A NATURAL AXIOMATIZATION OF CHURCH'S THESIS

NACHUM DERSHOWITZ AND YURI GUREVICH

Abstract. The Abstract State Machine Thesis asserts that every classical algorithm is behaviorally equivalent to an abstract state machine. This thesis has been shown to follow from three natural postulates about algorithmic computation. Here, we prove that augmenting those postulates with an additional requirement regarding basic operations implies Church's Thesis, namely, that the only numeric functions that can be calculated by effective means are the recursive ones (which are the same, extensionally, as the Turing-computable numeric functions). In particular, this gives a natural axiomatization of Church's Thesis, as Gödel and others suggested may be possible.

We can write down some axioms about computable functions
which most people would agree are evidently true.
It might be possible to prove Church's Thesis from such axioms.
—Joseph Shoenfield (1993)

1. Introduction: Effectivity

In the beginning of the twentieth century, Hilbert famously introduced fundamental questions of decidability to mathematics:

> **[Problem] 10. Determination of the solvability of a Diophantine equation.** Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.[1] [28]
>
> The Entscheidungsproblem [decision problem for first-order logic] is solved when we know a procedure that allows for any given logical expression to decide by finitely many operations its validity or satisfiability.... The Entscheidungsproblem must be considered the main problem of mathematical logic.... The solution of the Entscheidungsproblem is of fundamental significance for the theory

Date: July 10, 2007.

2000 *Mathematics Subject Classification.* 03D10.

Key words and phrases. Abstract state machines, Church's Thesis, Turing.

First author's research was supported in part by the Israel Science Foundation under grant no. 250/05.

[1]"**10. Entscheidung der Lösbarkeit einer Diophantischen Gleichung.** Eine *Diophantische* Gleichung mit irgend welchen Unbekannten und mit ganzen rationalen Zahlencoefficienten sei vorgelegt: man soll ein Verfahren angeben, nach welchem sich mittelst einer endlichen Anzahl von Operationen entscheiden läßt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist."

> of all domains whose propositions could be developed on the basis of a finite number of axioms.[2] [29, p. 73ff.]

Hilbert was looking for well-defined procedures that would solve each instance of a problem, positively or negatively, by applying a finite number of operations. He did not have a formal notion of which operations would be reasonable in this context and which not, but clearly he was looking for operations that could be carried out by a mathematician acting mechanically, sans ingenuity. "We assume that we have the capacity to name things by signs, that we can recognize them again. With these signs we can then carry out operations that are analogous to those of arithmetic and that obey analogous laws" (Hilbert, quoted in [69]).

In 1936, Church suggested that the recursive functions, which had been defined by Gödel earlier that decade (based on a suggestion of Herbrand's), adequately capture the intended concept of finite procedure. He wrote [12, pp. 346, 356]:

> The purpose of the present paper is to propose a definition of effective calculability which is thought to correspond satisfactorily to the somewhat vague intuitive notion....
> We now define the notion ... of an *effectively calculable* function of positive integers by identifying it with the notion of a recursive function of positive integers (or of a λ-definable function of positive integers).

Church identified "the commonly used term 'effectively calculable'" [13, p. 40] in reference to a function, that is, the existence of "an algorithm" for computing the value of that function for any arguments [12, p. 356], with the specific requirement that there be recursion equations by means of which the evaluation of the function can be effected. Only with such a formalization of effectivity in hand could one prove "absolute" undecidability results, namely, that no algorithmic solution whatsoever exists for some particular problem—like the Entscheidungsproblem—as Church set out to show in his papers.

Church was roundly criticized by Post [52, p. 105] for hiding a debatable formalization of "effective calculability" behind a *definition*:

> The work done by Church and others carries this identification considerably beyond the working hypothesis stage. But to mask this identification under a definition hides the fact that a fundamental discovery in the limitations of mathematicizing power of Homo Sapiens has been made and blinds us to the need of its continual verification.

For Post [54, p. 418], it "is not a matter of mathematical proof but of psychological analysis of the mental processes involved in combinatory mathematical processes".

When Church subsequently learned of Turing's work,[3] he conceded that Turing's machines have "the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately" [14, p. 43]. He

[2]"Das Entscheidungsproblem ist gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheindung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt. Das Entscheidungsproblem muss als das Hauptproblem der mathematischen Logik bezeichnet werden.... Die Lösung des Entscheidungsproblems ist für die Theorie aller Gebiete, deren Sätze überhaupt einer logischen Entwickelbarkeit aus endlich vielen Axiomen fähig sind, von grundsätzlicher Wichtigkeit."

[3]Church was Turing's official Ph.D. advisor at Princeton.

responded to Post, saying: "To define effectiveness as computability by an arbitrary machine, subject to restrictions of finiteness, would seem an adequate representation of the ordinary notion" [15].

A few years later, Church's student, Kleene, reformulated Church's contention that the recursive functions and the effective numeric functions are one and the same as a "thesis" [32, p. 60], [33, p. 300]:[4]

> **[Church's] Thesis I.** *Every effectively calculable function (effectively decidable predicate) is general recursive.*

Again, "effective" is meant in the "vague intuitive" sense of computable by humans acting in an algorithmic fashion. To be more inclusive, one can allow functions to be partial, as Kleene subsequently did [33, p. 332]:[5]

> **[Church's] Thesis I†.** *Every partial function which is effectively calculable (in the sense that there is an algorithm by which its value can be calculated for every n-tuple belonging to its range of definition) is . . . partial recursive.*[6]

Though Kleene spoke of this thesis as unprovable ("Since our original notion of effective calculability . . . is a somewhat vague intuitive one, the thesis cannot be proved" [33, p. 317]), he did present evidence in its favor [33, Chaps. XII–XIII]. Three main lines of argument have been adduced in support of Church's Thesis (already in [33, pp. 319–323]):

(1) In years of experience, all the many effective computational models that have been investigated (starting with the lambda calculus and continuing on down to the latest programming languages) have been shown to compute only partial recursive functions.
(2) A vast number of computational models and a multitude of variants, all yield the exact same class of functions. "The proposed characterizations of Turing and of Kleene, as well as those of Church, Post, Markov, and certain others, were all shown to be equivalent" (Rogers' *Basic Result* [56, pp. 18–19]).
(3) Turing's analysis of "the sorts of operations which a human computer could perform, working according to preassigned instructions" showed that these can be simulated by his machines [33, p. 321].

The first, "heuristic" argument is relatively unconvincing ("heuristic" is Kleene's word). History is full of examples of delayed discoveries. Aristotelian and Newtonian mechanics lasted much longer than the seventy years that have elapsed since Church proposed identifying effectiveness with recursiveness, but still those physical theories were eventually found deficient.

The empirical second argument, from "confluence" of models, is also weak, and has been deemed so by Kreisel [40, p. 144] and Mendelson [47, p. 228, n. 4]. Clearly the notion captured by these equivalent models is a robust one, but there still could

[4]The converse, namely, that all recursive functions are effectively computable, is almost universally held. (Goodstein is an exception [23, n. 29]; Kalmár is not [3, n. 10].) It can be proved (e.g. [34, p. 300]), though some (e.g. [20]) contend that such an argument, inasmuch as it, too, involves the informal notion of effectiveness, should not be designated a "proof". Cf. Gödel's assertion [24, p. 44] that primitive recursive functions "can be computed by a finite procedure".

[5]"Gödel points out that the precise notion of mechanical procedures is brought out clearly by machines producing partial rather than general recursive functions." Reported in [78, p. 84].

[6]The omitted word is "potentially", which is nowadays left unstated but understood.

be a class of effective algorithms not captured by it. As Kreisel [40, p. 144] put it: "What excludes the case of a *systematic* error?"

The third argument is by far the strongest, so strong, in fact, that Gödel [25, p. 168] thought the idea "that this really is the correct definition of mechanical computability was established beyond any doubt by Turing". See, also, Gandy [23, p. 72] and Soare [74] in this connection. Though the grand sweep of Turing's argument is overwhelming, there remain weaknesses when it comes to details. For example, the universality of Turing's supposition [76] that "the number of states of mind which need be taken into account is finite" is debatable. In any event, it would be hard to reduce Turing's analysis to a few general axioms.

Subsequent models of computation did not add much force to Turing's arguments, with the possible exception of Kolmogorov's model [38, 39], which, according to Leonid Levin [personal communication],[7] was inspired by an analysis of computation in physical space-time. But one can only guess at the analysis of computations that was in Kolmogorov's head. Kolmogorov machines, and other variants of the pointer machine, do provide greater fidelity to algorithmic behavior than do Turing machines; see [6]. But Kolmogorov's published work does not delve into philosophical motivations for, or implications of, his model. In any event, an algorithm need not fit the constraints on states of Kolmogorov's machines. (See Section 7.)

Hence, it remains of real importance to provide a small number of convincing postulates in support of Church's Thesis. Indeed, Gödel has been reported (by Church in a letter to Kleene cited by Davis in [18]) to have believed "that it might be possible ... to state a set of axioms which would embody the generally accepted properties of [effective calculability], and to do something on that basis". As explained by Shoenfield (and partially quoted in the opening tag line) [66, p. 26]:

> It may seem that it is impossible to give a proof of Church's Thesis. However, this is not necessarily the case.... In other words, we can write down some axioms about computable functions which most people would agree are evidently true. It might be possible to prove Church's Thesis from such axioms.... However, despite strenuous efforts, no one has succeeded in doing this (although some interesting partial results have been obtained).

Kalmar [30] (more recently, [20]) argued against provability of the thesis, while Gandy [22] and Mendelson [47, 48] (along with [62, 64, 42, 68, 3]) argued in favor of the possibility of axiomatizing effectivity. Kreisel described the discovery of "evident axioms about constructive functions" as "one of the really important open problems" [40] and "one of the more feasible problems at the present time" [41].

We propose just such an axiomatization in the sections that follow. We demonstrate that, under certain very natural hypotheses regarding algorithmic activity, called the "Sequential Postulates" [27], Church's Thesis is in fact provable. In brief, the postulates say the following about algorithms:

I. *An algorithm determines a sequence of "computational" states for each valid input.*
II. *The states of a computational sequence can be arbitrary structures.*
III. *The transitions from state to state in computational sequences are governed by some finite description.*

[7]Levin was Kolmogorov's student.

The "ASM Thesis" (described in Section 2) asserts that all (deterministic, sequential) algorithms do indeed satisfy these reasonable hypotheses. To ensure that an algorithm is not endowed from the outset with uncomputable oracles, we add the following postulate:

IV. *Only basic (computable) operations are available initially.*

We show that Church's Thesis provably follows from these four postulates. Thus, to the extent that one might entertain the notion that there exist non-recursive effective functions, one must reject one or more of these postulates.[8]

In the next section, we formulate the three Sequential Postulates (I, II, III) rigorously and motivate each of them. In Section 3, we recall the definition of abstract state machines, and the fact that they emulate any algorithm obeying those postulates. Then, in Section 4, we turn Church's Thesis into a precise mathematical statement and show how it is implied by the ASM Thesis, plus Postulate IV. In other words, the fact that only the recursive functions can be calculated by effective means follows provably from our four postulates. In the same way, as shown in Section 5, it follows that relative effectiveness (modulo oracles) and relative recursiveness are equivalent.

Church supplied examples to argue that a decision problem in a non-numeric domain could also "be interpreted as a problem in elementary number theory", since properties in other domains "can be described in number-theoretic terms" [12, p. 345]. Accordingly, in Section 6, we extend our analysis to deal with such algorithms that manipulate additional objects, besides numbers, like strings of symbols.

Unlike Turing's analysis [76], and its subsequent generalizations [38, 39, 22, 67, 68, 72, 71], our axioms of effective computation are, at the same time, both formal and generic. They are generic, in that they apply to computations with arbitrary states and arbitrary programmable transitions. These issues are discussed in the concluding section.

For more on the history of Church's Thesis, see the article by his student, Davis [18].

2. Stepwise Effectivity

Church was striving to characterize the numerical functions that are algorithmically computable. The question is how does one know that all possible algorithms have been characterized, or as Post already phrased the problem in 1921 [54]: one needs to capture "all the possible ways in which the human mind could set up finite processes".

Rogers, a student of Church's, starts his classic book [56, pp. 1–2] with the following elaboration on the informal notion of "effective procedure" (italics in original):

> Roughly speaking, an algorithm is a clerical (i.e., deterministic, bookkeeping) procedure which can be applied to any of a certain class of symbolic *inputs* and which will eventually yield, for each

[8]For one well-known example of the claim that there is an "effective" mode of reasoning that computes a non-recursive function, see Lucas [44]: "One can feel confident without having an effective method within the meaning given to effective—i.e. programmable into a Turing machine.... It is not necessary that all reasoning must in this sense be effective. And in the sense in which reasoning might be necessarily effective, effectiveness does not imply computability."

> such input, a corresponding symbolic *output*. An example of an algorithm is the usual procedure given in elementary calculus for differentiating polynomials. . . .
>
> Several features of the informal notion of algorithm appear to be essential. We describe them in approximate and intuitive terms.
> *1. *An algorithm is given as a set of instructions of finite size. . . .*
> *2. *There is a computing agent, usually human, which can react to the instructions and carry out the computation.*
> *3. *There are facilities for making, storing, and retrieving steps in a computation.*
> *4. *Let P be a set of instructions as in* *1 *and L be a computing agent as in* *2. *Then L reacts to P in such a way that, for any given input, the computation is carried out in a discrete stepwise fashion, without use of continuous methods or analogue devices.*
> *5. *L reacts to P in such a way that a computation is carried forward deterministically, without resort to random methods or devices, e.g., dice.*
>
> Virtually all mathematicians would agree that features *1 to *5, although inexactly stated, are inherent in the idea of algorithm.

In what follows, we formalize these considerations of finite program and stepwise deterministic computation (without delineating the rôles of program P and agent L).

2.1. **Sequentiality.** We begin by characterizing computations, in general. Computation, as opposed to the behavior of a physical process, is usually conceived of as a sequence of discrete computational steps.[9,10]

[9]Moschovakis [50, p. 919] claims that "algorithms are recursive definitions while machines model implementations, a special kind of algorithms". We beg to differ. In fact, in addition to a least fixed point solution, a recursive definition has a unique "optimal" (maximally consistent) fixed point [61], which (though not necessarily computable) could be taken as the intended semantics of the definition. Nevertheless, if one accepts Moschovakis's point of view, then it is the "implementation" of algorithms that we have set out to characterize. In [5], it is argued that such a machine-independent "implementation" does not reduce the level of abstraction.

[10]Turing was explicitly interested in formalizing "discrete" machines, not continuous processes [77]: "The nervous system is certainly not a discrete-state machine. A small error in the information about the size of a nervous impulse impinging on a neuron, may make a large difference to the size of the outgoing impulse. It may be argued that, this being so, one cannot expect to be able to mimic the behaviour of the nervous system with a discrete-state system. It is true that a discrete-state machine must be different from a continuous machine. But if we adhere to the conditions of the imitation game, the interrogator will not be able to take any advantage of this difference." A first attempt at characterizing analogue computation is [49].

This is what Kolmogorov presumably had in mind when he presented his view of algorithms in 1953 [38]:[11]

> We start with the following obvious ideas concerning algorithms:
>
> 1) An algorithm Γ being applied to any "input" (="initial state") A which belongs to some set (the "domain" of the algorithm) gives a "solution" (="final state") B.
> 2) An algorithmic process splits into separate steps of limited complexity; each step consists of an "immediate transformation" of the state S obtained up to this moment into the state $S^* = \Omega_\Gamma(S)$.
> 3) The process transforming $A^0 = A$ into $A^1 = \Omega_\Gamma(A^0)$, then A^1 into $A^2 = \Omega_\Gamma(A^1)$, then A^2 into $A^3 = \Omega_\Gamma(A^2)$, etc. is continued until the next step is impossible (i.e. the operator Ω_Γ is undefined on the current state) or a signal indicating the appearance of the "solution" is received. It is possible, however, that this process of transformations would never stop (if we get no signal at all).
> 4) The immediate transformation of S into $S^* = \Omega_\Gamma(S)$ is based only on information about the limited "active part" of S and affects this part only.

Much earlier, in 1922, Behmann [2, p. 166] expressed the stepwise nature of algorithmic activity by saying (cited in [79]):[12]

> A completely determined general [set of] instructions shall be exhibited, according to which the correctness or falsity of an arbitrary given claim, which can be formulated with purely logical means, can be decided after a finite number of steps.

This is also what Kleene envisioned when he wrote [36, pp. 16–17] (emphasis in the original):

> Such a method is given by a set of rules or instructions, describing a [decision] procedure that works as follows. *After* the procedure has been described, if we select *any* question from the class, the procedure will then tell us how to perform successive steps, so that after a finite number of them we will have the answer to the question selected. . . . After our performing any step to which the procedure has led us, the rules or instructions will *either* enable us to recognize that now we have the answer before us and read it off, *or else* that

[11] Cf. Knuth in 1966 [37]: "Algorithms are concepts which have existence apart from any programming language. . . . I believe algorithms were present long before Turing et al. formulated them, just as the concept of the number 'two' was in existence long before the writers of first grade textbooks and other mathematical logicians gave it a certain precise definition. . . . A computational method comprises a set Q (finite or infinite) of 'states', containing a subset X of 'inputs' and a subset Y of 'outputs'; and a function F from Q into itself. (These quantities are usually also restricted to be finitely definable, in some sense that corresponds to what human beings can comprehend.) . . . In this way we can divorce abstract algorithms from particular programs that represent them."

[12] "Es soll eine ganz bestimmte allgemeine Vorschrift angegeben werden, die über die Richtigkeit oder Falschheit einer beliebig vorgelegten mit rein logischen Mitteln darstellbaren Behauptung nach einer endlichen Anzahl von Schritten zu entscheiden gestattet."

> we do not yet have the answer before us, in which case they will tell us what steps to perform next.

Furthermore, we view computation as proceeding *deterministically.* As Rosser, also a student of Church, puts it [57]:

> "Effective method" is used here in the rather special sense of a method each step of which is precisely determined and which is certain to produce the answer in a finite number of steps.... An effective method of solving certain sets of problems exists if one can build a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer.

Shoenfield adds [66, p. 107], "A method must be *mechanical*.... Methods which involve chance procedures are excluded; ... methods which involve magic are excluded; ... methods which require insight are excluded."

Computations may, therefore, be formalized as a *(deterministic) state-transition system (STS)*, comprising a set of *states* S, a subset I of which are *initial*, and a (partial) *transition function* $\rightsquigarrow$ on states, which determines the next-state relation, as in Kolmogorov's description. States with no "next" state, namely $O = \{\beta \in S \mid \neg\exists\gamma.\beta \rightsquigarrow \gamma\}$, will be, for us, *terminal states.*[13]

This understanding of computation is encapsulated as follows:

Postulate I (Sequential Time)**.** *An algorithm is a state-transition system. Its transitions are partial functions.*

Continuous processes, nondeterministic transitions, and nonprocedural specifications are thereby excluded.[14]

Our insistence on a unique next state is not limiting, since classical algorithms, of the sort Church was considering, appeal at most to "don't care" nondeterministic choice—one that does not affect the final result, and which can be determinized. Just as it matters not whether one sums a list of decimal numbers by adding columns of digits from top to bottom or from bottom to top, so, too, such underspecified algorithms can be made deterministic by fixing some order.[15] Segments of the evaluation of a recursive function, for example, can proceed in a nondeterministic fashion, or even in parallel; all the same, it was obvious to everyone that recursive functions *can* be computed sequentially (see, for instance, [32, p. 45] and [65, p. 109]). The important point is that the final result is independent of the precise order of evaluation [32, p. 44]. For this reason, we may restrict our attention to fully deterministic algorithms.

2.2. **Abstractness.**

Specific models of computation work with specific data structures. For example, Turing designed a model that works with tapes, though he

[13]For Kolmogorov [38], terminal states somehow "signal" their appearance. In the original definition of ASMs [26], the transition function is always total and the output is extracted from a state in $O = \{\beta \in S \mid \beta \rightsquigarrow \beta\}$. This difference is of no significance.

[14]Turing's (human) computers operate sequentially when computing functions, though he did envision *choice-machines* that wait for an "arbitrary choice ... by an external operator", before continuing (in an exploration of proofs, say).

[15]The ASM Thesis has been extended to admit the more essential use of nondeterminism in modern distributed computations; see [7] for an in-depth treatment.

explained why such a one-dimensional medium suffices for what is normally carried out by people in two dimensions [76]: "I think that it will be agreed that the two-dimensional character of paper is no essential of computation." Kolmogorov-Uspensky's model [39], and later variants of the "pointer machine" (e.g. [58]), use graphs as a more free-form representation of state. Gandy [22] suggests hereditarily finite sets as a generic data structure for the same purpose.

Since we are interested here in characterizing arbitrary algorithms, we ought not limit the form of states a priori. Accordingly, we let the states of our state-transition systems be first-order structures [75], first-order structures being the most general thing mathematicians have in their arsenal for representing discrete states. It will simplify matters if the relations (predicates) of a first-order structure are viewed as truth-valued functions, and constants as nullary functions.[16] Structures, like our states, whose vocabulary does not include relation symbols are called *algebras* (in the universal algebra sense). The justification for postulating that algebras are appropriate for capturing algorithmic states is the enormous experience of mathematicians who have faithfully and transparently represented every kind of static mathematical reality as a first-order structure.

We shall refer to state-transition systems with algebraic states as *abstract transition systems (ATSs).* Without any loss of generality, we may assume that transitions preserve the domain (base set, carrier) of the structures, taking the union of differing domains (which may be a class, not a set), if need be. We may also safely assume that the states of any particular ATS all share the same fixed vocabulary, again taking the union of differing vocabularies as the common vocabulary, should that be necessary.

Additionally, we assume that the classes of states and initial states are closed under isomorphism, and that transitions commute with isomorphisms (so that isomorphic states transition to isomorphic states). Closure under isomorphism is justified by the understanding that such structures are mere representations of isomorphism types; there should be no more reality to a domain element than what is observable from the structure. This isomorphism constraint is what makes states "abstract". It reflects the fact that an ATS works at a fixed level of abstraction, and representations do not matter. The relevance of insisting on closure under isomorphism was underscored by Gandy [22, p. 128]. For more discussion, see [27, Sect. 4.6].

Usually, the states of an ATS include some static functions that are present in initial states and are never changed by transitions. The values of any other, "defined" functions, however, are dynamic and may change from state to state. Of course, each state must contain all the data required by the algorithm for making the next step.

To sum up, we have:

Postulate II (Abstract State)**.** *States are algebras. All states share the same vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain of states. Transitions and isomorphisms commute.*

[16]Neither constants, nor functions, are "constant", because their value may vary from state to state, which is what would make it awkward were we to talk about "variable constants".

Infinitary operations are thereby excluded, since the vocabulary is first-order, but—then again—operations that are not finitary cannot be expected to be evaluatable in a single algorithmic step. Extending the vocabulary when transitioning from one state to the next, in the manner in which speakers of a natural language might continually invent new names for bigger and bigger numbers, is also precluded, but—as indicated earlier—one may instead include all such names in the state-vocabulary from the outset. Algorithms working with higher-order structures are not precluded, since the domain may include sets and higher-order functions.

Post, according to his testimony in [54, pp. 420, 426–429], had the following intuitions about abstract representations of computational states in 1922:

> We are ... to regard our symbols as without properties except that of permanence, distinguishability and that of being part of certain symbol-complexes.
>
> We ... give what is at least a first approximation to a definitive solution of the difficulty of finding a natural normal form for symbolic representation....
>
> We ... assume [symbolic representations] to be finite and we might say discrete.... Each symbolization can be considered to consist of a finite number of unanalysable parts (unanalysable from the standpoint of the symbolization) these parts having certain properties and certain relations with each other.... The ways in which these parts can be related will be assumed to be specified for the whole system of symbolizations.... The number of these elementary properties and relations used is finite and ... there is a certain specific finite number of elements in each relation....
>
> The symbol-complexes are completely determined by specifying all the properties and relations of [their] parts.... Each complex of the system can be completely described [by a conjunction of relations]....
>
> Due to discreteness and finiteness we would thus have a finite sequence of symbol-complexes representing the various stages in the method.

2.3. **Boundedness.** So far, nothing we have said makes the behavior of a transition system effective. For effectivity, it must be possible to express the rules for going from state to state in some *finite* fashion. Kleene stresses this point repeatedly:

> An algorithm in our sense must be fully and finitely described before any particular question to which it is applied is selected. When the question has been selected, all steps must then be predetermined and performable without any exercise of ingenuity or mathematical invention by the person doing the computing. [34, pp. 240–241n.]

> The notion of an "effective calculation procedure" or "algorithm" (for which I believe Church's thesis) involves its being possible to convey a complete description of the effective procedure or algorithm by a finite communication, in advance of performing computations in accordance with it. [35, p. 493]

> An algorithm is a *finitely* described procedure.... In performing the steps, we simply follow the instructions like robots; no ingenuity

> or mathematical invention is required of us. Such methods as I have described ... have been called "algorithms". [36, p. 17]

Turing analyzed the need for transitions to depend on only a finite segment of the state, for his model, as follows [76, Sect. 9]:

> 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. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols.

This finiteness requirement is expressed in more general terms by Kolmogorov and Uspensky [39, pp. 6, 16]:

> The mathematical notion of Algorithm has to preserve two properties. ...
> 1. The computational operations are carried out in discrete steps, where every step only uses a bounded part of the results of all preceding operations.
> 2. The unboundedness of memory is only quantitative: i.e., we allow an unbounded number of elements to be accumulated, but they are drawn from a finite set of types, and the relations that connect them have limited complexity. ...
>
> It seems plausible to us that an arbitrary algorithmic process satisfies our definition of algorithms. We would like to emphasize that we are talking not about a reduction of an arbitrary algorithm to an algorithm in the sense of our definition, but that every algorithm essentially satisfies the proposed definition.

To achieve this for arbitrary abstract transition systems, we demand the following:

Postulate III (Bounded Exploration)**.** *Transitions are determined by a fixed finite "glossary" of "critical" terms. That is, there exists some finite set of (ground) terms over the vocabulary of the states, such that states that agree on the values of these glossary terms, also agree on all next-step state changes.*

Since a state α is a structure, it assigns a value $[\![t]\!]_\alpha \in \text{Dom } \alpha$ to each ground term t over its vocabulary. Let

$$\Delta(\alpha) = \{f(\bar{a}) := b \mid \alpha \rightsquigarrow \beta, a_i \in \text{Dom } \alpha,\ [\![f]\!]_\alpha(\bar{a}) \neq [\![f]\!]_\beta(\bar{a}) = b\}$$

be the set of "updates" (that is, tuples $\langle f, \bar{a}, b\rangle$, written—for clarity—as "assignments" $f(\bar{a}) := b$) to the graphs of functions f that transpire in a transition out of a state α. Bounded exploration demands that $\Delta(\alpha) = \Delta(\alpha')$ whenever $[\![t]\!]_\alpha = [\![t]\!]_{\alpha'}$ holds for all critical terms t in the glossary. This is what ensures that the step-by-step behavior of the procedure is effective, since it implies that the algorithm can be described by a finite text [27, p. 90]. See [27] for further details.

Infinite programs as well as individual steps that require examination of unboundedly many parts of the state, or which update unboundedly many values in one swoop, are precluded by this postulate. For example, a transition cannot be governed by an "instruction" like

$$n := 2^{3^{\cdot^{\cdot^{\cdot^{n}}}}} ,$$

as it refers to unboundedly many terms (depending on the value of n). One would need, instead, a new operation for such an exponential tower. Nor can the transition be something like

$$n := 3 \underbrace{\uparrow\uparrow \cdots \uparrow}_{n \text{ times}} n ,$$

where the arrow is Knuth's "uparrow" notation for iterated operations.[17] This instruction is also unbounded, as it is really a schema for infinitely many conditional assignments, each involving a different operation:

$$\begin{array}{l} \quad\vdots \\ \textbf{if } n = 2 \textbf{ then } n := 3 \uparrow\uparrow 2 \\ \textbf{if } n = 3 \textbf{ then } n := 3 \uparrow\uparrow\uparrow 3 \\ \textbf{if } n = 4 \textbf{ then } n := 3 \uparrow\uparrow\uparrow\uparrow 4 \\ \quad\vdots \end{array}$$

To obtain the equivalent effect would require some form of iteration involving a long sequence of state transitions, or a new operation $a \uparrow^n b$.

With Bounded Exploration, an algorithm computes in "steps of limited complexity", as demanded by Kolmogorov [38] (quoted above). This postulate thereby answers Kolmogorov's implicit question: What does it mean to bound the complexity of each individual step?

3. Abstract State Machines

An *abstract state machine*, or *ASM*, is a state-transition system in which algebraic states store the values of constants and graphs of functions, and each transition updates a finite number of elements of the current state. ASMs capture the notion that each step of an algorithm performs a bounded amount of work, whatever domain it operates over.

Definition 3.1 (ASM [27]). An *abstract state machine (ASM)* is given by:

- a set of algebraic states S, sharing a vocabulary V,
- a set $I \subseteq S$ of initial states, and
- a *program* P, consisting of finitely many *(guarded) updates*, each taking the form

$$\textbf{if } p \textbf{ then } t := u ,$$

  for ground terms t and u over V and conjunction p of equalities and disequalities between ground terms.

[17]A single $\uparrow$ is ordinary exponentiation, $\uparrow\uparrow$ is tetration (repeated exponentiation), $\uparrow\uparrow\uparrow$ is "pentration" (repeated tetration), and so on.

Program P defines a state transition $\alpha \rightsquigarrow \beta$, for states α and β sharing vocabularies and domains, as follows:

$$[\![f(\bar{a})]\!]_\beta = \begin{cases} [\![u]\!]_\alpha & \text{if } [\![p]\!]_\alpha = T \wedge [\![\bar{s}]\!]_\alpha = \bar{a} \\ & \text{for some update } \textbf{if } p \textbf{ then } f(\bar{s}) := u \\ [\![f(\bar{a})]\!]_\alpha & \text{otherwise ,} \end{cases}$$

where the valuation is extended to tuples $\bar{s}$ and to conditions p in the obvious way.[18] A terminal state is reached when none of the update guards p are true. The updates must be "consistent", meaning that different updates with true guards may never disagree as to the value to be assigned to any $f(\bar{a})$.

Any process that satisfies the three postulates of the previous section, collectively referred to as the *Sequential Postulates*, provably behaves just like some ASM:

Theorem 3.2 (ASM Theorem [27])**.** *For every process satisfying the Sequential Postulates, there is an abstract state machine in the same vocabulary (and with the same sets of states and initial states) that emulates it.*

We use the term "emulate" for *step-by-step* simulation.[19] The proof is based in large part on the ability to use critical terms to express all transitions of abstract (isomorphism-closed classes of) states.

This emulation is effective, in the sense that the abstract state machine provides an effective means of computing each state of a computation sequence from its predecessor, provided the latter is finitely representable.

The following thesis stakes a stronger claim than the ASM Theorem, in that it entails that any algorithmic computation whatsoever can, in fact, be naturally and faithfully emulated by some ASM:

(Sequential) Abstract State Machine (ASM) Thesis. *Every relatively-effective procedure satisfies the Sequential Postulates.*

As throughout this paper, the term "effective procedure" refers to an ordinary, traditional mathematical algorithm, one that proceeds in a well-defined step-by-step fashion. We add the modifier "relatively", since procedures are allowed to apply *arbitrary* operations. This notion includes: (1) the classical algorithm for greatest common divisor—which Euclid applied to both rational and irrational values, and which can be applied more generally to Euclidean rings; (2) the ancient "rectangular array" method for solving linear equations (from the two-millennia old Chinese classic, *Jiuzhang suanshu* [31]); and (3) the very similar method of Gaussian elimination, even when the field (or division ring) over which it is applied is unspecified. On the other hand, it excludes underspecified methods (like, "Guess a solution to a system of linear equalities"), and non-algorithms ("Try all numbers to see whether or not there is a solution to a system of linear equalities" or "... to a Diophantine equation"). It is also meant to exclude nondeterministic methods (like "pivot on any non-zero element"), randomized algorithms (like multiplying

[18]A more liberal and convenient language for ASMs is provided in [27], but this simple form suffices for our purposes.

[19]"**Emulate.** To duplicate the functions of one system with a different system, so that the second system appears to behave like the first system. Note: For example, a computer emulates another, different computer by accepting the same data, executing the same programs, and achieving the same results." American National Standards Institute [1].

by a random matrix prior to performing Gaussian elimination), probabilistic methods (like Rabin's algorithm for testing primality), modern distributed processes (like Internet routing), or massively parallel DNA computations (for the travelling salesman problem, say).

Up to this point, we have established conditions under which an algorithm is effective to the extent that the functions in the initial state are. In addition, it is traditionally required that states have constructive representation; see, for example, Markov [45]. "We are," Kleene writes [35, p. 493], "dealing with discrete objects (the arguments and the result included) – it is digital, not analog, computing." Indeed, if all initial states of an ATS are finitely-representable objects, then Bounded Exploration ensures that all subsequent states also are, and the ASM formalism effectively computes the ensuing sequence of states, until a terminal state is obtained—if ever. But, the Sequential Postulates do not, in and of themselves, guarantee that an algorithm computes a *computable* function, since they do not prevent initial states from being pre-endowed with non-computable functionalities. This issue is addressed in the next two sections.

4. Arithmetical Effectivity

Since we are interested in the computation of functions, we may suppose that the vocabulary of an ATS includes (nullary) symbols Out for the output and $In_1, \ldots, In_n$, for $n \geq 0$ input values. Furthermore, we should insist that there is exactly one input state for each n-tuple of input values. Then, an ATS may be said to *compute* the following partial function over the domain(s) of its states:

$$\{([\![In_1]\!]_\alpha, \ldots, [\![In_n]\!]_\alpha) \mapsto [\![Out]\!]_\beta \mid \alpha \in I,\ \beta \in O,\ \alpha \leadsto^* \beta\}\ ,$$

where I and O are the input and terminal states, respectively, of the system, and $\leadsto^*$ is the transitive closure of its transition function $\leadsto$. This input-output relation is a partial function, since $\alpha \leadsto^* \beta \in O$ is a partial function, by virtue of $\leadsto$ being undefined for terminal states.

Church was interested in formalizing numerical algorithms, that is, algorithms that apply arithmetic operations to numbers.

Definition 4.1 (Arithmetical State)**.** The domain of an *arithmetical state* includes the natural numbers $\mathbf{N}$, as well as the Boolean truth values, T and F, and some value $\bot$ signifying "undefined". Its operations include some or all of the "grade school" operations of arithmetic, namely zero (0), successor (+ 1), addition (+), subtraction (−), multiplication (·), quotient (÷), equality (=), and inequality (<), as well as logical constants and standard operations for the Booleans. (So that these functions are all total, let $m - n = 0$ when $n > m$ and let $n \div 0 = \bot$ for all n.) Besides symbols for these operations, the vocabulary of an arithmetical state may have various symbols for dynamic functions and constants.

When all dynamic operations, *other than the inputs* In_i, are completely undefined—that is, are assigned the value $\bot$ at every point of their graphs—we say that an arithmetic state is *blank*.

The choice of basic functions is somewhat flexible, as we will see. But these "grade school" operations are what one typically has in mind. Menabrea,[20] in his 1842 description of Babbage's Analytical Engine, wrote [46]:

[20]Luigi Federico Menabrea, an engineer, later became prime minister of Italy.

> We must limit ourselves to admitting that the first four operations of arithmetic, that is addition, subtraction, multiplication and division, can be performed in a direct manner through the intervention of the machine. The machine is thence capable of performing every species of numerical calculation, for all such calculations ultimately resolve themselves into the four operations we have just named.

Babbage's design also included a conditional branch on zero; see, for example, [23].

The dynamic functions act as "variables" in the programming sense; they are intended to be updated by the algorithm in the course of a computation.

To capture the fact that Church's Thesis is dealing specifically with numeric calculations, we need the following additional postulate:

Postulate IV (Arithmetical State)**.** *Initial states are arithmetical and blank. There is one input state for any given input values.*

Accordingly, we are interested in the following class of machines:

Definition 4.2 (Arithmetical ASM)**.** An *arithmetical ASM* is an ASM satisfying the Arithmetical State Postulate (IV).

The above postulate will be considerably weakened in Section 6 to allow richer domains than the purely numeric.

Our goal now is to characterize everything that is effectively computable from such a starting point. To begin with, since transitions do not change the domain, vocabulary, or static operations, we obviously have the following:

Proposition 4.3. *All states of an arithmetical ASM are arithmetical.*

Theorem 4.4. *A numeric function is partial recursive if and only if it is computable by an arithmetical ASM.*

Proof. It is well-known that counter machines can compute any partial recursive function. And any counter machine (which only uses $0, +1, -1, =0$) can be directly emulated by an arithmetical ASM, with nullary symbols for each counter (including the inputs In_i), plus another "program counter" to keep track of the current counter-machine instruction. The ASM instructions take the following forms:

```
if $p = \bot$ then $p := 0$ (initialize program counter);
if $c = \bot$ then $c := 0$ (initialize non-input counter);
if $p = i$ then $c := c + 1$ (increment counter);
if $p = i$ then $c := c - 1$ (decrement counter);
if $p = i$ then $p := i + 1$ (move to next instruction);
if $p = i \wedge c = 0$ then $p := j$ (branch on zero);
if $p = i \wedge c \neq 0$ then $p := j$ (branch on non-zero);
```

where p is the program counter (initially $\bot$), c is any of the other counters, and i and j are (instruction) numbers.[21]

On the other hand, it is also clear that any arithmetical ASM can be programmed in a standard programming language, by storing the current non-$\bot$ values of those constants and functions appearing in the glossary and interpreting the ASM's conditional updates step-by-step. In Lisp, for example, one can simply maintain an associative list (an "environment") that lists the values (other than $\bot$) of all dynamic variables and defined functions as location-value pairs $\langle (f, a_1, \ldots, a_n), b \rangle$,

[21]For details of counter-machine ASMs, consult [8].

where $f \in V$ is dynamic and $a_i, b \in \mathbf{N} \cup \{T, F\}$. Each step of the ASM involves looking for, and using, the values in the list to evaluate all the expressions in the ASM program's updates and prepending all updated values to that list. (Values closer to the head of the list take precedence; anything not in the list is undefined and should result in nontermination.) Such programs, of course, can compute only partial recursive functions.[22] $\square$

It can be seen from the proof, and the fact that three counters suffice to compute all recursive functions,[23] that it would be enough if states had just zero, successor, equality, and a handful of constants for all recursive functions to be computable. (Predecessor can be computed from zero, successor, and equality.) In fact, the (partial) functions computed by all arithmetical ASMs with $n \geq 0$ inputs plus three additional nullary symbols (including *Out*) are precisely the (partial) recursive functions of arity n.

It follows from the above theorem that:

Corollary 4.5. *Every numeric algorithm satisfying the Sequential Postulates and provided initially with only basic arithmetic is partial recursive.*

Proof. By the ASM Theorem, every such algorithm can be emulated by an ASM whose initial states are provided only with the basic arithmetic operations. By the above theorem, such an ASM computes a partial recursive function. $\square$

In other words, Church's Thesis is a consequence of the ASM Thesis. It is provably true for numerical algorithms satisfying the Sequential and Arithmetical Postulates. Call such an algorithm, *arithmetical*. Then, the above corollary, rephrased, is precisely what we have set out to establish, namely:

Church's Thesis. *Any numeric (partial) function computed by an arithmetical algorithm is (partial) recursive.*

5. Relative Effectivity

Church provided some justifications for his claim that "every function, an algorithm for the calculation of the values of which exists, is effectively calculable" by means of recursion [12, p. 357]. But, as Sieg [68, p. 78] explains, his argument lacked an analysis of the recursiveness of the individual steps in the algorithm. (See [73, p. 291].) So, as Shoenfield [66, pp. 120–121] also makes clear, it all boils down to the effectivity of the operations that are applied at each step of a computation.

We have seen in the previous section that effectiveness is guaranteed by the Sequential Postulates and the Arithmetical State Postulate. The situation portrayed by those results is maintained no matter how many additional recursive functions are permitted in arithmetical states: Bounded Exploration guarantees that each single step is in fact effective, because it allows only a bounded number of applications of those initial functions and of function values derived from them in the

[22]This implicit appeal to the effectivity of standard programming techniques (what can be programmed in any formalism can be expressed as general recursion) is sometimes also referred to as an invocation of Church's Thesis (cf. [56, pp. 20–21]), but the omitted details could be fleshed out in what amounts to no more than a programming assignment for an undergraduate course. Shoenfield [65, p. 121n.] refers to such uses of the thesis as "very convenient", but "not really essential".

[23]See [59] for the fundamental difference between a two-counter and three-counter machine.

preceding finitely many steps. Adding non-recursive functions to the initial state, on the other hand, is a different story, taken up next.

A (partial) function f is said to be *(partial) recursive relative to* a set of functions $\mathcal{B}$ if its values can be inferred by equational reasoning from a set of equations involving $\mathcal{B}$. We assume that $\mathcal{B}$ always includes zero, successor, and equality. This is equivalent to stating that f can be obtained by composition, primitive recursion, and/or minimization from $\mathcal{B}$. See [33, §63]. The ordinary recursive functions are just those that are defined relative to $\mathcal{B} = \{0,', =\}$.

Theorem 5.1. *A numeric function is partial recursive relative to "oracular" functions $\mathcal{B}$ if and only if it is computable by an ASM with domain $\mathbf{N} \cup \{T, F, \bot\}$ and initial functions $\mathcal{B}$ (containing at least zero, successor, and equality), but no other functions defined in its initial state.*

Proof. One can program the step-by-step evaluation of an expression (in terms of $\mathcal{B}$) by encoding terms as numbers in some standard fashion. So, we design an ASM with dynamic functions for needed arithmetic, plus static oracles for $\mathcal{B}$. If in the process of evaluation, one of the oracular functions is called, the ASM applies the corresponding function from $\mathcal{B}$.

On the other hand, one can write an interpreter for such ASMs, which can be programmed in any standard language, except for calls to the oracles. Such an interpreter can in turn be implemented in terms of the functions in $\mathcal{B}$, using composition, primitive recursion, and minimization. $\square$

Corollary 5.2. *The ASM Thesis implies that the only numeric functions that are algorithmically computable are those that are partial recursive with respect to the initial functions.*

We have, then, what Kleene [33, p. 332] (paraphrased) refers to as:

> **Thesis $\mathbf{I}^{*\dagger}$.** *Every partial function which is effectively calculable relative to some initial functions is partial recursive relative to those functions.*

This version of Church's Thesis follows from the Sequential Postulates alone, without Arithmetical State. The results of the previous section are obtained whenever $\mathcal{B}$ is a subset of the recursive functions.

In the development so far, all functions operate over numbers or truth values. But, one can add as many non-numerical values to the domain as one wishes to more naturally mimic human "notations" to keep track of things while in the midst of arithmetic calculations. Better yet, one can incorporate arbitrary algorithmic operations on strings of symbols, as shown in the next section.

6. Arithmetized Effectivity

Consider now algorithms that operate over larger domains than just natural numbers, domains that may include rationals, vectors, matrices, strings, lists, graphs, etc. To work with such objects, an algorithm would be provided with operations like division of rationals, vector addition, matrix multiplication, string concatenation, list sorting, or graph intersection.

Indeed, one might naturally employ such capabilities in the process of computing what is a strictly numerical function. The object domain of such an algorithm would include elements D besides numbers and truth values, and its initial states would include operations over D. But, as we are only interested in effective computations, it is necessary to limit the initial repertoire of operations to what is undeniably effective; anything more complicated should be programmed, just as sophisticated arithmetic operations are.

We adopt a generous version of this presumption.

Postulate IVb (Arithmetizability). *The operations of the initial states of an algorithm are arithmetizable. All input states have the same operations, except for input values. There is one input state for any given input values.*

To explain, let $D' = D \cup \mathbf{N} \cup \{T, F, \bot\}$ be the object domain of an algorithm and $\mathcal{F}$, its native typed operations, and suppose operations for mistyped arguments yield $\bot$. By saying that D' is *arithmetizable*, we mean that there exists an injection $\rho : D \cup \{T, F, \bot\} \to \mathbf{N}$ such that, for every function $f \in \mathcal{F}$ of arity n over D', the corresponding function $\widehat{f}$ (of arity n over $\mathbf{N}$) is recursive, where

$$\widehat{f}(\rho'(x_1), \ldots, \rho'(x_n)) = \rho'(f(x_1, \ldots, x_n) ,$$

and $\rho'(x)$ extends ρ by giving x when $x \in \mathbf{N}$ and $\rho(x)$, otherwise. This is well-defined, even though ρ' is not one-one, as long as the native operations are in fact well-typed. So, if $k \in \mathbf{N}$ is not in the range of ρ, then $\widehat{f}(\ldots k \ldots) = \rho'(f(\ldots k \ldots)) = \rho'(\bot) = \rho(\bot)$ for all f expecting an element of $D \cup \{T, F\}$ as its argument, rather than a number k. (There is no need to insist that ρ is effective in any sense, as can be seen from the proof of the theorem below.)

For example, an algorithm that uses finite strings Σ^* over some alphabet Σ would have object domain $\Sigma^* \cup \mathbf{N} \cup \{T, F, \bot\}$. To manipulate strings, Post's tag machines [53], for example, use the following all-powerful set of basic string operations:

- Read: $\Sigma^* \to \Sigma$ (read first letter).
- Delete: $\Sigma^* \to \Sigma^*$ (delete first letter).
- Add: $\Sigma^* \times \Sigma \to \Sigma^*$ (add letter to end).

It is not hard to set up a mapping from Σ^* to $\mathbf{N}$ that encodes strings as numbers in some standard fashion, turning these string operations into combinations of addition, multiplication, exponentiation, and remaindering.

As Ada Lovelace asserted in 1843 [46]:

> Many persons who are not conversant with mathematical studies imagine that because the business of [Babbage's Analytical Engine] is to give its results in numerical notation, the nature of its processes must consequently be arithmetical and numerical rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and in fact it might bring out its results in algebraical notation were provisions made accordingly.

Call an algorithm satisfying the Sequential Postulates and the above Arithmetizability Postulate, *arithmetized.* We have the following:

Theorem 6.1 (Extended Church's Thesis). *Any numeric (partial) function computed by an arithmetized algorithm is (partial) recursive.*

Of course, the special case when the object domain is not enriched ($D = \emptyset$) is what was already shown in Section 4.

Proof. By the ASM Theorem, such an algorithm is emulated by an ASM M operating over the algorithm's domain D', and computing the same numeric function g. For any such M, consider the arithmetical ASM $\widehat{M}$ that includes in its initial states all the $\widehat{f}$ corresponding to M's operations f over D'.

Recall that $\widehat{f}(\rho'(\bar{a})) = \rho'(f(\bar{a}))$, for $f \in \mathcal{F}$ and $a_1, \ldots, a_n \in D'$, where n is the arity of f, $\rho'(\bar{a})$ is short for $\rho'(a_1), \ldots, \rho'(a_n)$, and the postulated arithmetization ρ' maps elements of D into the naturals and maps each natural number to itself. For every state α of M, let the corresponding state $\widehat{\alpha}$ of $\widehat{M}$ be such that $[\![\widehat{f}(\rho'(\bar{a}))]\!]_{\widehat{\alpha}} = \rho'([\![f(\bar{a})]\!]_\alpha)$, for each function f and domain elements $\bar{a}$ of M. (This, too, is well-defined on account of the fact that M's operations are typed.) For each guarded update of M, there is a corresponding update for $\widehat{M}$, with each occurrence of a function symbol f from the vocabulary of M replaced by the corresponding symbol $\widehat{f}$ of $\widehat{M}$. This way, whenever a value b is assigned to $f(\bar{a})$ in a state α of a computation of M, the value $\rho'(b)$ is assigned to $\widehat{f}(\rho'(\bar{a}))$ in the corresponding state $\widehat{\alpha}$ of $\widehat{M}$. (Again, on account of the typing of arguments, the corresponding updates must be consistent, like those of M.) It follows by induction on the length of a computation that these updates maintain the correspondence $[\![\widehat{f}(\rho'(\bar{a}))]\!]_{\widehat{\alpha_i}} = \rho'([\![f(\bar{a})]\!]_{\alpha_i})$ between states α_i of a computation of M and states $\widehat{\alpha_i}$ of the corresponding computation of $\widehat{M}$.

In this manner, $\widehat{M}$ emulates M step-by-step, and—by the Arithmetizability Postulate—its initial states provide recursive functions only. Any ASM, like $\widehat{M}$, with only recursive functions at its disposal must compute a recursive function $\widehat{g}$ (as indicated at the beginning of Section 5). But if g is numeric, then $\widehat{g}$ is identical to g, as the extended mapping ρ' is the identity function on all of g's numeric arguments. Hence, g must likewise be recursive. □

7. Conclusion

Our goal in this work has been to remedy the situation described thus by Montague [49, p. 432]: "Discussion of Church's thesis has suffered for lack of a precise general framework within which it could be conducted." We have shown how the Sequential ASM Postulates provide just such a framework, and how Church's Thesis follows from them, plus the postulate that nothing uncomputable is given *ab initio*. We saw in the introduction that Gödel surmised that Church's Thesis may follow from appropriate axioms of computability. But, as far as we can ascertain, no complete axiomatization has previously been presented in the literature. In fact, the challenge of proving Church's Thesis is first in Shore's list of "pie-in-the-sky problems" for the twenty-first century [11].

Turing long ago dissected the essentials of computation. As Gödel [24, p. 72] commented: "Turing's work gives an analysis of the concept of 'mechanical procedure' (alias 'algorithm' or 'computation procedure' or 'finite combinatorial procedure'). This concept is shown to be equivalent with that of a Turing machine." See also Kleene in [36, p. 30]:

> Computation, theoretically considered (to be performable for all possible values of the independent variables), is idealized. Turing's analysis takes this idealized aspect of it into account. A Turing machine is like an actual digital computing machine, except that (1) it is error free..., and (2) by its access to an unlimited tape

> it is unhampered by any bound on the quantity of its storage of information or "memory".

Turing [76] stressed the finite limitations on exploration of state by an "idealized" human computer, but his analysis was on an informal level, and related only to two-dimensional symbolic manipulations.

Just as we have shown here that the recursive functions are the only numeric functions that can be computed by an effective algorithm, we could likewise have shown that Turing machines compute all effective *string* functions, by adopting some basic set of primitive string operations, like Post's, and postulating that initial states of string algorithms are endowed with nothing additional.

Kolmogorov [38] generalized Turing's analysis, insisting on the "limited complexity" of operations and on the locality of information needed to determine the next state, but he too gave no precise characterizations. In contrast, our postulates are formal and apply to arbitrary structures—without encoding states as numbers, strings, or graphs.

Gandy [22] proposed postulates for human and machine effectivity. He defined a model, "Gandy machines", whose states are described by hereditarily finite sets. Effectivity of Gandy machines is achieved by bounding the rank (depth) of states, insisting that they be unambiguously assemblable from individual "parts" of bounded size, and requiring that transitions have local causes. Gandy's ideas have been expounded by Sieg and Byrnes [72].

In contrast to all previous analyses of effectiveness, the postulates proposed here apply to transition systems with *arbitrary* states. States can hold one-dimensional tapes (as in Turing's original work [76]), two-dimensional (as extended in [67]), bounded-degree graphs (as in Kolmogorov machines [38, 39]), or bounded-rank hereditarily finite sets (as in Gandy machines [22, 72]). But, in fact, states can also be multi-dimensional grids, or unbounded-degree graphs, or sets with unbounded rank, so long as the program satisfies the Bounded Exploration Postulate.

Moreover, we place no restrictions on state transitions, other than Bounded Exploration and the respecting of isomorphism. We do not straightjacket transitions to follow any particular format, as in Turing's formalism. Transitions can take very big steps, as long as they can all be described finitely.

It is true that convoluted structures can be *represented* linearly or graphically, and that complex transitions can be *decomposed* into smaller steps. But it is not a priori evident that computational power is not increased or decreased by such representations or decompositions. Indeed, one can "effectively" compute uncomputable functions, given a sufficiently malevolent representation. See [63, 55, 9] for discussions of this crucial point. It is, therefore, imperative to deal with effectiveness on the actual level of algorithmic computation, as undertaken here.

Some of Gandy's considerations were motivated by physical limitations of *machines*, like "the finite velocity of propagation of effects and signals" [22, p. 135]. We, on the other hand, are not concerned at all with what Gandy calls "Thesis M" (or what is also called the "Physical Church-Turing Thesis"), namely, that whatever can be calculated by a *physical machine* can be computed by a Turing machine, a claim regarding limitations that physics may, or may not, place on *physical* computing devices.[24] Furthermore, our analysis is not specific to a computational model

[24]Copeland [16] argues against the all too common misconstrual of Turing as having himself asserted such a physical claim.

operating over hereditarily finite sets, but applies to arbitrary state-transition systems with arbitrary structures for states. Gandy admitted that his model cannot emulate all computations [22, p. 146]: "Despite the liberality advertised ... there is a limit to what a machine can do in a single step." Indeed, the algorithm in [19] (determining the "parity" of certain graphs) cannot be naturally encoded as hereditarily finite sets of bounded rank, as shown there. For a recent critique of Gandy, see [60]. For another set of physical postulates, satisfied by Turing machines, see [21].

We should point out that, nowadays, one deals daily with more flexible notions of algorithm, such as interactive and distributed computations. To capture such non-sequential processes and non-classical algorithms, additional postulates are required. For such a development, see [6].

We also do not address the question of the computational capabilities of the human mind, what Shagrir [60, p. 223] refers to as "The Human version of the Church-Turing Thesis" (more generally called the "AI [Artificial Intelligence] Thesis"), that (idealized) humans cannot compute any uncomputable function. Nevertheless, it does follow from our axiomatization that any state-transition mechanism that computes a non-recursive function, whether physical or biological, must violate (at least) one of the Sequential Postulates, and/or must include at least one non-recursive function in its initial states.

See [51, pp. 101–123] and [16] for discussions of these and other variants of Church's Thesis.

Finally, the question of what effectiveness means for computations over *arbitrary*, non-numerical domains is taken up in [43, 8, 10]; see also [49].

Acknowledgements

We thank Nikolaj Bjørner, Andreas Blass, Andreas Glausch, and Moshe Vardi for their perceptive comments.

[The Sliced Bread Thesis:]
"Turing machine = coolest idea since sliced bread"

Even a rabid fan of the Turing machine concept, who firmly believes the Sliced Bread Thesis, would not claim that the Sliced Bread Thesis is formalizable in ZFC (or whatever). Possibly one could come up with an axiomatic definition of "effective algorithm" that is not trivially equivalent to the definition of a Turing machine, and then one could formalize Church's Thesis and ask for a proof of it. Shoenfield worked on this for a while, I am told, but didn't get very far.

—Tim Chow, *Foundations of Mathematics (FOM) Forum* (January 12, 2004)

References

[1] ATIS Committee T1A1 Performance and Signal Processing, *Telecom Glossary 2000*, American National Standards Institute Document ANS T1.523-2001, approved 28 February 2001. Available at `http://www.atis.org/tg2k`.

[2] Heinrich Behmann, "Beitrage zur Algebra der Logik, insbesondere zum Entscheidungsproblem", *Mathematische Annalen*, vol. 86, pp. 163–229, 1922.

[3] Robert Black, "Proving Church's Thesis", *Philosophia Mathematica*, vol. 8, pp. 244–258, 2000.

[4] Andreas Blass and Yuri Gurevich, "Evolving algebras and linear time hierarchy", in: *IFIP 1994 World Computer Congress, Volume I: Technology and Foundations*, B. Pehrson and I. Simon, eds., North-Holland, Amsterdam, pp. 383–390, 1994. Available at `http://research.microsoft.com/~gurevich/Opera/110.pdf`.

[5] Andreas Blass and Yuri Gurevich, "Algorithms vs. machines", *Bulletin of the European Association for Theoretical Computer Science*, no. 77, pp. 96–118, June 2002. Reprinted in G. Paun, G. Rozenberg, and A. Salomaa, eds., *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, vol. 2: *Formal Models and Semantics*, World Scientific Publishing Company, pp. 215–236, 2004. Available at `http://research.microsoft.com/~gurevich/Opera/158.pdf`.

[6] Andreas Blass and Yuri Gurevich, "Algorithms: A quest for absolute definitions", in Adam Olszewski, Jan Wolenski, and Robert Janusz, eds., *Church's Thesis After 70 Years*, Ontos Verlag, 2006, pp. 24–57. A previous version appeared in *Bulletin of the EATCS*, vol. 81, pp. 195–225, 2003. Available at `http://research.microsoft.com/~gurevich/Opera/164.pdf`.

[7] Andreas Blass and Yuri Gurevich, "Abstract state machines capture parallel algorithms: Correction and extension", *ACM Transactions on Computation Logic*, to appear. Microsoft Research Technical Report MSR-TR-2006-137, Sept. 2006. Available at `http://www.acm.org/pubs/tocl/accepted/314gurevich.pdf`.

[8] Udi Boker and Nachum Dershowitz, "Abstract effective models", *New Developments in Computational Models: Proceedings of the First International Workshop on Developments in Computational Models (DCM 2005)* (Lisbon, Portugal, July 2005), M. Fernández and I. Mackie, eds., *Electronic Notes in Theoretical Computer Science*, vol. 135, no. 3, pp. 15-23, Feb. 2006. Available at `http://dx.doi.org/10.1016/j.entcs.2005.09.017`.

[9] Udi Boker and Nachum Dershowitz, "Comparing computational power", *Logic Journal of the IGPL*, vol. 14, no. 5, pp. 633–648, Oct. 2006. Available at `http://www.cs.tau.ac.il/~nachum/papers/ComparingComputationalPower.pdf`.

[10] Udi Boker and Nachum Dershowitz, "The Church-Turing Thesis over arbitrary domains", *Festschrift in Honor of Boris (Boaz) Trakhtenbrot*, *Lecture Notes of Computer Science*, Springer Verlag, 2007, to appear.

[11] Samuel R. Buss, Alexander A. Kechris, Anand Pillay, and Richard A. Shore, "Prospects for mathematical logic in the twenty-first century", *Bulletin of Symbolic Logic*, vol. 7, no. 2, pp. 169–196, June 2001. Available at `http://math.ucsd.edu/~sbuss/ResearchWeb/FutureOfLogic/paper.pdf`.

[12] Alonzo Church, "An unsolvable problem of elementary number theory", *American Journal of Mathematics*, vol. 58, pp. 345–363, Apr. 1936. Reprinted in [17], pp. 89–107. Available at `doi:10.2307/2371045`.

[13] Alonzo Church, "A note on the Entscheidungsproblem", *Journal of Symbolic Logic*, vol. 1, no. 1, pp. 40–41, 1936; "Correction to a Note on the Entscheidungsproblem". *Journal of Symbolic Logic*, vol. 1, no. 3, 101–102, 1936. Reprinted in [17], pp. 110–115.

[14] Alonzo Church, review of Alan Turing, "On computable numbers, with an application to the Entscheidungsproblem" (Proc. London Math. Soc., vol. 2, no. 42, pp. 230–265, 1936), *Journal of Symbolic Logic*, vol. 2, pp. 42–43, 1937.

[15] Alonzo Church, review of Emil Post, "Finite combinatory processes, Formulation I" (*The Journal of Symbolic Logic*, vol. 1, pp. 103–105, 1936), *Journal of Symbolic Logic*, vol. 2, p. 43, 1937.

[16] B. Jack Copeland, "The Church-Turing Thesis", in E. Zalta, ed., *Stanford Encyclopaedia of Philosophy*, 1996. Available at `http://plato.stanford.edu/entries/church-turing`.

[17] Martin Davis, ed., *The Undecidable: Basic Papers on Undecidable Propostions, Unsolvable Problems and Computable Functions*, Raven Press, New York, 1965.

[18] Martin Davis, "Why Gödel didn't have Church's thesis", *Information and Control*, vol. 54, pp. 3–24, July/August 1982.

[19] Anuj Dawar, David Richerby, and Benjamin Rossman, "Choiceless polynomial time, counting and the Cai-Fürer-Immerman graphs", *Proceedings of the Twelfth Workshop on Logic, Language, Information and Computation*, pp. 13–24, 2005. To appear in *Electronic Notes in Theoretical Computer Science*.

[20] Janet Folina, "Church's thesis: Prelude to a proof", *Philosophia Mathematica*, vol. 6, no. 3, pp. 302–323, 1998.

[21] Edward F. Fredkin and Tommaso Toffoli, "Conservative logic", *International Journal of Theoretical Physics*, vol. 21, no. 3/4, pp. 219–253, 1982.

[22] Robin O. Gandy, "Church's Thesis and principles for mechanisms", In: *The Kleene Symposium*, J. Barwise, D. Kaplan, H. J. Keisler, P. Suppes, A. S. Troelstra, eds., vol. 101 of *Studies in Logic and The Foundations of Mathematics*, pp. 123–148, North-Holland, 1980.

[23] Robin O. Gandy, "The confluence of ideas in 1936", in Rolf Herken, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford: Oxford University Press, pp. 55–111, 1988.

[24] Kurt Gödel, "On undecidable propositions of formal mathematical systems", in [17], pp. 41–73, 1965. Based on lecture notes from 1934.

[25] Kurt Gödel, in S. Feferman, ed., *Kurt Gödel: Collected Works*, vol. III: *Unpublished essays and lectures*, Oxford University Press, Oxford, 1995.

[26] Yuri Gurevich, "Evolving algebras 1993: Lipari guide", in: *Specification and Validation Methods*, E. Börger, ed., Oxford University Press, pp. 9–36, 1995. Available at `http://research.microsoft.com/~gurevich/Opera/103.pdf`.

[27] Yuri Gurevich, "Sequential abstract state machines capture sequential algorithms", *ACM Transactions on Computational Logic*, vol. 1, pp. 77–111, 2000. Available at `http://research.microsoft.com/~gurevich/Opera/141.pdf`.

[28] David Hilbert, "Mathematische Probleme: Vortrag, gehalten auf dem internationalen Mathematiker-Kongreß zu Paris 1900", available at `http://www.mathematik.uni-bielefeld.de/~kersten/hilbert/rede.html`.

[29] David Hilbert and Wilhelm Ackermann, *Grundzüge der theoretischen Logik*, Springer Verlag, Berlin, 1928 (in German). A translation of the second (1938) edition appeared as *Principles of Theoretical Logic*, Chelsea Publishing, 1950.

[30] László Kalmar. "An argument against the plausibility of Church's thesis". In Heyting, A. (ed.), *Constructivity in Mathematics*, pp. 201–205 Amsterdam, North-Holland, 1959.

[31] Shen Kangshen, John N. Crossley, and Anthony W. C. Lun, *The Nine Chapters on the Mathematical Art: Companion and Commentary*, Oxford University Press, Oxford, 1999.

[32] Stephen C. Kleene, "Recursive predicates and quantifiers", *Transactions of the American Mathematical Society*, vol. 53, no. 1, pp. 41–73, 1943. Reprinted in [17], pp. 255–287.

[33] Stephen C. Kleene, *Introduction to Metamathematics*, North Holland, Amsterdam, 1952.

[34] Stephen C. Kleene, *Mathematical Logic*, John Wiley & Sons, 1967.

[35] Stephen C. Kleene, "Reflections on Church's thesis", *Notre Dame Journal of Formal Logic*, vol. 28, no. 4, pp. 490–498, 1987.

[36] Stephen C. Kleene, "Turing's analysis of computability, and major applications of it", in Rolf Herken, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford University Press, New York, pp. 17–54, 1988.

[37] Donald E. Knuth, "Algorithm and program; information and data", *Communications of the ACM*, vol. 9, no. 9, p. 654, Sept. 1966.

[38] Andrei N. Kolmogorov, "O ponyatii algoritma" ("On the concept of algorithm"), *Uspekhi Matematicheskikh Nauk*, vol. 8, no. 4, pp. 175–176, 1953 (in Russian). An English translation is in: Vladimir A. Uspensky and Alexei L. Semenov, *Algorithms: Main Ideas and Applications*, Kluwer, pp. 18–19, 1993.

[39] Andrei N. Kolmogorov and Vladimir A. Uspensky, "K opredeleniu algoritma", *Uspekhi Matematicheskikh Nauk*, vol. 13, no. 4, pp. 3–28, 1958 (in Russian). An English translation is "On the definition of an algorithm", American Mathematical Society Translations, ser. II, vol. 29, 1963, pp. 217–245.

[40] Georg Kreisel, "Mathematical logic", in T. L. Saaty, ed., *Lectures in Modern Mathematics III*, Wiley and Sons, New York, pp. 95–195, 1965.

[41] Georg Kreisel, "Mathematical logic: what has it done for the philosophy of mathematics", in Ralph Schoenman, ed., *Bertrand Russell: Philosopher of the Century*, Allen & Unwin, London, pp. 201–272, 1967.

[42] Saul Kripke, "Elementary recursion theory and its applications to formal systems. Preliminary version", unpublished draft, 1996. Available at `http://www.phil.uu.nl/~jjoosten/krip/notes/pdfchomps/Kripke-LectureI-IV.pdf`.

[43] William M. Lambert, Jr., "A notion of effectiveness in arbitrary structures", *The Journal of Symbolic Logic*, vol. 33, no. 4, pp. 577–602, Dec. 1968.

[44] John R. Lucas, review of Judson C. Webb, *Mechanism, Mentalism and Metamathematics: An Essay on Finitism* (D. Reidel, Dordrecht, 1980), *The British Journal for the Philosophy of Science*, vol. 33, no. 4, pp. 441–444, Dec. 1982.

[45] Andrey A. Markov, "Teoriya algorifmov", *Transactions of the Steklov Institute of Mathematics*, vol. 42, 1954. (In Russian.) Available in English as "Theory of algorithms", American Mathematical Society Translations, ser. 2, vol. 15, pp. 1–14, and Israel Program for Scientific Translations, 1961.

[46] Luigi Federico Menabrea, "Sketch of the Analytical Engine invented by Charles Babbage (with notes upon the memoir by the translator A. A. L. [Ada Augusta, Countess of Lovelace])", in R. Taylor, ed., *Scientific Memoirs*, vol. 3, pp. 666–731, 1843. French original: "Notions sur la machine analytique de M Charles Babbage", *Bibliothèque Universelle de Genève n.s.*, vol. 41, pp. 352–376, October 1842, and in M. Campbell-Kelly, ed., *The Works of Charles Babbage*, vol. 3, pp. 62–82, Pickering, London. Available at `http://www.fourmilab.ch/babbage/sketch.html`.

[47] Elliott Mendelson, "Second thoughts about Church's thesis and mathmeatical proofs", *The Journal of Philosophy*, vol. 87, no. 5, pp. 225–233, May 1990.

[48] Elliott Mendelson, "On the impossibility of proving the 'hard-half' of Church's thesis", in Adam Olszewski, Jan Wolenski, and Robert Janusz, eds., *Church's Thesis After 70 Years*, Ontos Verlag, pp. 304–309, 2006.

[49] Richard Montague, "Towards a general theory of computability", *Synthese*, vol. 12, no. 4, pp. 429–438, Dec. 1960.

[50] Yiannis N. Moschovakis, "What is an algorithm?", in B. Engquist and W. Schmid, eds., *Mathematics Unlimited—2001 and beyond*, Springer, 2001, pp. 919–936. Available at `http://www.math.ucla.edu/~ynm/papers/eng.ps`.

[51] Piergiorgio Odifreddi, *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*, Studies in Logic and the Foundations of Mathematics, vol. 125, North-Holland, Amsterdam, 1992.

[52] Emil L. Post, "Finite combinatory processes, Formulation I", *The Journal of Symbolic Logic*, vol. 1, pp. 103–105, 1936. Reprinted in [17], pp. 289–303.

[53] Emil L. Post, "Formal reductions of the general combinatorial decision problem", *American Journal of Mathematics*, vol. 65, no. 2, pp. 197–215, Apr. 1943. Available at `doi:10.2307/2371809`.

[54] Emil L. Post, "Absolutely unsolvable problems and relatively undecidable propositions: Account of an anticipation", unpublished paper, 1941. In [17], pp. 340–433.

[55] Michael Rescorla, "Church's Thesis and the Conceptual Analysis of Computability", *Notre Dame Journal of Formal Logic*, 2007.

[56] Hartley Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.

[57] J. Barkley Rosser, "An informal exposition of proofs of Gödel's Theorem and Church's Theorem", *Journal of Symbolic Logic*, vol. 4, pp. 53–60, 1939. Reprinted in [17], pp. 223–230.

[58] Arnold Schönhage, "Storage modification machines", *SIAM Journal on Computing*, vol. 9, no. 3, pp. 490–508, August 1980.

[59] Rich Schroeppel, "A two counter machine cannot calculate 2^N", Technical Report AIM-257, A. I. Laboratory, Massachusetts Institute of Technology, May 1972. Available at `ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-257.pdf`.

[60] Oron Shagrir, "Effective computation by humans and machines", *Minds and Machines* vol. 12, no. 2, pp. 221–240, May 2002. Available at `http://edelstein.huji.ac.il/staff/shagrir/papers/Effective_computation_by_human_and_machine.pdf`.

[61] Zohar Manna and Adi Shamir, "The optimal approach to recursive programs", *Communications of the ACM*, vol. 20, no. 11, pp. 824–831, 1977.

[62] Stewart Shapiro, "Understanding Church's thesis", *Journal of Philosophical Logic*, vol. 10, pp. 353–365, 1981.

[63] Stewart Shapiro, "Acceptable notation," *Notre Dame Journal of Formal Logic*, vol. 23, pp. 14–20, 1982.

[64] Stewart Shapiro, "Understanding Church's thesis, again", *Acta Analytica*, vol. 11, pp. 59–77, August 1995.

[65] Joseph R. Shoenfield, *Mathematical Logic*, Addison-Wesley, Reading, MA, 1967.

[66] Joseph R. Shoenfield, *Recursion Theory*, vol. 1 of *Lecture Notes in Logic*, Springer Verlag, Heidelberg, New York, 1991.

[67] Wilfried Sieg, "Mechanical procedures and mathematical experiences", in: *Mathematics and Mind*, A. George, ed., Oxford University Press, pp. 71–117, 1994.

[68] Wilfried Sieg, "Step by recursive step: Church's analysis of effective computability", *Bulletin of Symbolic Logic*, vol. 3, no. 2, pp. 154–180, June 1997. Available at `www.math.ucla.edu/~asl/bsl/0302/0302-002.ps`.

[69] Wilfried Sieg, "Hilbert's programs: 1917–1922", *Bulletin of Symbolic Logic*, vol. 5, no. 1, pp. 1–44, 1999.

[70] Wilfried Sieg, "Church without dogma – Axioms for computability", submitted. Available at `http://www.hss.cmu.edu/philosophy/sieg/Church%20without%20dogma.pdf`.

[71] Wilfried Sieg and John Byrnes, "K-graph machines: generalizing Turing's machines and arguments", in: *Gödel 96: Logical Foundations of Mathematics, Computer Science, and Physics*, P. Hajek, ed., Lecture Notes in Logic, vol. 6, Springer Verlag, Berlin, pp. 98–119, 1996.

[72] Wilfried Sieg and John Byrnes, "An abstract model for parallel computations: Gandy's thesis", *The Monist*, vol. 82, no. 1, pp. 150–164, 1999.

[73] Robert I. Soare, "Computability and recursion", *Bulletin of Symbolic Logic*, vol. 2, no. 3, pp. 284–321, September 1996. Available at `http://www.math.ucla.edu/~asl/bsl/0203/0203-002.ps`.

[74] Robert I. Soare, "Computability and incomputability", unpublished lecture, *Computation and Logic in the Real World, Third Conference on Computability in Europe (CiE 2007)*, Siena, Italy, June 2007. Available at `http://www.people.cs.uchicago.edu/~soare/siena502m.pdf`.

[75] Alfred Tarski, "Der Wahrheitsbegriff in den formalisierten Sprachen", *Studia Philosophica*, vol. 1, pp. 261–405, 1936. English: J. H. Woodger, trans., "The concept of truth in formalized languages", in J. Corcoran, ed., *Logic, Semantics, Methamathematics*, Hackett, Indianapolis, pp. 152–277, 1983.

[76] Alan M. Turing, "On computable numbers, with an application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, vol. 42, pp. 230–265, 1936–37. Corrections in vol. 43 (1937), pp. 544–546. Reprinted in [17], pp. 116–154, and in Jack Copeland and Alan M. Turing, *The Essential Turing*, Oxford University Press, pp. 58–90, 2004. Available at `http://www.abelard.org/turpap2/tp2-ie.asp`.

[77] Alan M. Turing, "Computing machinery and intelligence", *Mind*, vol. 59, pp. 433–460, 1950. Reprinted in Jack Copeland and Alan M. Turing, *The Essential Turing*, Oxford University Press, pp. 433–464, 2004. Available at `http://web.comlab.ox.ac.uk/oucl/research/areas/ieg/e-library/sources/t_article.pdf`.

[78] Hao Wang, *From Mathematics to Philosophy*, Routledge and Kegan Paul, London, 1974.

[79] Richard Zach, "Completeness before Post: Bernays, Hilbert, and the development of propositional logic", *The Bulletin of Symbolic Logic*, vol. 5, no. 3, pp. 331–366, Sept. 1999. Available as `http://www.math.ucla.edu/~asl/bsl/0503/0503-003.ps`.

Contents

Microsoft Research, Redmond, WA 98052, USA and School of Computer Science, Tel Aviv University, 69978 Ramat Aviv, Israel
E-mail address: `nachum.dershowitz@cs.tau.ac.il`

Microsoft Research, Redmond, WA 98052, USA
E-mail address: `gurevich@microsoft.com`