Is Universal Computation a Myth?

Selmer Bringsjord

Abstract Akl has claimed that universal computation is a myth, and has offered a number of ingenious arguments in support of this claim, one of which features the challenge of tracking the locations of multiple, ever-moving robots on Mars. I provide what I see as a refutation of this argument; my counter-argument is based on a thesis that is less informal and more plausible than the Church-Turing Thesis, and on my own generalized variant of Kolmogorov-Uspensky machines. While I concede that it doesn't deductively follow from the success of my refutation that universal computation is, or can be, real, I conclude by pointing toward a route that I believe can vindicate the counter-claim that universal computation is specifiable, and instantiable.

1 Introduction

Selim Akl's remarkable oeuvre provides innumerable opportunities for one to write about the foundations, both formal and philosophical, of computation. For the present volume, I've seized upon a single opportunity: his ingenious and provocative "The Myth of Universal Computation" [1]. My analysis, in a further narrowing, is specifically targeted at a key argument of Akl's within this paper, a fascinating one involving the tracking of multiple robots (assumed to be) on Mars. I denote this argument as '$\mathcal{A}_{\overline{u}_{\text{TM}}}$.' Because I shall use '$\overline{u}_{\text{TM}}$' to denote the statement that no Turing machine can be a universal computer, the subscript in '$\mathcal{A}_{\overline{u}_{\text{TM}}}$' is just a convenient reminder that $\overline{u}_{\text{TM}}$ is the conclusion of this argument.

Akl's overall goal in Akl [1] is in fact much more ambitious than establishing $\overline{u}_{\text{TM}}$, for he doesn't think *any* rigorous, fixed, abstract model of computation can be

I'm indebted to Selim Akl for bringing to my attention countless stimulating ideas, only one of which I explore herein.

S. Bringsjord (✉)
Rensselaer AI and Reasoning (RAIR) Lab, Department of Computer Science, Rensselaer Polytechnic Institute (RPI), Troy, NY 12180, USA
e-mail: selmerbringsjord@gmail.com

A. Adamatzky (ed.), *Emergent Computation*, Emergence, Complexity and Computation 24, DOI 10.1007/978-3-319-46376-6_2

universal. This is made clear by Akl at the very outset of his paper, in fact in his paper's abstract. There he says this about what the paper by his lights accomplishes:

> It is shown that the concept of a Universal Computer cannot be realized. ... This result applies not only to idealized models of computation, such as the Turing Machine and the like, but also to all general-purpose computers, including existing conventional computers, as well as contemplated ones such as quantum computers [1].

It obviously follows from this quote that Akl takes himself to have shown not only that the Turing machine (TM) isn't a universal computer, but that any other candidates for the title of 'universal computer' will likewise fail to reach universality. Leveraging the notation that we have already allowed ourselves, we can hence observe that Akl sees his paper as providing a sound argument $\mathcal{A}_{\overline{u}_{\text{QC}}}$ for the conclusion $\overline{u}_{\text{QC}}$; here, 'QC' is an acronym referring to quantum computers. Indeed, letting 'C' be a variable ranging over any established class of idealized computing machines, we can safely say that Akl's ultimate goal (which he believes he has reached in the paper in question) is to establish

$$\overline{u} := \neg\exists\, C\, \overline{u}_{\text{C}};$$

and we can denote his overarching argument by '$\mathcal{A}_{\overline{u}_{\text{C}}}$.' However, again, my objective is the narrow, focused one of showing that Akl's multiple-robot argument $\mathcal{A}_{\overline{u}_{\text{TM}}}$ for $\overline{u}_{\text{TM}}$ is unsuccessful. While it doesn't follow deductively from my refutation that Akl's overarching argument $\mathcal{A}_{\overline{u}_{\text{C}}}$ is overthrown (because he gives additional arguments for $\overline{u}_{\text{TM}}$ beyond the one I target), if his other arguments for $\overline{u}_{\text{TM}}$ fail, his overarching case $\mathcal{A}_{\overline{u}_{\text{C}}}$ would fall, and hence despite his clever analysis and argumentation there may well be a form of *bona fide* universal computation. I contend, but do not prove, that my counter-argument against $\mathcal{A}_{\overline{u}_{\text{TM}}}$ can in fact be generalized into a recipe that overthrows the other arguments Akl gives against universal computation. At the end of the present chapter I suggest a logic-based route toward formalizing a form of universal computation.

My selection of Akl's paper and the specific $\mathcal{A}_{\overline{u}_{\text{TM}}}$ within it, I confess, is not without an element of selfishness, since the topics with which Akl deals in this important work are ones I too have thought a bit about. Nonetheless, as will soon be seen, our respective points of view are fundamentally different. Put with brutal brevity, I come to computation after reflecting upon the cognition of animals and persons, and from there move to the relevant logico-mathematics for modeling and computationally simulating that cognition; Akl, on the other hand, draws morals about the nature of computation after considering "de-agentized" information flowing at the mercy of time and change, in the real, physical world. (His multiple-robots-on-Mars scenario is a perfect case of his orientation in action.) We both move on from our respective starting points to consider the limits of computation, but our respective conclusions turn out to be quite different: Akl (obviously) regards $\mathcal{A}_{\overline{u}_{\text{TM}}}$ (and $\mathcal{A}_{\overline{u}_{\text{QC}}}$, and indeed $\mathcal{A}_{\overline{u}_{\text{C}}}$) to be sound; I don't. Moreover, as I've already indicated, I think that the concept of universal computation can in principle be formally defined via increasingly powerful logics, and that the concept can in fact be instantiated in our universe (in some mind sufficiently powerful to reason in these logics).

The plan for the remainder of the chapter is straightforward: The first step (Sect. 2) is to explain that Akl's understanding of the Church-Turing Thesis (CTT) is inaccurate. Next, in Sect. 3, using my analysis of CTT as a springboard, I present a different and much "safer" thesis: "Selmer's Safer Thesis," or just 'SST' for short. More accurately, the safer thesis is actually a thesis *schema*. Whereas CTT (as I shall point out) relies on the concept of *effective* computation, my safer thesis schema relies instead upon what I call *reflective-C* computation. Here, where C is again (recall above) a variable ranging over any of the established idealized frameworks for computing at the Turing level, reflective-C computation is a semi-formal description of the fully formal computation in C. With SST in hand, I next (Sect. 4) recapitulate and analyze Akl's multiple-robot argument $\mathcal{A}_{\overline{u}_{\text{TM}}}$, drawing directly from his paper to do so. Then in the next Sect. 5, I refute Akl's argument. Some concluding remarks that gesture toward a universal computer wrap up the paper (Sect. 6).

2 The Church-Turing Thesis (CTT), for Real

Our first step is to isolate and analyze what Akl takes to be the "Church-Turing Thesis" (CTT). Doing so is easy, for here is a verbatim quote from Akl [1, p. 172]:

> While fairly simple conceptually, the Turing Machine is a truly powerful model of computation. So powerful in fact, that it was believed until recently that no model more powerful than the Turing Machine can possibly exist (in other words, a model that would be able to perform computations that the Turing Machine cannot perform). This belief is captured in the following statement, known as
>
> > **Church-Turing Thesis**: Any computable function can be computed on a Turing Machine [73, 54].

Unfortunately, this is not CTT. The reason is perfectly simple and uncontroversial: It must be a particular *kind* of function that is said in the thesis to be a Turing-computable one. Church [14, p. 356] originally used the informal phrase 'effectively calculable' to label the kind of function in question. The phrase 'effectively computable' is the syntactic variant of Church's phrase that is currently used. Now, notice that Akl, in the quote immediately above, gives two citations immediately after typographically setting out his version of the thesis in question. Could it be that Akl has been led astray by the authors in question? I investigated; sure enough, this appears to be exactly what happened. For example, here is how [28, p. 209] puts it: "The Turing machine (TM) is believed to be the most general computational model that can be devised (the **Church-Turing thesis**)." This is what Akl is referring to when he offers the citation '[54].'[1]

[1] Unfortunately for Savage, the super-recursive computational models explored by Turing in his doctoral dissertation under Church (i.e. [30]) refute Savage's claim that (at least at the time of his writing) it is believed that the Turing machine is the "most general computational model." A wonderful discussion of these matters in relation to Turing's dissertation is provided in Feferman [18]; cf. Bringsjord [5].

But why do I say that Akl has been led astray? The reason, again, is simple, and quite decisive: The Church-Turing Thesis, CTT as we abbreviate it, is not what Akl says it is, because the core idea in CTT is the equivalence of what is "effectively" or "mechanically" or "algorithmically" computable, with what is Turing-computable. Hence we must be more precise and accurate.[2]

Again, the heart of CTT is the informal notion of an *algorithm*,[3] which has been nicely characterized (in traditional fashion) by Mendelson as

> an effective and completely specified procedure for solving a whole class of problems. . . . An algorithm does not require ingenuity; its application is prescribed in advance and does not depend upon any empirical or random factors [24, p. 225].

An *effectively computable* function is thus the computing of a function by an idealized "worker" or "computist" following an algorithm.[4] (Without loss of generality, we can for present purposes view all functions as taking natural numbers into natural numbers; that is, for some arbitrary f, $f : \mathbb{N} \mapsto \mathbb{N}$).

CTT also involves a more formal notion, that of a so-called *Turing-computable* function. If the formal notion is wed to a different paradigm, then we would no longer have the Church-*Turing* Thesis. For example, we could refer instead to a *recursive* function, or a register machine-computable function, etc. Mendelson employs Turing's approach, and Turing machines are what Akl focuses upon in the paper we're analyzing. A function $f : \mathbb{N} \mapsto \mathbb{N}$ is *Turing-computable* iff there exists a TM m which, starting with n on its tape (perhaps represented by n |s), leaves $f(n)$ on its tape after processing. (The details of the processing are harmlessly left aside.) Given this definition, CTT amounts to

CTT A function f is effectively computable if and only if it's Turing-computable.

Most scholars, as the reader herself is likely to know, regard CTT to be true. However, I'm not one of them. So while I have on hand a counter-argument against $\mathcal{A}_{\overline{u}_{\text{TM}}}$ that employs CTT, I certainly can't use it here. Not only that, but in a rather interesting twist, even if I was inclined to affirm CTT, I still couldn't use it as a premise in a counter-argument against $\mathcal{A}_{\overline{u}_{\text{TM}}}$. The reason is that a careful reading of Akl [1] reveals that Akl himself is quite prepared to give up CTT. In fact, he appears to hold that $\overline{u}_{\text{TM}}$ entails the falsity of CTT.

---

[2] And here I follow my own prior work, and the work of others, including those who have instructively sought to *prove* CTT. In my own case, devoted in part to arguments *against* CTT, see e.g. Bringsjord and Arkoudas [6], Bringsjord and Govindarajulu [7]; for an attempt to prove CTT, see the chapter on CTT in Smith [29]; and for a wonderful exposition of CTT and its history, including coverage of the trap of stating CTT erroneously as in the case of Savage [28], see Copeland [15].

[3] Interestingly enough, Lewis and Papadimitriou [23], the pair of authors Akl [1] draws from in order to formally characterize Turing machines, well understand that CTT asserts an equivalence between an intuitive notion of algorithm and Turing-computability, for—in a quote isolated by Akl himself—we read that CTT consists in the proposition that "the idea of a 'computational procedure' or an 'algorithm' is equivalent to the idea of a Turing Machine."

[4] Turing [31] spoke of "computists" and Post [27] of "workers," humans whose sole job was to slavishly follow explicit, excruciatingly simple instructions.

3 Selmer's "Safer" Thesis (SST)

The received view is that CTT is not only true, but unprovable.[5] The main rationale in support of this view is the claim that the concept of effective computation is too informal to allow a proof of CTT [4]. Whether or not this rationale is correct, the fact certainly remains that the "left side" of the biconditional that constitutes CTT is not formal, while the right side, which refers to the Turing-computability of a function, can be rendered formal. Turing-computability (and of course also therefore Turing-decidability, etc.) can be formalized in various ways, as Akl [1] points out. He draws heavily on Lewis and Papadimitriou [23] to recount some of the formalism in question. And we could even be more formal and rigorous about Turing-computability, since we could move from the naïve set theory of Lewis and Papadimitriou [23] to axiomatic set theory (e.g., ZFC), and laboriously build up to CTT from there. But a move to greater rigor in defining the right side of CTT would still leave the left side vague. I introduce now the promised thesis schema SST, which includes still on its left side a somewhat informal concept, but not one as intuitive and informal as effective computation. As I've said, while I can't for the reasons given above use CTT in my counter-argument against $\mathcal{A}_{\overline{u}_{\text{TM}}}$, I can, and do, use SST.

The first step toward SST is to recall that above we quantified over idealized computational schemes C to introduce $\overline{u}$. We can leverage this simple idea in order to formulate a thesis schema that is at once both much more plausible and much less informal than CTT. Instead of employing the concept of *effective computation* as in the case of CTT, SST employs the concept of *reflective-C* computation, where C here is once again functioning as a variable ranging over the space of established idealized computational schemes that are provably equivalent to that of the Turing machine. This space includes not just Turing machines (T), but also for example Post machines (P) [26], register machines (R) [16], the μ-recursive functions (M) [23], unrestricted (= Type 0) grammars (G) [25], the λ-calculus (Λ) [12, 13], and my favorite formal model of Turing-level computation that doesn't explicitly use logic and deduction, and one that is clearly the most cognitively realistic category under C, Kolmogorov-Uspensky (KU) machines (K) [21]. Each of these idealized frameworks is an acceptable instance of the general variable C that ranges over all established idealized frameworks equivalent to Turing machines; and all of these frameworks are equivalent. Hence, for instance, a function f is G-computable if and only if (iff) f is T-computable iff f is Λ-computable iff f is R-computable, and so on.

In this context, I now introduce the new concept of *reflective-C* computation. This concept is not fully formal, but it's much more formal than the very vague and intuitive concept of effective computation. To see the basic idea is this, start by bringing to mind some formal description of one of the idealized frameworks listed in the previous paragraph. For focus and to ease exposition, but without loss

[5] A few have held that CTT is provable, but they are in the extreme minority, and, joined by others, I have shown that defenders of the provability of CTT are incorrect. E.g., while Smith [29] has tried to prove CTT, see Bringsjord and Govindarajulu [7].

of generality, let's first choose T, Turing machines. Many examples of reflective-T computation can be given, and in fact many are given in the literature; here's one:

Imagine an old-fashioned railroad track that starts at a certain point and extends infinitely in one direction. Imagine as well that this track is laid upon railroad ties spaced at a regular interval, and that on the ground, bounded by two ties and two stretches of track, is a blackboard. In addition, suppose that there is a small boxcar that can roll on the track, powered by a simple mechanical lever, and a switch onboard the boxcar that controls the direction of movement (left or right on the track). The boxcar is occupied by a well-trained chimpanzee who can power the car by pushing the lever up and down, and who can through toggling of the switch move left or right. In addition, the bottom of the boxcar is hollow, so when the boxcar is positioned over a blackboard, the chimpanzee can reach down to erase symbols appearing on the blackboard, and to write symbols on the blackboard. He does so in accordance with very simple instructions. Finally, the combined ensemble of the chimp, the boxcar, and his simple tools, at any one moment, are assumed to be in any one of a particular number of finite, pre-defined configurations. Whatever a—as we shall call them—*chimp machine* can (reflective-T-compute) a Turing-machine can compute, and *vice versa*. The reason for this, in a word, is that chimp-machine computation, while intuitive, is directly reflective of T computation. And of course we didn't need to refer to chimps and boxcars. Instead, we could have referred to any number of an infinite number of other props, and we could still be depicting computation that is directly reflective of the formal Turing-machine model. The general truth in play can be elevated to the following statement:

SST_T A function f is reflective-T-computable if and only f is T-computable (= Turing-computable).

To make sure there is no misunderstanding or resistance, let me explain that we can do the same kind of trick for register machines, formal, idealized machines which are reflected by the less formal concept of *raven machines*, as I now explain.[6]

Raven machines include, first and foremost, a raven: Roger. Roger is a thoroughly obedient bird whose range of activity is highly restricted. Roger is shown in Fig. 1. You will note that he is holding something in his beak. What is it? It's a little round stone. Roger doesn't fly (at least when he is working); when we tell him to start a work session, he simply moves little round stones around, in accordance with programs that we provide to him, and he halts when we tell him to conclude a work session. More specifically, his movement of the stones is confined to moving them into and out of numbered boxes. For any given work session, we provide Roger with n boxes to start, and if his program makes reference to the number m of a box beyond the ones he intially has, Roger calls out "More," and instantly a new box numbered m appears for him to employ.[7] Raven machines consist of the combination of: programs to

[6] Here I draw upon my Bringsjord and Taylor [9], which I use to teach introductory formal logic and computability theory. Most readers will be familiar with register machines, which are elegantly and economically defined in Ebbinghaus et al. [17].

[7] Alternatively, we could imagine that Roger's call for another box results in a box from an infinite supply provided at the outset of his efforts, moving into his work area.

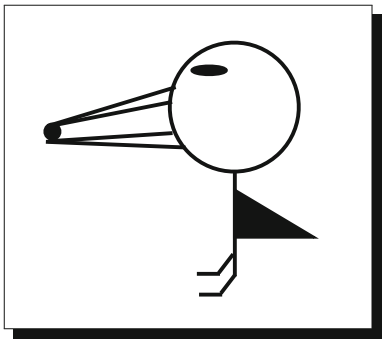


Fig. 1 Roger the raven

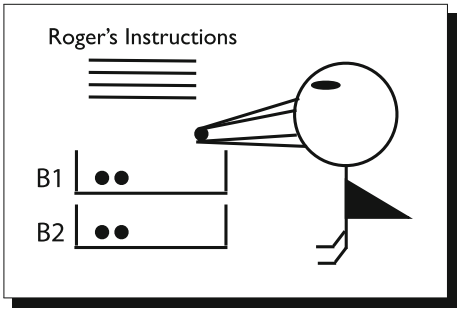


Fig. 2 Snapshot of a raven machine in operation

instruct Roger, Roger himself with his perception and action powers, and the stones and boxes. Figure 2 shows a snapshot of a raven machine during some computation.

We are interested in having raven machines compute number-theoretic functions, that is functions from $\mathbb{N}^n$ to $\mathbb{N}$. In order to enable this, we shall understand a given natural number n to correspond to n stones located in a given box. The natural number 0 will correspond to the absence of any stones; so an empty box is assumed to be holding 0. Hence Fig. 2 shows a configuration in which box B_1 holds stones representing the numeral '2,' and box B_2 holds stones representing the numeral '3.' Ordinary addition $+$ of natural numbers is of course such that

$$+ : \mathbb{N}^2 \mapsto \mathbb{N}.$$

Can Roger compute it? Yes, easily. But in order to see how, we need to specify the format of the instructions that we provide him with.

Each instruction to Roger is one of five possible types. We now define this quintet, by giving the schema for each one, and in addition an intuitive explanation of what the instruction communicates to our bird. Note that every instruction begins with a natural number l that serves as its label.

1. l SET $\mathrm{B}_i = \mathrm{B}_i - \bullet$

   This instruction tells Roger to take away one stone from box B_i. If this box happens to be empty, Roger doesn't do anything, and simply moves on to the next instruction.

2. l SET $\mathrm{B}_i = \mathrm{B}_i + \bullet$

   This instruction tells Roger to add one stone to box B_i.

3. l IF $\mathrm{B}_i = e$ THEN j ELSE k

   This instruction tells Roger that if box B_i is empty, he should shift his attention to, and follow, instruction with label j; otherwise he should move to instruction with label k.

4. l ROGER, POINT TO B_i

   This instruction tells Roger to point to box B_i, in order to inform us that this is output he wishes us to have.

5. l ROGER, HALT

   This instruction simply tells Roger to halt. In any set of instructions given to Roger (i.e., in any *raven program* given to him), there can only be one instruction of this type.

Let's now put these schemas into action, in the form of a raven program for Roger that carries out addition. In order to do that, we shall assume that box B_1's contents denotes the first of the two numbers to be added, and that box B_2's contents denotes the second. Here then is a program for addition:

0 IF $\mathrm{B}_1 = e$ THEN 5 ELSE 1
1 IF $\mathrm{B}_2 = e$ THEN 6 ELSE 2
2 SET $\mathrm{B}_2 = B_2 - \bullet$
3 SET $\mathrm{B}_1 = B_1 + \bullet$
4 IF $\mathrm{B}_2 = e$ THEN 6 ELSE 2
5 ROGER, POINT TO B_2
6 ROGER, POINT TO B_1
7 ROGER, HALT

With an initial input of $\bullet\bullet$ in box B_1, and $\bullet\bullet\bullet$ in box B_2 (i.e., an initial configuration that constitutes a request to Roger that he tells us what $2 + 3$ is), the program given here will cause Roger to point to B_1 when it contains $\bullet\bullet\bullet\bullet\bullet$, at which point he will stop.

The overall point of this account of raven machines is to flesh out and make rather obvious the proposition that raven-machine computation, while somewhat informal, is equivalent to register-machine computation ($= R$ computation). In short, raven

computation, albeit informal, is nonetheless directly reflective of register-machine computation. In addition, raven machines could be replaced by an indefinite number of schemes of the same intuitive sort, but ones featuring different animals, and/or objects other than stones to be manipulated, etc. All these schemes would preserve reflection of R computation. We hence have the general statement:

SST_R A function f is reflective-R-computable if and only f is R-computable.

Now let's spend just a bit of time on the idealized computing framework that I said above was my favorite: KU machines (or the space of idealized machines denoted by just 'K'). It's my favorite because it's the most cognitively robust model of information-processing to emerge from the mid 20th century, as far as I know. I have my own variant of the original conception that is set out in Kolmogorov and Uspenskii [21]. I have neither the time nor the space to set out either the original conception or my own formal generalization (*workbook machines*) in full detail, but I can certainly give a sufficiently detailed explanation of the latter, and while my formalization is more general that KU machines, workbook machines can serve as an adequate stand-in in the present paper for the more primitive KU-machine framework. (While KU machines are equivalent to Turing machines, workbook machines have settings that can be configured in such a way as to allow these machines to compute functions beyond the Turing Limit in the Arithmetic Hierarchy, e.g. the halting problem.) Also, because workbook machines are built from scratch to be reflected by the ordinary notebooks used by systematic human thinkers through their careers, the account that I now give of workbook machines will serve both to introduce such machines, and to do so by providing an informal correlate of workbook machines. The informal correlate is what I call *notebook machines*. Obviously, the situation is thus such that notebook machines are reflective-K machines.

A workbook machine has an associated formal language L of the type customarily used to specify the syntax of the formulae allowed in a logic, and to specify the inferential machinery by which formulae can be linked to each other. The language is composed of formulae that can be constructed according to a formal grammar of a familiar type (e.g. a BNF grammar) from a list of syntactic ingredients: variables, constants (= names), function symbols, relation symbols, quantifiers, operators (e.g., modal operators), connectives, and punctuation symbols. In addition, and this is unusual, L includes the elements necessary to precisely write expressions that are typically in meta-theory; but we can leave this aside in the present context.[8] The language may also include things necessary to tap into abstract algebra, and thereby move beyond what readers are used to seeing in standard logics to regiment typical symbolic formulae. For example, the language of a given workbook might need to be extended to allow for the precise specification of diagrams; such a language is used in the Vivid family of logics for reasoning over symbolic formulae and

[8]The formal language associated with a workbook machine for information-processing in line with logic programming would thus allow formulae in the language of horn-clause logic, and would also allow for meta-logical expressions like $\forall \mathcal{I} : \mathcal{I} \models \texttt{Ra} \leftrightarrow \texttt{Ra}$, where $\mathcal{I}$ is an interpretation from model theory.

diagrams [2]. Finally, another part of L is the machinery needed to specify proof theories and argument theories; for example, rules of inference.

We have already implicitly made use of such languages above; for example, when we defined the statement $\overline{u}$ above we made implicit use of the formal language that underlies first-order logic, which has only the two quantifiers $\exists$ and $\forall$, and no operators. But workbooks allow formal languages that are simpler or more complex than the language of first-order logic. In fact, the language for a given workbook may allow formulae that are infinitely long. Most readers will have seen grammars used to define formal languages for logics, so I spend no more time on formulae.

Workbooks are composed of *pages* that come in sequences like a conventional book of the type that people read. Pages can come in any size, as long as the size is finite. Workbooks can have arbitrarily many pages, though here let's confine our attention to books that have only a finite number of pages. What can be written on the pages that are in workbooks? The answer is that formulae in L can be written inside labeled nodes (e.g. inside ovals), the nodes can then be connected by directed arcs, and the arcs can be labeled by such things as the names for inference schemas. For convenience and clarity, the labels can be put inside their own shapes (e.g. boxes).

How does computation happen in a workbook machine? It happens when a *scribe* is given instructions for what modifications to make on a page, within a proper subset of space on the page that is the focus of attention of the scribe. As a profitable example for the reader to consider, imagine that a scribe is given instructions for how to carry out long division. The only differences between how people in the real world carry out long division (on a piece of paper or a blackboard) versus how such an algorithm gets mechanized in a workbook machine is that in the latter case each number must be encased within a labeled node, the arrangement of nodes relative to each other is enforced by arcs connecting them, and a scribe can make what would for some humans be a number of sequential actions on page in one step.

I now provide an example of computation by a scribe in an implemented workbook machine, the Slate environment for producing proofs.[9] The example is shown through two snapshots of a page in Slate. In the first snapshot (see Fig. 3), the scribe's attention is focused on the simple theorem that $0 \neq 5$ (notice that it is highlighted), to be proved from the axioms of Peano Arithmetic (PA) (shown on the left), plus some helpful definitions (shown on the right). In the next snapshot, the theorem has in one step been proved. In order to do this, the scribe has moved AXIOM 1, and cited this axiom along with the definitions for support of a provability claim (viz., that the theorem can be proved). It's very important to realize that this progress has been made in a single step, because when below I model the tracking of Akl's Mars robots it will be a key fact that in a single step the distances of multiple robots from a landmark can be computed.

I have described workbook machines, which are formal, idealized machines that subsume KU machines, by way of the less formal class of notebook machines. Notebook machines, like the chimp machines and raven machines also characterized

[9] Slate is provided with Bringsjord and Taylor [9]; an early version is described in Bringsjord et al. [10].

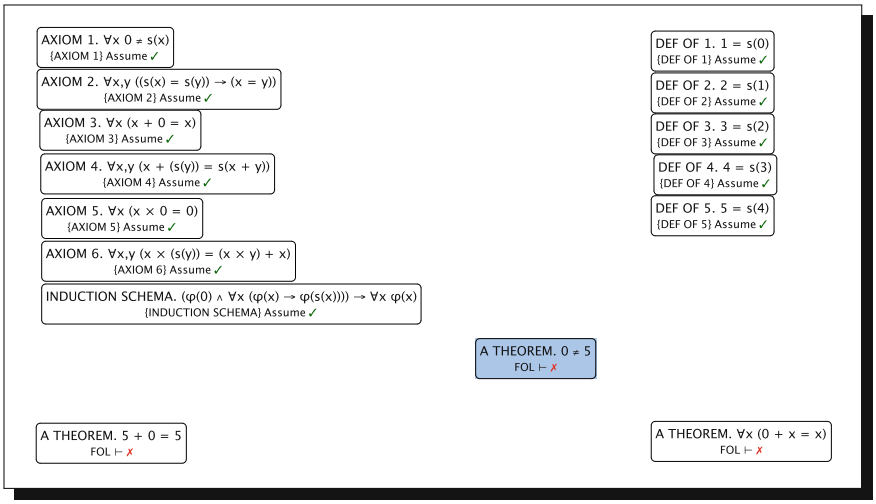


Fig. 3 Snapshot of page built in slate system as workbook machine

above, are not fully formal; however, notebook machines are clearly directly reflective of workbook machines/KU machines (although, again, the former can be set to allow super-Turing computation). In addition, instead of my own conception of notebook machines, which is based on the concept of a scribe, any number of other quasi-formal description of KU machines could be created[10]—and in all these other variants, computational equivalence between them and KU machines would be preserved. Summing up, we have:

SST_K A function f is reflective-K-computable if and only f is K-computable.

While my refutation of Akl's $\mathcal{A}_{\overline{u}_{\text{TM}}}$ can be articulated with only SST_K (see Sect. 5), there is no reason, in general, to stay at the level of only instances of SST, and a better version of my counter-argument uses SST itself, as the schema that it is. In order to move to SST itself, in its fully general schematic form, we have only to invoke again quantification over the entire space of Turing-level idealized frameworks for computation, via C. Doing so yields the following general proposition:

SST For all established idealized computational frameworks C, a function f is reflective-C-computable if and only f is C-computable.

4 Akl's Robot Argument ($\mathcal{A}_{\overline{u}_{\text{TM}}}$)

To start his argument $\mathcal{A}_{\overline{u}_{\text{TM}}}$, Akl presents a scenario involving multiple robots:

[10] Smith [29] provides an alternative.

> On the surface of Mars n robots, $R_0, R_1, \ldots, R_{n-1}$, are roaming the landscape. The itinerary of each robot is unpredictable; it depends on the prevailing conditions in the robot's environment, such as wind, temperature, visibility, terrain, obstacles, and so on. At regular intervals, each robot R_i relays its current coordinates $x_i(t) = (a_i(t), b_i(t))$ to mission control on Earth. Given the coordinates of the n robots at time t, mission control determines the distance of $R_i, 0 \leq i \leq n-1$, to a selected landmark $L(t)$ using a function F_i [1, p. 181].

A key additional aspect of the scenario, which Akl has introduced before giving the scenario described in the quote immediately above (he introduces it on p. 180), is that there is a composite n-ary function $\mathcal{F}$ that takes as its input tuple that which is returned by the F_i. Akl says (p. 180) that $\mathcal{F}$ might return for instance the sum or the minimum of the values of the F_i.

Leaving aside the exotic imaginary setting of Mars, this type of scenario is one that is quite relevant for my own laboratory, which has more than its share of robots, and which specifically often investigates the coördination of multiple robots acting simultaneously in various environments. Of course, we don't use any such low-level formalism as that used to specify Turing machines to track and reason about the diachronic attributes of robots through time. For that matter, *no one* writes sophisticated software to control and coördinate multiple robots in the language of Turing machines, certainly if the robots in question do anything cognitive.[11] Instead, my approach, and derivatively that of the lab I direct, is a logicist one; specifically, we draw from a space $\mathscr{CC}$ of *cognitive calculi* to model both cognitive and physical states of artificial agents (including robots) through time [8, 11, 19, 20]. A cognitive calculus is a highly expressive computational logic, one that in some instances subsumes so-called "BDI" logics, and in some cases includes provision for natural-language understanding and/or generation, uncertainty, and non-monotonic reasoning. In the present paper, it would be inappropriate to review in detail any of the calculi in $\mathscr{CC}$; instead, in the Sect. 5, I will make informal and rapid use of a particular cognitive calculus [2] that allows for the representation of, and reasoning over, pictorial information in human-level fashion. Importantly, the reasoning is such that single steps can comprise what humans working with paper and pencil would need to do in a number of sequential steps; recall the discussion above centering around Figs. 3 and 4. With information expressed pictorially, it turns out that the kind of rapid and real-time processing of Akl's function F_i can be accomplished in the manner he says is beyond the reach of Turing machines. But before we get to this, we of course need to have before us the remainder of Akl's $\mathcal{A}_{\overline{u}_{\text{TM}}}$, which, in his own words, runs as follows:

> A Turing Machine fails to compute all the F_i as desired. Indeed, suppose that $x_0(t)$ is read initially and placed onto the tape. It follows that $F_0(x_0(t))$ can then be computed correctly (perhaps at a later time). However, when the next variable x_1, for example, is to be read, the time variable would have changed from t to $t+1$, and we obtain $x_1(t+1)$, not $x_1(t)$. Continuing in this fashion, $x_2(t+2), x_3(t+3), \ldots, x_{n-1}(t+n-1)$ are read from the input. In [my example], by the time $x_0(t)$ is read, robots $R_1, R_2, \ldots, R_{n-1}$ would have moved away from $x_1(t), x_2(t), \ldots, x_{n-1}(t)$.

[11] Cognitive robotics is, at least in its original form, a logic-based affair; see e.g. Levesque and Lakemeyer [22].

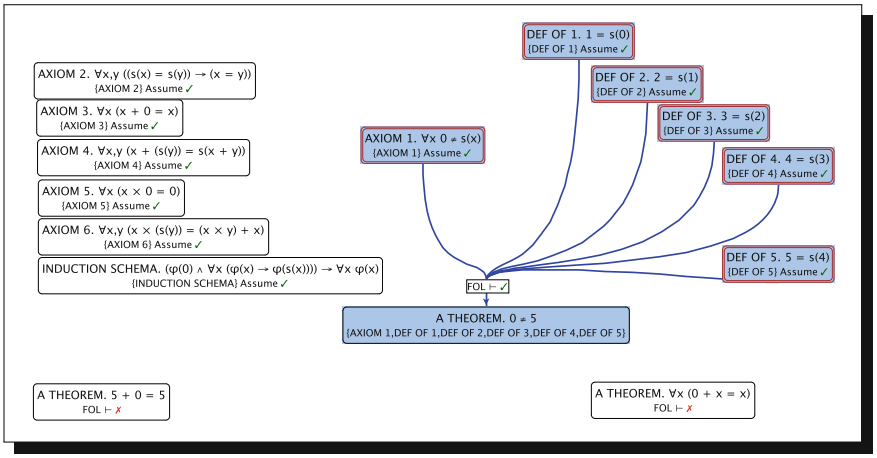


Fig. 4 Snapshot of successor page built in slate system as workbook machine

> Since the function according to which each x_i changes with time is not known, it is impossible to recover $x_i(t)$ from $x_i(t+i)$, for $i = 1, 2, \ldots, n-1$. Consequently, this approach cannot produce $F_1(x_1(t)), F_2(x_2(t)), \ldots, F_{n-1}(x_{n-1}(t))$ as required [1, p. 181].

There are some important aspects of this argument that can and should be revealed by some analysis. First, when Akl says "A Turing Machine fails to compute all the F_i as desired." he is not to be interpreted as saying only that there exists a Turing Machine that fails to compute all the F_i. The kernel of the logical shape of this trivial proposition would be

$$\exists m(TM(m) \wedge FailsCompute(m, F_i)], \tag{1}$$

whereas what Akl is claiming is (as must be the case if he is to succeed) in line with the claim $\overline{u}_{\mathrm{TM}}$, and the logical shape of his claim is the much more ambitious and much more interesting

$$\neg\exists m(TM(m) \wedge Compute(m, F_i)]. \tag{2}$$

This really is a very ambitious claim indeed. For there are a lot of Turing machines; there are overall a countably infinite number of them, of course. How does Akl know that no Turing machine in this vast space can compute F_i? His reasoning appears to be that while the configuration of the robots and the landmark at a given time t can be placed into memory (= onto a tape of a given Turing machine), there isn't enough time to compute all the distances, because by the next timepoint $t+1$ the robots have moved and the distances will be different—and to further complicate matters the environmental forces that partially determine the itineraries (I would say *plans*, and I imagine that the robots are running AI planners) of the robots aren't retrievable. But is this reasoning sound? I don't think so, and now I explain why.

5 A Refutation of the Robot Argument

Since Akl concedes F_i to be a Turing-computable function, there exists a Turing machine able to compute the distance of each robot R_j from the landmark at every timepoint t, *before* another such computation needs to occur. By definition, we are dealing with number-theoretic functions, so we are dealing with a digitization of the entire scenario. In 'the 'real world" the robots must take some time to travel to new locations, and when they have arrived at those new locations, a new configuration can be perceived, and processed afresh. So yes, I certainly agree that the solution for a Turing machine is not to be found in computation carried out, as Akl says, "later." But the solution is to be found in the fact that there exists some Turing machine $m^\star$ that computes F_i, for all relevant i, very quickly and at once, in parallel, in an *intervening* moment, *before* the robots arrive at their new locations. (Even if robot movement is staggered in time, it remains possible, in principle, to *any* configuration of any subset of the entire collection of robots to be perceived and distances to be computed at a single intervening timestep.) And having computed that, m^* can compute $\mathcal{F}(F_i)$, for all relevant i, in another intervening moment. As Akl says, the composite function $\mathcal{F}$ might return something like the sum of all the distances of each robot from the landmark, at a given timepoint.

Of course, Akl's claim is that the "intervening" activity I have described is excluded. But what excludes it? It is true that Akl can stipulate a constraint according to which there is no intervening time available to be used. But such a stipulation merely establishes Eq. 1; it doesn't establish what he needs: Eq. 2.

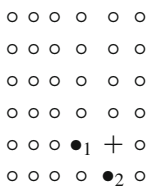
An even more severe problem for Akl is that such back-and-forth dialectic is entirely irrelevant, because there is a non-constructive way of establishing that there does exist the Turing machine $m^\star$ that can compute F_i. In fact, I now give such a non-constructive way: a formally valid argument whose conclusion is that the problem in this case, *contra* Akl, can be solved by some TM (i.e., I establish the negation of Eq. 2). (Since my argument is formally valid on any standard proof theory, I could classify my argument as an outright proof, save for the fact that "Selmer's Safer Thesis," SST, isn't itself proved in the present paper—though certainly it can be proved.) My refutation is in defense of the proposition that, relative to standard idealized computation — that is computation characterized by P, R, M, G, Λ, K, and the other frameworks that can be readily found in the literature—Turing machines are universal.

As mentioned above (Sect. 2), I can't employ CTT in any refutation of Akl's argument (since, again, I believe that, and indeed believe that I've shown that, CTT is in fact false). However, I'm able to use SST and a the particular instance of it for KU machines. Here's my argument:

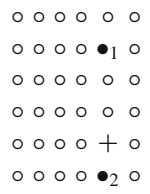
Refutation of $\mathcal{A}_{\overline{u}_{\text{TM}}}$

SST tells us that any number-theoretic function f that is reflective-C-computable is C-computable. It follows directly by universal instantiation on C that any f that is reflective-K-computable is K-computable. (Or we could of course use SST_K directly.) Where F_i is the function (informally) defined by Akl that maps location information regarding n robots to the distance of each robot from a given landmark (all indexed, of course, to a particular time t), F_i is reflective-K-computable; hence F_i is K-computable. But it's a theorem that K-computability and T-computability (= Turing-computability) are equivalent. Therefore, *contra* Akl, F_i is Turing-computable (and with it also $\mathcal{F}$). **QED**

In order to undergird this argument, I introduce a simple pictorial framework that enables us to represent snapshots of the locations of robots and the landmark on Mars. In this framework, which is grid-based, $\bullet_j$ indicates a robot at the relevant location, and $+$ indicates the location of the landmark. Each configuration of the grid corresponds to a page in a workbook machine. (I leave out messy ovals to define nodes, and explicit arcs, in order to increase readability.) Consider the following configuration:



Here there are only two robots (but there is no loss of generality). Please try to make a quick ruling as to how far each robot is from the landmark. . . . Correct, both robots are the same distance (1 unit) from the landmark. Notice that you rendered this verdict by taking in the workspace as a whole. Now here is a second configuration:



To ensure that you understand what is being depicted in this second configuration, I ask again: How far is robot R_2 from the landmark, and how far is robot R_1 from the landmark? Once again, I'm quite sure that you can see what the answer is: R_1 is 3 units away, and R_2 is 1 away. This shows that a scribe in a workbook machine could, presented with any such configuration, write down in one step, the distance for each robot R_i. This in turn shows that F_i is reflective-K-computable, since in principle

there is no reason why what you have instantly seen in the two configurations just given can't be seen in *any* such configuration, by a scribe perceiving a page in a workbook. The area of focus in any given configuration will never be too large to in principle be taken in, since it is always going to be finite, and since it will never increase from one page to the next in an expansion that grows non-recursively fast. (There are only a finite number of robots in Akl's scenario, separated from each other and the landmark by only finite distances.) This suffices to undergird the non-constructive deductive counter-argument given above.

6 Concluding Remarks on Another Path

Alert-and-astute readers know that the failure of Akl's robot argument $\mathcal{A}_{\overline{u}_{\mathrm{TM}}}$, as revealed above, is formally consistent with the statement $\overline{u}$ that there doesn't exist a universal computer. (This is a fact that Akl himself, in rebuttal, might well convey.) Hence, in the current state of the inquiry into whether or not a universal computer is a myth, we are unable to resolve the central question. After all, perhaps one of the *other* arguments given by Akl for $\overline{u}_{\mathrm{TM}}$ succeeds. However, for what it's worth, I've analyzed each of these arguments and find each to be at best inconclusive. I encourage those interested in getting to "ground truth" on $\overline{u}$ not to accept on faith this report on my analysis, but to study Akl's inventive arguments for themselves. That said, I do want to end, as promised, by introducing the reader to what I see as a better route for settling the central question. My intuition, based on initial reflection upon this other route, is that universal computation does in fact exist, and that therefore u (notice that the overline is gone) holds.

So what is the route I recommend? To see its general shape and direction, suppose, first, that *universal computation*, which we'll symbolize by the predicate U, is stipulated to be a *disjunctive* concept, one with so much in-built latitude that it ranges across all forms of information-processing, not just computation as it's systematized in standard Turing-level-and-below information processing, and indeed not just information-processing as it's formalized in the entire Arithmetic Hierarchy (of which the Turing-computable portion is only a small part). (Akl's writings never allow under U forms of information-processing beyond AH, as far as I can tell.) We therefore admit information-processing over uncountable sets. So far this is quite imprecise, of course. But the disjunction can be made precise by appeal to formal logics — as long as we countenance formal logics of more and more power, including those that exceed information-processing in AH. Let me explain, at least to a degree.

We know that to capture the behavior and power of standard Turing machines, and any rigorous form of information-processing at and below this level in AH (e.g., to harken back to the categories deployed above, register machines, the λ-calculus, KU machines, etc.), we can use standard first-order logic $\mathscr{L}_I$. This is shown in excruciating detail in traditional proofs of the undecidability of theoremhood in $\mathscr{L}_I$ (= the undecidability of the *Entscheidungsproblem*), wherein deciding theoremhood in $\mathscr{L}_I$ is (frequently) reduced to the halting problem [4]. Encapsulating, we know

that a number-theoretic function f is Turing-computable if and only if, from a suitable theory Γ, representing the operation of a standard Turing machine and initial information (including a given element a of the domain of f),

$$\Gamma \vdash \phi_{f(a)},$$

where $\phi_{f(a)}$ is the formulae expressing the value that f returns on a as an argument, and where $\vdash$ is interpreted in standard form, that is to indicate provability in some typical proof theory for $\mathscr{L}_I$. (Such a proof theory underlies the single-step inference shown in Fig. 4.)

For distinctions within the sub-space of Turing-computable functions that pertain to how much time it takes for Turing machines and automata below them to compute a relevant function (the sub-space that covers such categories as NP-complete), that too can be captured by $\mathscr{L}_I$ with suitable function symbols and relation symbols to capture time and change, number of steps in a computation, and size of input. But we also know that once we for instance move from finitary logic (and $\mathscr{L}_I$ is of course certainly finitary: all its formulae are of finite length, as are all its formal, object-level proofs) to infinitary logic, we can quickly move to information-processing that is beyond Turing machines.[12] (In parallel, we know that such a move allows us to surmount Gödelian incompleteness, since such results are based on Turing-level axiom systems in $\mathscr{L}_I$, such as Peano Arithmetic.) For a quick example, note that the "small" infinitary logic $\mathscr{L}_{\omega_1\omega}$ allows countably infinite disjunctions and conjunctions, and countably infinite proofs. Using infinitary logics, we can build up coverage of increasingly challenging functions to compute, where we express the computing of a function g in terms of what is expressible and deducible in the relevant logic, following the general recipe sketched above in the case of $\mathscr{L}_I$, where what is to be proved is that from a declarative representation of a given argument a in g's domain, $g(a)$ is what is returned. If we take this route, we can say that a universal computing framework, that is a framework to which can be accurately ascribed the relation U for universal computation, is one which, given any well-defined function g and input a, can prove $g(a)$, in either logic $\mathscr{L}_1$, or $\mathscr{L}_2$, or $\mathscr{L}_3$, or ..., where this is a progression of increasingly powerful logics. I wonder what Akl would say about this route. One thing certainly seems clear: This disjunctive, logicist route, without all that much work, would yield a precise framework on which all the challenges in the remarkably fertile and suggestive [1] can be modeled and thereby met. In fact, the more rigorous and accurate is Akl's reasoning in setting out a challenge, the easier such modeling, for some logic $\mathscr{L}_k$, becomes.

[12] Readers interested in learning more, can consult as a starting point the excellent [3].

References

1. Akl, S.: The myth of universal computation. In: Vajteršic, M., Trobec, R., Zinterhof, P., Uhl, A. (eds.) Parallel Numerics '05: Theory and Applications, University of Saltzburg, pp. 167–192. Saltzburg, Austria. https://www.cosy.sbg.ac.at/events/parnum05/book/prolog.pdf (2005)
2. Arkoudas, K., Bringsjord, S.: Vivid: an AI framework for heterogeneous problem solving. Artif. Intell. **173**(15), 1367–1405. http://kryten.mm.rpi.edu/vivid_030205.pdf (2009). (The URL http://kryten.mm.rpi.edu/vivid/vivid.pdf provides a preprint of the penultimate draft only. If for some reason it is not working, please contact either author directly by email)
3. Barwise, J.: Infinitary logics. In: Agazzi, E. (ed.) Modern Logic: A Survey, pp. 93–112. Reidel, Dordrecht, The Netherlands (1980)
4. Boolos, G.S., Burgess, J.P., Jeffrey, R.C.: Computability and Logic, 4th edn. Cambridge University Press, Cambridge, UK (2003)
5. Bringsjord, S.: Theorem: general intelligence entails creativity, assuming .... In: Besold, T., Schorlemmer, M., Smaill. A. (eds.) Computational Creativity Research: Towards Creative Machines, pp. 51–64. Atlantis/Springer, Paris, France. http://kryten.mm.rpi.edu/SB_gi_implies_creativity_061014.pdf (2015). (This is Volume 7 in *Atlantis Thinking Machines*, edited by Kühnbergwer, Kai-Uwe of the University of Osnabrück, Germany)
6. Bringsjord, S., Arkoudas, K.: On the provability, veracity, and AI-relevance of the Church-Turing thesis. In: Olszewski, A., Wolenski, J., Janusz, R. (eds.) Church's Thesis After 70 Years, pp. 66–118. Ontos Verlag, Frankfurt, Germany. http://kryten.mm.rpi.edu/ct_bringsjord_arkoudas_final.pdf (2006). (This book is in the series *Mathematical Logic*, edited by W. Pohlers, T. Scanlon, E. Schimmerling, R. Schindler, and H. Schwichtenberg)
7. Bringsjord, S., Govindarajulu, N.S.: In defense of the unprovability of the Church-Turing thesis. J. Unconv. Comput. **6**, 353–373. http://kryten.mm.rpi.edu/SB_NSG_CTTnotprovable_091510.pdf (2011). (Preprint available at the url given here)
8. Bringsjord, S., Govindarajulu, N.S.: Given the web, what is intelligence, really? Metaphilosophy **43**(4), 361–532. http://kryten.mm.rpi.edu/SB_NSG_Real_Intelligence_040912.pdf (2012). (This URL is to a preprint of the paper)
9. Bringsjord, S., Taylor, J.: Logic: A Modern Approach: Beginning Deductive Logic. Motalen, Troy, NY (2015). (This is an e-book edition of January 25 2016. The book is accompanied by the Slate software system, ISBN of 78-0-692-60734-3, and version of January 25 2016)
10. Bringsjord, S., Taylor, J., Shilliday, A., Clark, M., Arkoudas, K.: Slate: an argument-centered intelligent assistant to human reasoners. In: Grasso, F., Green, N., Kibble, R., Reed, C. (eds.) Proceedings of the 8th International Workshop on Computational Models of Natural Argument (CMNA 8), pp 1–10. University of Patras, Patras, Greece. http://kryten.mm.rpi.edu/Bringsjord_etal_Slate_cmna_crc_061708.pdf (2008)
11. Bringsjord, S., Licato, J., Govindarajulu, N., Ghosh, R., Sen, A.: Real robots that pass tests of self-consciousness. In: Proccedings of the 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2015), pp 498–504. IEEE, New York, NY. http://kryten.mm.rpi.edu/SBringsjord_etal_self-con_robots_kg4_0601151615NY.pdf (2015). (This URL goes to a preprint of the paper)
12. Church, A.: A set of postulates for the foundation of logic (Part I). Ann. Math. **33**(2), 346–366 (1932)
13. Church, A.: A set of postulates for the foundation of logic (Part II). Ann. Math. **34**, 839–864 (1933)
14. Church, A.: An unsolvable problem of elementary number theory. Am. J. Math. **58**(2), 345–363 (1936)
15. Copeland, J.: The Church-Turing thesis. In: Zalta, E. (ed.) The Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/archives/sum2015/entries/church-turing (2002)
16. Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic. Springer, New York, NY (1984)
17. Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic, 2nd edn. Springer, New York, NY (1994)

18. Feferman, S.: Turing in the land of O(Z). In: Herken, R. (ed.) The Universal Turing Machine, 2nd edn, pp. 103–134. Springer, Secaucus, NJ (1995)
19. Govindarajalulu, N.S., Bringsjord, S., Taylor, J.: Proof verification and proof discovery for relativity. Synthese **192**(7), 2077–2094 (2015)
20. Govindarajulu, N.S., Bringsjord, S.: Ethical regulation of robots must be embedded in their operating systems. In: Trappl, R. (ed.) A Construction Manual for Robots' Ethical Systems: Requirements, Methods, Implementations, pp. 85–100. Springer, Basel, Switzerland. http://kryten.mm.rpi.edu/NSG_SB_Ethical_Robots_Op_Sys_0120141500.pdf (2015)
21. Kolmogorov, A., Uspenskii, V.: On the definition of an algorithm. Uspekhi Matematicheskikh Nauk **13**(4), 3–28 (1958)
22. Levesque, H., Lakemeyer, G.: Chapter 24: cognitive robotics. In: Handbook of Knowledge Representation. Elsevier, Amsterdam, The Netherlands. http://www.cs.toronto.edu/~hector/Papers/cogrob.pdf (2007)
23. Lewis, H., Papadimitriou, C.: Elements of the Theory of Computation. Prentice Hall, Englewood Cliffs, NJ (1981)
24. Mendelson, E.: Second thoughts about Church's thesis and mathematical proofs. J. Philos. **87**(5), 225–233 (1986)
25. Partee, B., Meulen, A., Wall, R.: Mathematical Methods in Linguistics. Kluwer, Dordrecht, The Netherlands (1990)
26. Post, E.: Finite combinatory processes-formulation 1. J. Symb. Log. **1**(3), 103–105 (1936)
27. Post, E.: Recursively enumerable sets of positive integers and their decision problems. Bull. Am. Math. Soc. **50**, 284–316 (1944)
28. Savage, J.: Models of Computation: Exploring the Power of Computing. Addison-Wesley, Reading, MA. http://cs.brown.edu/~jes/book/pdfs/ModelsOfComputation.pdf (1998). (The URL given here goes to a free version of the book available online)
29. Smith, P.: An Introduction to Gödel's Theorems. Cambridge University Press, Cambridge, UK (2013). (This is the second edition of the book)
30. Turing, A.: Dissertation for the PhD: "Systems of Logic Based on Ordinals". Princeton University, Princeton, NJ (1938)
31. Turing, A.M.: On computable numbers with applications to the entscheidungsproblem. Proc. Lond. Math. Soc. **42**, 230–265 (1937)

http://www.springer.com/978-3-319-46375-9

Emergent Computation
A Festschrift for Selim G. Akl
Adamatzky, A. (Ed.)
2017, XIII, 643 p. 183 illus., 106 illus. in color.,
Hardcover
ISBN: 978-3-319-46375-9