulate.

Q: I guess you just replace the transition function with a transition relation?

A: This would not be enough. There may be steps of the algorithm that transform a state X to a state X' but have different interactions with the oracles. The steps are different but a transition relation would not distinguish them.

Q: I see. Interaction is, at least in principle, externally observable, and so the two steps exhibit different behavior.

A: Correct.

Q: Does the interactive-classical framework support quantum computing?

A: It was in fact our work on quantum circuit algorithms [13] that prompted the revision of [2]. Quantum circuits have quantum-transformation gates (unitary gates and measurement gates) but no code to perform them. Accordingly, we need oracles for those quantum transformations. Quantum circuit algorithms are naturally interactive classical/traditional algorithms.

References

[1] Andreas Blass and Yuri Gurevich, "Abstract state machines capture parallel algorithms," *ACM Transactions on Computational Logic* 4:4 (Oct. 2003) 578–651 with "Correction and Extensions" in 9:3 (June 2008) article 19

[2] Andreas Blass and Yuri Gurevich, "Ordinary interactive small-step algorithms", *ACM Transactions on Computational Logic*, part I in 7:2 (April 2006), pages 363–419, parts II and III in 8:3 (July 2007) articles 15 and 16

[3] Andreas Blass, Yuri Gurevich, Dean Rosenzweig and Benjamin Rossman "Interactive small-step algorithms," parts I and II, *Logical Methods in Computer Science* 3:4 (2007) articles 3 and 4

[4] Alonzo Church, "An unsolvable problem of elementary number theory", *American Journal of Mathematics* 58 (1936), 345–363

[5] Stephen Cook, "The complexity of theorem-proving procedures," Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC) 1971 151—158

[6] Robin O. Gandy, "Church's thesis and principles for mechanisms", In "The Kleene Symposium" (eds. J. Barwise et al.), North-Holland 1980 123–148.

[7] Nachum Dershowitz and Yuri Gurevich, "A natural axiomatization of computability and proof of Church's thesis," *Bulletin of Symbolic Logic* 14:3 (2008) 299–350

[8] Andreas Glausch and Wolfgang Reisig, "An ASM-characterization of a class of distributed algorithms," *Lecture Notes in Computer Science* 5115, Springer, `https://doi.org/10.1007/978-3-642-11447-2_4`

[9] Yuri Gurevich, "Evolving algebras: An introductory tutorial," *Bulletin of EATCS* 43 (1991) 264–284. Also (slightly revised) in "Current Trends in Theoretical Computer Science: Essays and Tutorials" (eds. G. Rozenberg and A. Salomaa), World Scientific 1993 266–292

[10] Yuri Gurevich, "Evolving algebra 1993: Lipari guide," in "Specification and Validation Methods" (ed. E. Börger), Oxford University Press 1995 9–36, `https://arxiv.org/pdf/1808.06255`

[11] Yuri Gurevich, “Sequential abstract state machines capture sequential algorithms,” *ACM Transactions on Computational Logic* 1:1 (2000), 77–111

[12] Yuri Gurevich, “Unconstrained Church-Turing thesis cannot possibly be true,” *Bulletin of EATCS* 127 (2019), `https://arxiv.org/pdf/2002.03145`

[13] Yuri Gurevich and Andreas Blass, “Software science view on quantum circuit algorithms,” *Information and Computation* 292 (2023) article 105024, `https://arxiv.org/pdf/2209.13731`

[14] Yuri Gurevich and Jim Huggins, “The semantics of the C programming language,” Springer Lecture Notes in Computer Science 702 (1993) 274–308.

[15] Stephen C. Kleene, “Introduction to metamathematics,” Wolters-Noordhoff Publishing and North-Holland Publishing Company, 1971 (First published by D. Van Nostrand Company in 1952.)

[16] Andrey N. Kolmogorov, “On the notion of algorithm,” *Uspekhi Matematicheskikh Nauk* (in Russian) vol. 8 issue 4(56) (1953) p. 175, `https://www.mathnet.ru/rus/rm/v8/i4/p173`

[17] Leonid Levin, “Universal search problems,” *Problems of Information Transmission* (in Russian). 9:3 (1973) 115–116

[18] Michael Rabin and Dana Scott, “Finite automata and their decision problems,” in *IBM Journal of Research and Development* 3:2 (1959) 114–125

[19] Hartley Rogers, Jr. “Theory of recursive functions and effective computability,” McGraw-Hill 1967

[20] Joseph R. Shoenfield, “Recursion theory,” Springer Verlag 1993

[21] StackExchange contributors, “What would it mean to disprove Church-Turing thesis?” Theoretical Computer Science StackExchange, Asked August 17, 2010, `https://cstheory.stackexchange.com/questions/88/`

[22] Alan M. Turing, “On computable numbers, with an application to the Entscheidungsproblem,” *Proceedings of the London Mathematical Society*, ser. 2 vol. 42 parts 3 and 4, 1936, 230–265. Corrigenda in vol. 43 (1937) 544–546.

[23] Wikipedia, “Straightedge and compass construction,” © CC BY-SA 4.0, seen on May 15, 2025