Adam Olszewski, Jan Woleński, Robert Janusz. (Eds.)
Church's Thesis After 70 Years

ontos mathematical logic

edited by Wolfram Pohlers, Thomas Scanlon, Ernest Schimmerling
Ralf Schindler, Helmut Schwichtenberg

Volume 1

Adam Olszewski, Jan Woleński, Robert Janusz. (Eds.)

Church's Thesis After 70 Years

ontos
verlag

Frankfurt I Paris I Ebikon I Lancaster I New Brunswick

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie;
detailed bibliographic data is available in the Internet at http://dnb.ddb.de

North and South America by
Transaction Books
Rutgers University
Piscataway, NJ 08854-8042
trans@transactionpub.com

United Kingdom, Ire Iceland, Turkey, Malta, Portugal by
Gazelle Books Services Limited
White Cross Mills
Hightown
LANCASTER, LA1 4XS
sales@gazellebooks.co.uk

Livraison pour la France et la Belgique:
Librairie Philosophique J.Vrin
6, place de la Sorbonne ; F-75005 PARIS
Tel. +33 (0)1 43 54 03 47 ; Fax +33 (0)1 43 54 48 18
www.vrin.fr

P.O. Box 15 41, D-63133 Heusenstamm
www.ontosverlag.com

ISBN 3-938793-09-0
2006

Printed on acid-free paper
ISO-Norm 970-6
FSC-certified (Forest Stewardship Council)
This hardcover binding meets the International Library standard

Printed in Germany
by buch bücher **dd ag**

Contents

Preface 7

Darren Abramson
Church's Thesis and Philosophy of Mind 9

Andreas Blass, Yuri Gurevich
Algorithms: A Quest for Absolute Definitions . . 24

Douglas S. Bridges
Church's Thesis and Bishop's Constructivism . . . 58

Selmer Bringsjord, Konstantine Arkoudas
On the Provability, Veracity, and AI-Relevance of the Church–Turing Thesis 66

Carol E. Cleland
The Church–Turing Thesis. A Last Vestige of a Failed Mathematical Program 119

B. Jack Copeland
Turing's Thesis 147

Hartmut Fitz
Church's Thesis and Physical Computation 175

Janet Folina
Church's Thesis and the Variety of Mathematical Justifications 220

Andrew Hodges
Did Church and Turing Have a Thesis about Machines? 242

Leon Horsten
Formalizing Church's Thesis 253

Stanisław Krajewski
Remarks on Church's Thesis and Gödel's Theorem 269

Charles McCarty
THESIS AND VARIATIONS 281

Elliott Mendelson
ON THE IMPOSSIBILITY OF PROVING THE "HARD-HALF" OF CHURCH'S THESIS 304

Roman Murawski, Jan Woleński
THE STATUS OF CHURCH'S THESIS 310

Jerzy Mycka
ANALOG COMPUTATION AND CHURCH'S THESIS 331

Piergiorgio Odifreddi
KREISEL'S CHURCH 353

Adam Olszewski
CHURCH'S THESIS AS FORMULATED BY CHURCH — AN INTERPRETATION 383

Oron Shagrir
GÖDEL ON TURING ON COMPUTABILITY 393

Stewart Shapiro
COMPUTABILITY, PROOF, AND OPEN-TEXTURE 420

Wilfried Sieg
STEP BY RECURSIVE STEP: CHURCH'S ANALYSIS OF EFFECTIVE CALCULABILITY 456

Karl Svozil
PHYSICS AND METAPHYSICS LOOK AT COMPUTATION . . 491

David Turner
CHURCH'S THESIS AND FUNCTIONAL PROGRAMMING . . 518

Index 545

Preface

The 1930s became the golden decade of mathematical logic. The new generation of logicians, like Kurt Gödel, Alfred Tarski, Jacques Herbrand, Gerhardt Genzten, Arend Heyting, Alonzo Church and Alan Turing joined older masters, like Hilbert, Brouwer or Skolem. The received paradigm of logic, emphatically and optimistically expressed by Hilbert's famous phrase "Wir müssen wissen, und wir werden wissen", had to be replaced by a more limited one and forced by discoveries of incompleteness, undefinability and undecidability. In 1936 three men, Church, Post and Turing, almost simultaneously and mutually independently, proposed the identification of the intuitive concept of computability and the mathematical concept of recursiveness. Although this proposal had three fathers, it was baptized as Church's thesis (CT) and this label has been preserved even to this day. This thesis became one of the conceptual highlights of all time, at least in the area of logic and the foundations of mathematics.

In fact, it is a very rare case in history when a proposal which has all the features of a definition, becomes as dexterous as Church's thesis. On the one hand, almost everyone accepts it as very satisfying, but, on the other hand, it is continuously discussed by mathematicians, logicians, philosophers, computer scientists and psychologists. The discussion is many-sided and concerns the evidence for Church's thesis, its history, various formulations, possible objections, the role in mathematics or applications in philosophy. This collection of papers, mostly commissioned specifically for the present volume, is intended as a testimony of a great significance and a fascinating story of the statement that computability is equivalent to recursiveness. Our assumption was to focus on the following topics: Church's formulation of his thesis and its interpretations, different formulations of CT, CT and intuitionism, CT and intensional mathematics, CT and physics, epistemic status of CT, CT and philosophy of mind, (dis)provability of CT, CT and functional programming. Yet we have decided to publish these articles in the alphabetical order of

the authors' names. We hope that the result gives proper justice to this beautiful landmark in the history of conceptual analysis. However, we cannot abstain from reporting an anecdote. We began our attempts to start this project with an offer to one of the leading world publishers, hoping that the password 'Church's thesis' opens every door. Our query was forwarded to their religious (sic!) department, for which, in turn, this book was not interesting enough to publish. *Habent sua fata libelli.* Fortunately, Ontos Verlag had no problem with finding out the proper meaning of 'Church's thesis'.

Acknowledgments

We are also grateful to the following publishing houses for giving permission to reprint the listed papers:

1. A K Peters Ltd: for P. Odifreddi, "Kreisel's Church".
2. The Association for Symbolic Logic: for W. Sieg, "Step by Recursive Step: Church's Analysis of Effective Calculability".
3. Bulletin of the European Association for Theoretical Computer Science: for A. Blass, Y. Gurevich, "Algorithms: A Quest for Absolute Definitions".

The Editors

Darren Abramson*

Church's Thesis and Philosophy of Mind

1. Introduction

In this paper I examine implications of Church's Thesis, and related theses, for the philosophy of mind. I will argue that the present indeterminate status of theses concerning what functions physical objects compute have significant import for some arguments over what mental states are. I show, however, that other arguments are not affected by this status, and I argue against claims that we can prove that certain views in philosophy of mind fail by consideration of computability.

2. The Church–Turing Thesis and the Chinese Room

B. Jack Copeland has written extensively on the topic he calls 'hypercomputation' [Copeland 2002b], occasionally with reference to the philosophy of mind [Copeland 2000; 2002c]. Hypercomputation is referred to as 'computation beyond the Turing limit' [Copeland and Sylvan 1999]. A machine that hypercomputes, then, can produce values of a function that is not Turing computable. Copeland's thesis is that a crucial error has crept into the writings of numerous philosophers of mind that becomes clear upon consideration of hypercomputation. In his writings, Copeland claims to find a widespread philosophical error that concerns the Church–Turing Thesis, often called Church's Thesis. I will refer to it here as 'CTT'. Since Copeland takes such great care to distinguish 'CTT properly so-called' from other related theses, let us agree to understand by CTT the definition he gives.

*D. Abramson, Department of Philosophy, Dalhousie University, Halifax, Nova Scotia, Canada, B3H 4P9, da@dal.ca

CTT: "[...] no human computer, or machine that mimics a human computer, can out-compute a universal Turing machine" [Copeland 2002a, p. 67].

Copeland's complaint can be easily summarized. The locutions 'computable by a physical device' and 'computable by a machine following an effective procedure' mean different things, and may describe different sets of functions. Copeland thus distinguishes between CTT and another thesis, which we may call 'PCTT' for 'physical Church–Turing Thesis'.

PCTT: The *physically* computable functions are a subset of the Turing computable functions.

The so-called 'Church–Turing Fallacy' is the conflation of PCTT with CTT. In some of his papers, Copeland cites examples in which, he claims, well known philosophers of mind and cognitive scientists argue as follows. Some proposition P follows from CTT; everyone thinks CTT is true; therefore P. However, Copeland says, the Church–Turing Fallacy is committed insofar as P only follows from PCTT and not CTT.

To better understand what is at stake, let us examine a supposed instance of the fallacy. Copeland claims that Searle's famous Chinese Room Argument founders on the Church–Turing Fallacy [Copeland 2002c].

Recall the form that Searle's Chinese Room Argument takes. Its conclusion is that there is no computer program such that, merely by implementing it, can a machine possess understanding.[1] Searle imagines himself, a philosopher who speaks no Chinese, and an implementation of a program which supposedly can take as inputs Chinese stories and then understand them, interacting in an enclosed room. Searle imagines himself carrying out the program's instructions inside the enclosed room and observes that neither he, nor the room, understands Chinese, regardless of what people outside the room may think.

[1]Note that this thesis is consistent with computational functionalism, which states merely that implementing some programs is sufficient for having mental states. Whether mental states count as understanding may depend on facts other than computational ones.

We will now look at aspects of Searle's argument relevant to CTT a little closer. The so-called 'many-mansions reply' consists of the claim that Searle has merely shown that *existing* computers do not understand, and there may come a point in the future in which other, human-constructed machines, do in fact think, perhaps due to novel causal powers they possess. Here is Searle's response to the many-mansions reply.

> The interest of the original claim made on behalf of artificial intelligence is that it was a precise, well defined thesis: mental processes are computational processes over formally defined elements. I have been concerned to challenge that thesis. If the claim is redefined so that it is no longer that thesis, my objections no longer apply because there is no longer a testable hypothesis for them to apply to. [Searle 1980, p. 197][2]

Of considerable relevance to our discussion is the ambiguity in the word 'computational' in Searle's original article. A charitable interpretation, given Copeland's concern over misinterpreting CTT, is 'effectively computable'. However, Copeland points out that Searle's later writings ask us to read the Chinese Room Argument as concerning *any* form of computation, where computation *simpliciter* is "the manipulation of formal symbols" [Copeland 2002c, p. 120].

Let us generalize the argument to take into account arbitrary manipulations of symbols by supposing that instead of a program, we are considering any rule-governed relationship between inputs and outputs that may be implemented in physical matter. Then, successfully constructing the thought experiment requires the assumption that John Searle could effectively follow the functional specification associated with understanding Chinese.

If Searle is limiting his view to effectively computable functions, then his assumption, that he could implement an arbitrary implementation of such a function, would be justified. After all, 'effectively computable' simply describes those functions that a person, with enough time, paper, and pencil, can compute. However, as we have seen, he takes 'functional specification' to be more general than 'effectively computable'. He takes it to mean 'any symbol manipulation that may occur'. So, to conclude that the symbol manipulations which functionalism claims to underlie cognition can be effectively

[2]Pagination follows the reprint of Searle's article in [Haugeland 1997].

computed, Searle needs PCTT. We might think that some intermediate thesis is required, for example that all of the 'humanly computable' functions are Turing computable, where a humanly computable function is one that a person can compute, effectively or otherwise. However, this thesis is too weak, since the functions that underlie cognition may not be computable by humans.[3]

So, Copeland says, Searle commits the Church–Turing Fallacy. If PCTT is false, which it very well may be, then some physically computable functions might not be Turing computable. Then, from CTT, it follows that some physically computable functions are not effectively computable, by John Searle or anyone else. So, the Chinese Room Argument proves nothing about whether cognition really is formal symbol manipulation.

This is, in essence, Copeland's argument. A first response that one might make is that Searle really is, despite lack of care in some places, concerned with what Turing-equivalent computers can do. After all, in the article that introduced the Chinese Room Argument, Searle [1980] is explicitly interested in evaluating the claims of contemporary practitioners of 'Strong AI', those who believe that the programs they are writing to be run on mainframe and desktop computers do really understand stories, and can reason. For the time being, though, let us agree with Copeland's exigesis: Searle intends his argument against computational functionalism in general. After all, irrespective of Searlean hermeneutics, it is of interest to see if some kind of computational functionalism can survive the Chinese Room Argument.

Notice that Copeland's analysis leads in the right direction, but ignores a key property of Searle's argument: its thoroughgoing modal character. Copeland says that Searle needs PCTT to be able to pick an arbitrary function, and then imagine himself computing it in the Chinese room. However, he needs the following stronger thesis: $\Box$PCTT. Thought experiments work by showing us what is possible. If we claim a necessary identity and are then shown that this breaks down in a possible world, we must revise our identity claim. However, we just don't know if PCTT is true. We construct a thought experiment to investigate what it's like to compute whatever functions underlie cognition. To do so, we must assume that PCTT holds

[3]We can easily imagine an epistemic limitation on our ability to ascertain what functions biological tissue computes, for example.

in the arbitrary possible world we are considering. So, the Searlean strategy requires □PCTT, not merely PCTT or ◇PCTT.

However, the bare PCTT is already a modal claim. To say that all physically computable functions are Turing computable, presumably, doesn't trivially fail in a finite universe. In other words, we take 'physically computable' to be some sort of augmentation of 'effectively computable'. In investigating the class of effectively computable functions we, as Turing did, see what you can do with access to unbounded quantities of time, pencil, and paper. When we investigate PCTT, presumably, we ought still to consider unbounded access to resources. We may add, therefore, to any physical computation an unbounded number of steps which involve use of physical devices and/or substances, say, chunks of radium and geiger counters. Now, suppose that all the radium and geiger counters in the universe are made inaccessible, by being thrown into a black hole. In investigating PCTT we would still want to know what we could compute with access to objects consistent with the laws of physics. So, we would ignore the contingent unavailability of radium and geiger counters and reason over their use in deciding if PCTT were true. Notice that widespread agreement that all people are mortal does not prevent us from thinking that people can add—and the adding function has an infinite domain. If it turns out that effective methods do not capture all the methods at work in physical computation, then we must treat physical methods in the same unbounded fashion with which we treat effective methods.

If we take PCTT to be 'intrinsically modal' in the sense described above, and subscribe to popular views of rigid designation and scientific law, PCTT implies □PCTT. For, suppose it is the case that all the functions which are physically computable are Turing computable. By an externalist theory of reference, we take 'the laws of physics' to determine a set of possible worlds such that anything that is consistent with the laws of physics in our world is consistent with the laws of physics in those worlds. If it turns out that nothing can travel faster than the speed of light in this world, then in no physically possible world can anything travel faster than the speed of light. The same goes for physical computability.

Of course, if we construe PCTT non-modally, or at least with only the same modal relaxations as CTT, then we get different results. Then, Searle really does need □PCTT. For, suppose that there

is some physical property P which human brains need unbounded access to in order to compute the functions underlying, say, Chinese understanding. Suppose also that, in the limit, measuring P results in an uncomputable series of values. Then assuming that PCTT is true won't permit us to conclude that in other possible worlds what is physically computable is Turing- and effectively computable. For, the actual world might have finite physical resources, while some possible worlds might have infinite physical resources. With deference to Copeland, I find it most convenient to simply read PCTT in the thoroughgoing modal nature described above. As we will see very shortly, the matters of rigid designation and possible worlds are of central importance to understanding the impact of hypercomputation on the philosophy of mind. After looking at another argument involving conceivable worlds and functionalism, I will be in a position to offer a new, relaxed thesis—a generalized computational functionalism.

3. A Second Modal Argument

In this section I examine a more recent argument that attempts to argue that, on the basis of the purported *failure* of a version of CTT, that computational functionalism is false. In their recent book, Bringsjord and Zenzen [2003] make this claim. However, since their goal is to convince their reader that at least part of what constitutes our mental life is hypercomputation, they are sensitive to issues of computability. Following Searle's [1992] lead, they argue that, given certain a priori considerations, these three propositions are not cotenable.

1. Persons are material things, viz., their brains (or some proper part of their nervous systems).

2. There is a conceptual or logical connection between P-consciousness, i.e. phenomenal consciousness, as in [Block 1995], and the structure of, and information flow in, brains. Necessarily, if a person a enjoys a stretch of P-consciousness from t_i to t_k, this consciousness is identical to some computation c from t_i to t_k of some Turing machine or other equivalent computational system m, instantiated by a's brain.

3. A person's enjoying P-consciousness is conceptually or logically distinct from that person's brain being instantiated by a Turing machine running through some computation.

With Searle [*ibid.*], they note that the following scenario both seems plausible, and demonstrates that these three propositions cannot be simultaneously held. A neuroscientist incrementally replaces portions of my brain with little silicon-based computers designed to mimic the information flow of brain matter perfectly. As bits of brain matter are replaced, conscious experience disappears bit by bit, but my outward behavior remains the same.

Bringsjord and Zenzen argue that, if this scenario seems *possible* then we must reject the second proposition above, and thereby reject computationalism. Here is their argument in a nutshell. The second proposition tells us that in all possible worlds, having a brain with the right Turing-computable information flow implies the possession of particular conscious experiences. However, this is incompatible with the third proposition above, and the plausible claim that conceivability is a guide to possibility. The possible scenario described shows the conceptual distinction between P-consciousness and instantiating a Turing machine running through some computation.

What happens, however, when we remove the second clause of the second proposition above? Suppose we simply hold that 'there is a conceptual or logical connection between P-consciousness and the structure of, and information flow in, brains'. Given the truncated version of the second proposition, it seems at first that we can employ precisely the same scenario as before to reject the claim of necessary identity between objects instantiating the right information flow, and objects having P-consciousness. However, Copeland's lesson against the Chinese Room Argument can be recast.

For, suppose that the information flow instantiated in our brains is not recursive. Consider an arbitrary set S that is not recursively enumerable. Now, suppose that if computational neuroscientists could discover the truth of the matter, they would find that a fundamental property of the brain is to receive summed inputs from afferent neurons, convert that input to a real number r, and fire to an efferent neuron if and only if the initial, non-repeating decimal value of r is in S.

First, this is an exemplary instance of 'structured information flow'. Second, despite commitments we may have against PCTT,

this is at least as plausible as any zombie scenario. Finally, notice that we cannot even construct the zombie scenario once we admit that this outlandish, mathematical characterization of brain activity *might* be true. For, all the computers we know how to build are implementations of Turing-complete architectures. In other words, any set of natural numbers decidable by our bits of silicon are decidable by a Turing machine. So, we can deny that the scenario involving the surgeon and his replacement parts pick out any possible world. Once we observe that recursion theory shows us the conceptual possibility of physical structures that implement non-Turing computable functions, both zombies and the Chinese Room fail alike as counterexamples to a generalized form of computational functionalism.

4. Thought Experiments and Philosophy of Mind

Careful attention to CTT and related theses does not solve all of the problems we might have in holding computational functionalism. To show this, I will invoke Kripke's arguments concerning rigid designation of natural kind terms. According to Lycan, Kripke's comments on identity, necessity, and materialism can be easily modified to concern not the identity theorists he has in mind in his writings, but functionalists also [Lycan 1974]. Lycan considers the now quaint machine-state functionalism. However, let us paraphrase, with slight alterations, his application of rigid designators to machine-state functionalism, allowing that having mental states may be identical to computing of a function that no Turing machine can compute. 'According to us, my pain is identical with my functional state (of type) S_p. "My functional state S_p" is just *whatever* state of me can be construed as obeying the symbolically sensitive method that defines S_p' [see *ibid.*, p. 688 for comparison]. Given the failure of machine-state functionalism, we must offer something other than 'state of a Turing machine' for defining S_p.

In a moment I will show why I think we cannot answer Lycan's version of Kripke's challenge by appeal to the failure of CTT. First, however, we will examine more closely what the above statement of computational functionalism amounts to. I concede that Block's argument against the validity of any finitely bounded Turing Test shows that we are committed to counting as important for possessing mental abilities "the character of the internal information processing that produces [them]" [Block 1981, p. 5]. However, I

am reluctant to use the phrase 'algorithm', or especially, 'effective method' in defining S_p since we are discussing the possibility that there might be other forms of information processing. So, by consideration of the fact that physical computability might outrun effective computability, we have a proposal for a modified version of computational functionalism. S_p includes *any* method, effective, physical, or otherwise that biological systems have available for converting inputs to ouputs. I call this a 'relaxed' version of computational functionalism because it takes its general form while permitting more computational methods. While this view seems to deal with certain arguments against traditional computational functionalism, we will now see that it is not immune to others.

Recall Kripke's famous challenge. We can conceive of worlds in which we have pain but instantiate none of the correlates the materialist tell us are identical to pain. Lycan shows us that the same is true for functionalism. Since we fix the reference of pain directly, we cannot claim that we are mistaking conceivable worlds in which there merely *appears* to be pain for ones in which there actually *is* pain, at least not in the usual way.

When we are told why it is that we can conceive of worlds in which there appears to be, say, heat but not mean molecular kinetic energy, we accept that we fix the reference of heat by something contingently related to the heat itself—namely, by our experiences of heat. We can't use this strategy in the case of pain since we *do* fix its reference directly. Moreover, there is no difficulty in imagining possible worlds in which we have mental states, but there are only disembodied souls and no machines computing any functions at all. The move that countered the zombie and Chinese Room cases does not work here because we begin by imagining mental phenomena, which we can do easily, and the thought experiment does not rely on our ability to imagine some other phenomena.

I take it that there are live, interesting projects for showing why we are mistaken in taking these apparently conceivable worlds to be possible. I can't do the topic justice here, but for two examples of an attempt to provide a non-Kripkean (i.e., disanalogous to the heat case) solution to the problem of apparently conceivable worlds as just described see George Bealer's paper 'Mental Properties' [1994] and the recent book *Names, Necessity, and Identity* by Christopher Hughes [2004]. Because our imagination limits itself here to the

possible existence of disembodied souls (say), we cannot block the crucial step against a theory of mind by invoking the Church–Turing fallacy.

However, we should be optimistic that progress has been made. There is intuitive appeal to the idea that the possible worlds which afford counterexamples to different conditionals must be dealt with in different ways. The first conditional at work in the Chinese Room Argument is that, in any possible world, if I have certain functional properties then I have certain mental ones also. The conditional being examined now is that, in any possible world, if I have mental properties, I have certain computational ones also. Suppose we start with a theory according to which mental states, events, and processes, are identical to computational ones. Then, when we conceive of worlds with the computational phenomena but not the mental ones, there should be something about the computational phenomena we have failed to notice. Similarly, when we we are dazzled by our apparent ability to imagine the mental phenomena but not the functional ones, it should be the imagined properties of the mental phenomena that reveal our error. So, I claim that the failure to address Kripke's problem for computational functionalism is to be expected. Also, it is no small matter to deal directly with threatening thought experiments in which there are all the computational phenomena but none of the mental ones.

Suppose we think with Yablo that "[almost] everything in *The Conscious Mind* [Chalmers 1997] turns on the single claim. The claim is that there can be *zombie worlds*: worlds physically like our own but devoid of consciousness." [Yablo 1999, p. 455]. Then we have ammunition against some of the most prominent, recent threats to materialism. Note that the argument presented here against the conceivability of zombie worlds can be applied to other cases including dancing and inverted qualia. In responding to Yablo, Chalmers tells us "[most commentators on his book] bite the bullet and argue that psychophysical necessities are different in kind from the Kripkean examples, and not explicable by the two-dimensional [semantic] framework." [Chalmers 1999, p. 477]. With the line of argument being presented, we can avoid bullet biting and take on necessary psychophysical (token) identification directly.

5. Do We Hypercompute?

So far, I have claimed that it is possible that people hypercompute, and that this possibility is enough to defend a relaxed form of computationalism against some familiar thought experiments. In this section, I will insist on taking the middle ground: I don't think we can *prove* that we hypercompute. In a recent paper Bringsjord and Arkoudas have presented an argument which they think *proves* that people do hypercompute [Bringsjord and Arkoudas 2004]. In essence, they apply Kripke's technique for arguing against materialists in order to show that we hypercompute. Let $Dxyz$ mean 'Turing machine x determines whether Turing machine y halts on input z. Let m range over Turing machines. Then, the unsolvability of the halting problem tells us that there is a particular Turing machine m_0 such that

Proposition 1 $\forall m \exists i \neg \Diamond D m m_0 i$

Let p range over people. Then, we are told, computationalism can be construed as

Proposition 2 $\forall p \exists m \; p = m$

1 and 2 together let us derive

Proposition 3 $\forall p \exists i \neg \Diamond D \; p m_0 i$

In plain language, 3 says that no matter who you are, there is a Turing machine out there—the one the proof that shows the halting problem is unsolvable says exists—and some input, such that we cannot determine whether that machine halts on that input.

To complete their reductio against computationalism, Bringsjord and Arkoudas introduce the following premise which is inconsistent with 3:

Proposition 4 $\forall p \forall i \Diamond D p m_0 i$

3 and 4 together yield a contradiction. Since we are not prepared to reject Turing's proof of the unsolvability of the halting problem, we must instead reject 2. Since people compute functions, and are identical to no Turing machine, Bringsjord and Arkoudas conclude

that people hypercompute. 4 says that for any person and any input, it possible for that person to decide whether Turing machine m_0 halts on that input. They do, of course, recognize that 4 is tendentious, and provide some arguments for it.

Two main lines of argument can be isolated. First, they claim, mathematical practice by children, let alone experts, reveals the *method* by which we hypercompute. A favorite example, given in [Bringsjord and Zenzen 2003] as well as the paper under discussion, is the familiar pictorial argument for $\lim_{n\to\infty} 1/2^n = 0$. According to the authors, we can complete an infinite number of steps in a finite amount of time to see that the equality holds—by completing such supertasks, human beings can hypercompute. However, understanding the precise relationship between the phenomenal nature of doing mathematics and the computational resources underlying this ability is difficult to say the least. Nor does the mere fact that we use calculus or picture proofs imply that we ever actually complete an infinite number of mathematical steps in a proof.

The second argument for 4 runs as follows. For over 50 years, mathematicians, including Turing and Gödel, have investigated the properties of notional machines that solve the halting problem. Many results concerning the structure of the arithmetical hierarchy make extensive use of such machines: see the classic discussion in [Rogers 1967, p. 129] for an introduction. So, it seems to involve no contradiction to suppose that our brain mechanisms make regular queries of the halting set just as oracle machines do.

Could this possibility be an illusion? It is far from obvious that hypercomputation is an intrinsic property of our mental life in the same way that pain sensations are constitutive of being in pain. As a matter of fact, either we hypercompute or we don't. If we do, then it is possible that we do. But if we do not hypercompute, then despite the imaginability of our brains behaving as though they are Turing machines with oracles attached, we may one day discover that we do not hypercompute.

We are in precisely the same position with respect to claims that 'persons possibly hypercompute' as 15th century alchemists were with respect to the claim that 'heat can exist in the absence of mean molecular kinetic energy'. Bringsjord and Arkoudas paraphrase Chalmers [1997] and say that "when some state of affairs ψ seems, by all accounts, to be perfectly coherent, the burden of proof

is on those who would resist the claim that ψ is logically or mathematically possible" [Bringsjord and Arkoudas, p. 183]. I have hinted at how this burden must be borne, and now will do so explicitly.

For any Turing degree T, one can coherently hold the view that mentation is the same as instantiating information flow which is not above T in the Kleene hierarchy. Now, if it is constitutive of having a human mental life to instantiate functions of a particular Turing degree, then anyone who thinks that they can imagine a person computing functions above that degree have deluded themselves. It is true that we don't know which Turing degree, or less, is the correct one for fixing our mental properties. For that very reason, we may deny that some scenarios, such as those in which we solve the halting problem, are possible. We may grant that such scenarios hold epistemic possibility. We do not have recourse, as Kripke does, to the fact that we fix the reference of mental states directly by phenomenal feel.

In fact, first-order axiomatizations of mathematics suggest that we have good reason for limiting the epistemic possibility of hypercomputation. That is, if mathematicians think that, despite the tedium involved, all mathematical results are really consequences of the Peano, or Zermelo–Fraenkel Axioms, then I have positive reasons to deny the possibility claimed by Bringsjord and Arkoudas. A significant argument, which is not yet on offer, is that mathematics is what mathematics feels like. In short, we may question the metaphysical possibility of hypercomputation—and reject 4—on the basis of necessary identity and an epistemic position provided by the philosophy of mathematics.

6. Conclusion

I have shown three things. Assume that we don't know whether it is constitutive of us to hypercompute. Then zombie arguments fail, because we are unable to construct the relevant thought experiments—the same goes for the Chinese Room. Second, we cannot conclude that we do hypercompute from the mere appearance that it seems logically possible. Finally, those who seek the philosophical benefits of computationalism may find them in its relaxed form, in which 'information flow', Turing computable or not, underlies our mental lives. Each of these conclusions has been reached by consideration of CTT and related theses.

It might be argued that these conclusions are merely of conditional interest. If we have good reason to think that hypercomputation is not possible, and that PCTT or a modal counterpart holds, then the conclusions I have argued for would be in vain. Deflationary arguments such as offered in [Kreisel 1982] and [Davis 2005] rely on epistemological arguments against the metaphysical possibility of hypercomputation, and so are less than convincing. I do not have the resources here to offer a positive argument for a model of hypercomputation; however, I leave the reader to the other papers in this collection that do.

References

Bealer, G. [1994], "Mental Properties", *Journal of Philosophy* **91**(4), 185–208.

Block, N. [1981], "Psychologism and Behaviorism", *The Philosophical Review* **90**(1), 5–43.

Block, N. [1995], "On a Confusion about a Function of Consciousness", *Behavioral and Brain Sciences*, 18.

Bringsjord, S. and Arkoudas, K. [2004], "The Modal Argument for Hypercomputing Minds", *Theoretical Computer Science* **317**, 167–190.

Bringsjord, S. and Zenzen, M. [2003], *Superminds: People Harness Hypercomputation and More*, Kluwer.

Chalmers, D. [1997], *The Conscious Mind*, Oxford University Press.

Chalmers, D. [1999], "Materialism and the Metaphysics of Modality", *Philosophy and Phenomenological Research* **59**(2), 473–493.

Copeland, B.J. [2000], "Narrow versus Wide Mechanism: Including a Reexamination of Turing's Views on the Mind-Machine Issue", *The Journal of Philosophy* **96**, 5–32.

Copeland, B.J. [2002a], *Computationalism: New Directions*, chapter: *Narrow versus Wide Mechanism*, MIT Press, pp. 59–86.

Copeland, B.J. [2002b], "Hypercomputation", *Minds and Machines* **12**, 461–502.

Copeland, B.J. [2002c], *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence*, chapter: *The Chinese*

Room from a Logical Point of View, Oxford University Press, pp. 109–122.

Copeland, J. and Sylvan, R. [1999], "Beyond the Universal Turing Limit", *Australasian Journal of Philosophy* **77**(46–66).

Davis, M. [2005], "The Myth of Hypercomputation", in *Alan Turing: Life and Legacy of a Great Thinker*, (C. Teuscher ed.), Springer-Verlag, pp. 195–210.

Haugeland, J. [1997], *Mind Design II*, MIT Press.

Hughes, C. [2004], *Kripke: Names, Necessity, and Identity*, Oxford University Press.

Kreisel, G. [1982], Review: "A Computable Ordinary Differential Equation Which Possesses No Computable Solution; The Wave Equation With Computable Initial Data Such That Its Unique Solution Is Not Computable", *The Journal of Symbolic Logic* **47**(4), 900–902.

Lycan, W.G. [1974], "Kripke and the Materialists", *Journal of Philosophy* **71**(18), 677–689.

Rogers, H. [1967], *Theory of Recursive Functions and Effective Computability*, MIT Press.

Searle, J. [1980], "Minds, Brains, and Programs", *Behavioral and Brain Sciences*, 3.

Searle, J. [1992], *The Rediscovery of the Mind*, MIT Press.

Yablo, S. [1999], "Concepts & Consciousness", *Philosophy and Phenomenological Research* **59**(2), 455–463.

Andreas Blass, Yuri Gurevich*

Algorithms: A Quest for Absolute Definitions

What is an algorithm? The interest in this foundational problem is not only theoretical; applications include specification, validation and verification of software and hardware systems. We describe the quest to understand and define the notion of algorithm. We start with the Church–Turing thesis and contrast Church's and Turing's approaches, and we finish with some recent investigations.

1. Introduction

In 1936, Alonzo Church published a bold conjecture that only recursive functions are computable [Church 1936]. A few months later, independently of Church, Alan Turing published a powerful speculative proof of a similar conjecture: every computable real number is computable by the Turing machine [Turing 1937]. Kurt Gödel found Church's thesis "thoroughly unsatisfactory" but later was convinced by Turing's argument. Later yet he worried about a possible flaw in Turing's argument. In Section 2 we recount briefly this fascinating story, provide references where the reader can find additional details, and give remarks of our own.

By now, there is overwhelming experimental evidence in favor of the Church–Turing thesis. Furthermore, it is often assumed that the Church–Turing thesis settled the problem of what an algorithm is. That isn't so. The thesis clarifies the notion of computable function. And there is more, much more to an algorithm than the function it

*A. Blass, Mathematics Department, University of Michigan, Ann Arbor, MI 48109; Y. Gurevich, Microsoft Research, One Microsoft Way, Redmond, WA 98052.

computes. The thesis was a great step toward understanding algorithms, but it did not solve the problem what an algorithm is.

Further progress in foundations of algorithms was achieved by Kolmogorov and his student Uspensky in the 1950s [Kolmogorov 1953; Kolmogorov and Uspensky 1958]. The Kolmogorov machine with its reconfigurable "tape" has a certain advantage over the Turing machine. The notion of pointer machine was an improvement of the notion of Kolmogorov machine. These issues are discussed in Section 3.

This paper started as a write-up of the talk that the second author gave at the Kolmogorov Centennial conference in June 2003 in Moscow. The talk raised several related issues: physics and computation, polynomial time versions of the Turing thesis, recursion and algorithms. These issues are very briefly discussed in Section 4.

In 1991, the second author published the definition of sequential abstract state machines (ASMs, called evolving algebras at the time) [Gurevich 1991]. In 2000, he published a definition of sequential algorithms derived from first principles [Gurevich 2000]. In the same paper he proved that every sequential algorithm A is behaviorally equivalent to some sequential ASM B. In particular, B simulates A step for step. In Section 5 we outline the approach of [Gurevich 2000].

In 1995, the second author published the definition of parallel and distributed abstract state machines [Gurevich 1995]. The Foundations of Software Engineering group at Microsoft Research developed an industrial strength specification language AsmL that allows one to write and execute parallel and distributed abstract state machines [AsmL]. In 2001, the present authors published a definition of parallel algorithms derived from first principles as well as a proof that every parallel algorithm is equivalent to a parallel ASM [Blass and Gurevich 2003]. Section 6 is a quick discussion of parallel algorithms.

The problem of defining distributed algorithms from first principles is open. In Section 7 we discuss a few related issues.

Finally let us note that foundational studies go beyond satisfying our curiosity. Turing machines with their honest counting of steps enabled computational complexity theory. Kolmogorov machines and pointer machines enabled better complexity measures. Abstract state machines enable precise executable specifications of

software systems though this story is only starting to unfold [ASM, AsmL, Börger and Stärk 2003].

Added in proof. This paper was written in 2003. Since then the ASM characterization has been extended to small-step interactive algorithms. Work continues on other aspects [Gurevich 2005].

2. The Church–Turing Thesis

2.1. Church + Turing

The celebrated Church–Turing thesis [Church 1936, Turing 1937] captured the notion of computable function. Every computable function from natural numbers to natural numbers is recursive and computable, in principle, by the Turing machine. The thesis has been richly confirmed in practice. Speaking in 1946 at the Princeton Bicentennial Conference, Gödel said this [Gödel 1990 (article 1946)]:

> Tarski has stressed in his lecture (and I think justly) the great importance of the concept of general recursiveness (or Turing's computability). It seems to me that this importance is largely due to the fact that with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e., one not depending on the formalism chosen. In all other cases treated previously, such as demonstrability or definability, one has been able to define them only relative to the given language, and for each individual language it is clear that the one thus obtained is not the one looked for. For the concept of computability, however, although it is merely a special kind of demonstrability or decidability, the situation is different. By a kind of miracle it is not necessary to distinguish orders, and the diagonal procedure does not lead outside the defined notion.

2.2. Turing − Church

It became common to speak about the Church–Turing thesis. In fact the contributions of Church and Turing are different, and the difference between them is of importance to us here. Church's thesis was a bold hypothesis about the set of computable functions. Turing analyzed what can happen during a computation and thus arrived at his thesis.

Church's Thesis. The notion of an effectively calculable function from natural numbers to natural numbers should be identified with that of a recursive function.

Church [1936] had in mind total functions. Later Kleene [1938] improved on Church's thesis by extending it to partial functions. The fascinating history of the thesis is recounted in [Davis 1982]; see also [Sieg 1997].

Originally Church hypothesized that every effectively calculable function from natural numbers to natural numbers is definable in his lambda calculus. Gödel didn't buy that. In 1935, Church wrote to Kleene about his conversation with Gödel [Davis 1982, p. 9].

> In discussion [*sic*] with him the notion of lambda-definability, it developed that there was no good definition of effective calculability. My proposal that lambda-definability be taken as a definition of it he regarded as thoroughly unsatisfactory. I replied that if he would propose any definition of effective calculability which seemed even partially satisfactory I would undertake to prove that it was included in lambda-definability. His only idea at the time was that it might be possible, in terms of effective calculability as an undefined notion, to state a set of axioms which would embody the generally accepted properties of this notion, and to do something on this basis.

Church continued:

> Evidently it occurred to him later that Herbrand's definition of recursiveness, which has no regard to effective calculability, could be modified in the direction of effective calculability, and he made this proposal in his lectures. At that time he did specifically raise the question of the connection between recursiveness in this new sense and effective calculability, but said he did not think that the two ideas could be satisfactorily identified "except heuristically".

The lectures of Gödel mentioned by Church were given at the Institute for Advanced Study in Princeton from February through May 1934. In a February 15, 1965, letter to Martin Davis, Gödel wrote [Davis 1982, p. 8]:

> However, I was, at the time of these lectures [1934], not at all convinced that my concept of recursion comprises all possible recursions.

Soon after Gödel's lectures, Church and Kleene proved that the Herbrand–Gödel notion of general recursivity is equivalent to lambda definability (as far as total functions are concerned), and Church became sufficiently convinced of the correctness of his thesis to publish it. But Gödel remained unconvinced.

Indeed, why should one believe that lambda definability captures the notion of computability? The fact that lambda definability is equivalent to general recursivity, and to various other formalizations of computability that quickly followed Church's paper, proves only that Church's notion of lambda definability is very robust.

To see that a mathematical definition captures the notion of computability, one needs an analysis of the latter. This is what Turing provided to justify his thesis.

Turing's Thesis. Let Σ be a finite alphabet. A partial function from strings over Σ to strings over Σ is effectively calculable if and only if it is computable by a Turing machine.

Remark. Turing designed his machine to compute real numbers but the version of the Turing machine that became popular works with strings in a fixed alphabet. Hence our formulation of Turing's thesis.

Turing analyzed a computation performed by a human computer. He made a number of simplifying without-loss-of-generality assumptions. Here are some of them. The computer writes on graph paper; furthermore, the usual graph paper can be replaced with a tape divided into squares. The computer uses only a finite number of symbols, a single symbol in a square. "The behavior of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment". There is a bound on the number of symbols observed at any one moment. "We will also suppose that the number of states of mind which need to be taken into account is finite [...] If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused". He ends up with a Turing machine simulating the original computation. Essentially Turing derived his thesis from more or less obvious first principles though he didn't state those first principles carefully.

"It seems that only after Turing's formulation appeared," writes Kleene in [1981, p. 61], "did Gödel accept Church's thesis, which had

then become the Church–Turing thesis." "Turing's arguments," he adds in [Kleene 1988, p. 48], "eventually persuaded him."

Church's lambda calculus was destined to play an important role in programming theory. The mathematically elegant Herbrand–Gödel–Kleene notion of partial recursive functions served as a springboard for many developments in recursion theory. The Turing machine gave us honest step counting and became eventually the foundation of complexity theory.

2.3. Remarks on Turing's Analysis

Very quickly the Church–Turing thesis acquired the status of a widely shared belief. Meantime Gödel grew skeptical of at least one aspect of Turing's analysis. In a remark published after his death, Gödel writes this [Gödel 1990 (article 1972a, p. 306)].

> *A philosophical error in Turing's work.* Turing in his [Turing 1937, p. 250], gives an argument which is supposed to show that mental procedures cannot go beyond mechanical procedures. However, this argument is inconclusive. What Turing disregards completely is the fact that *mind, in its use, is not static, but constantly developing*, i.e. that we understand abstract terms more and more precisely as we go on using them, and that more and more abstract terms enter the sphere of our understanding. There may exist systematic methods of actualizing this development, which could form part of the procedure. Therefore, although at each stage the number and precision of the abstract terms at our disposal may be *finite*, both (and therefore, also Turing's number of *distinguishable states of mind*) may *converge toward infinity* in the course of the application of the procedure.

Gödel was extremely careful in his published work. It is not clear whether the remark in question was intended for publication as is. In any case, the question whether mental procedures can go beyond mechanical procedures is beyond the scope of this paper, which focuses on algorithms. Furthermore, as far as we can see, Turing did not intend to show that mental procedures cannot go beyond mechanical procedures. The expression "state of mind" was just a useful metaphor that could be and in fact was eliminated: "we

avoid introducing the 'state of mind' by considering a more physical and definite counterpart of it" [Turing 1937, p. 253].

But let us consider the possibility that Gödel didn't speak about biology either, that he continued to use Turing's metaphor and worried that Turing's analysis does not apply to some algorithms. Can an algorithm learn from its own experience, become more sophisticated and thus compute a real number that is not computable by the Turing machine? Note that the learning process in question is highly unusual because it involves no interaction with the environment. (On the other hand, it is hard to stop brains from interacting with the environment.) Gödel gives two examples "illustrating the situation", both aimed at logicians.

> Note that something like this indeed seems to happen in the process of forming stronger and stronger axioms of infinity in set theory. This process, however, today is far from being sufficiently understood to form a well-defined procedure. It must be admitted that the construction of a well-defined procedure which could actually be carried out (and would yield a non-recursive number-theoretic function) would require a substantial advance in our understanding of the basic concepts of mathematics. Another example illustrating the situation is the process of systematically constructing, by their distinguished sequences $\alpha_n \to \alpha$, all recursive ordinals α of the second number-class.

The logic community has not been swayed. "I think it is pie in the sky!" wrote Kleene [1988, p. 51]. Here is a more expansive reaction of his [Kleene 1988, p. 50].

> But, as I have said, our idea of an algorithm has been such that, in over two thousand years of examples, it has separated cases when mathematicians have agreed that a given procedure constitutes an algorithm from cases in which it does not. Thus algorithms have been procedures that mathematicians can describe completely to one another *in advance* of their application for various choices of the arguments. How could someone describe completely to me *in a finite interview* a process for finding the values of a number-theoretic function, the execution of which process for various arguments would be keyed to more than the *finite* subset of our mental states that

> would have developed by the end of the interview, though the total number of mental states might converge to infinity if we were immortal? Thus Gödel's remarks do not shake my belief in the Church–Turing thesis [...]

If Gödel's remarks are intended to attack the Church–Turing thesis, then the attack is a long shot indeed. On the other hand, we disagree with Kleene that the notion of algorithm is that well understood. In fact the notion of algorithm is richer these days than it was in Turing's days. And there are algorithms, of modern and classical varieties, not covered directly by Turing's analysis, for example, algorithms that interact with their environments, algorithms whose inputs are abstract structures, and geometric or, more generally, non-discrete algorithms. We look briefly at the three examples just mentioned.

Interactive algorithms. This is a broad class. It includes randomized algorithms; you need the environment to provide random bits. It includes asynchronous algorithms; the environment influences action timing. It includes nondeterministic algorithms as well [Gurevich 2000 (sec. 9.1)]. Clearly, interactive algorithms are not covered by Turing's analysis. And indeed an interactive algorithm can compute a non-recursive function. (The nondeterministic Turing machines, defined in computation theory courses, are known to compute only partial recursive functions. But a particular computation of such a machine cannot in general be simulated by a deterministic Turing machine.)

Computing with abstract structures. Consider the following algorithm P that, given a finite connected graph $G = (V, E)$ with a distinguished vertex s, computes the maximum distance of any vertex from s.

A $S := \{s\}$ and $r := 0$.

B If $S = V$ then halt and output r.

C If $S \neq V$ then $S := S \cup \{y\colon \exists x\,(x \in S \wedge E(x, y))\}$ and $r := r+1$.

D Go to **B**.

P is a parallel algorithm. Following Turing's analysis we have to break the assignment $S := \{y\colon \exists x\,(x \in S \wedge E(x, y))\}$ into small tasks

of bounded complexity, e.g. by going systematically though every $x \in S$ and every neighbor y of x. But how will the algorithm go through all $x \in S$? The graph G is not ordered. A nondeterministic algorithm can pick an arbitrary vertex and declare it the first vertex, pick one of the remaining vertices and declare it the second vertex, etc. But a deterministic algorithm cannot do that.

Algorithms like P are not covered directly by Turing's analysis. But there is an easy patch if you don't care about resources and use parallelism. Let n be the number of vertices. In parallel, the desired algorithm orders the vertices in all $n!$ possible ways and then carries on all $n!$ computations.

Non-discrete computations. Turing dealt with discrete computations. His analysis does not apply directly e.g. to the classical, geometrical ruler-and-compass algorithms. The particular case of ruler-and-compass algorithms can be taken care of; such algorithms do not allow you to compute a non-recursive function [Kijne 1956]. In general, however, it is not clear how to extend Turing's analysis to non-discrete algorithms.

3. Kolmogorov Machines and Pointer Machines

The problem of the absolute definition of algorithm was attacked again in 1953 by Andrei N. Kolmogorov; see the one-page abstract [Kolmogorov 1953] of his March 17, 1953, talk at the Moscow Mathematical Society. Kolmogorov spelled out his intuitive ideas about algorithms. For brevity, we express them in our own words (rather than translate literally).

- An algorithmic process splits into steps whose complexity is bounded in advance, i.e., the bound is independent of the input and the current state of the computation.
- Each step consists of a direct and immediate transformation of the current state.
- This transformation applies only to the active part of the state and does not alter the remainder of the state.
- The size of the active part is bounded in advance.
- The process runs until either the next step is impossible or a signal says the solution has been reached.

In addition to these intuitive ideas, Kolmogorov gave a one-paragraph sketch of a new computation model. The ideas of [Kolmogorov 1953] were developed in the article [Kolmogorov and Uspensky 1958] written by Kolmogorov together with his student Vladimir A. Uspensky. The Kolmogorov machine model can be thought of as a generalization of the Turing machine model where the tape is a directed graph of bounded in-degree and bounded out-degree. The vertices of the graph correspond to Turing's squares; each vertex has a color chosen from a fixed finite palette of vertex colors; one of the vertices is the current computation center. Each edge has a color chosen from a fixed finite palette of edge colors; distinct edges from the same node have different colors. The program has this form: replace the vicinity U of a fixed radius around the central node by a new vicinity W that depends on the isomorphism type of the digraph U with the colors and the distinguished central vertex. Contrary to Turing's tape whose topology is fixed, Kolmogorov's "tape" is reconfigurable.

Remark. We took liberties in describing Kolmogorov machines. Kolmogorov and Uspensky require that the tape graph is symmetric—for every edge (x, y) there is an edge (y, x). The more liberal model is a bit easer to describe. And the symmetry requirement is inessential in the following sense: any machine of either kind can be step-for-step simulated by a machine of the other kind.

Like Turing machines, Kolmogorov machines compute functions from strings to strings; we skip the description of the input and output conventions. In the footnote to the article title, Kolmogorov and Uspensky write that they just wanted to analyze the existing definitions of the notions of computable functions and algorithms and to convince themselves that there is no hidden way to extend the notion of computable function. Indeed, Kolmogorov machines compute exactly Turing computable functions. It seems, however, that they were more ambitious. Here is a somewhat liberal translation from [Kolmogorov and Uspensky 1958, p. 16]:

> To simplify the description of algorithms, we introduced some conventions that are not intrinsic to the general idea, but it seems to us that the generality of the proposed definition re-

> mains plausible in spite of the conventions. 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.

In this connection the second author formulated a Kolmogorov–Uspensky thesis [Gurevich 1988, p. 227]: "every computation, performing only one restricted local action at a time, can be viewed as (not only being simulated by, but actually being) the computation of an appropriate KU machine". Uspensky concurred [Uspensky 1992, p. 396].

Kolmogorov's approach proved to be fruitful. It led to a more realistic complexity theory. For example, given a string x, a Kolmogorov machine can build a binary tree over x and then move fast about x. Leonid Levin used a universal Kolmogorov machine to construct his algorithm for NP problems that is optimal up to a multiplicative constant [Levin 1973; Gurevich 1988]. The up-to-a-multiplicative-constant form is not believed to be achievable for the multitape Turing machine model popular in theoretical computer science. Similarly, the class of functions computable in *nearly linear* time $n(\log n)^{O(1)}$ on Kolmogorov machines remains the same if Kolmogorov machines are replaced e.g. by various random access computers in the literature; it is not believed, however, that the usual multitape Turing machines have the same power [Gurevich and Shelah 1989].

Kolmogorov machines allow one to do reasonable computations in reasonable time. This may have provoked Kolmogorov to ask new questions. "Kolmogorov ran a complexity seminar in the 50s or early 60s," wrote Leonid Levin, a student of Kolmogorov, to us [Levin 2003a]. "He asked if common tasks, like integer multiplication, require necessarily as much time as used by common algorithms, in this case quadratic time. Unexpectedly, Karatsuba reduced the power to $\log_2(3)$ [Karatsuba and Ofman 1963]." (Readers interested in fast integer multiplication are referred to [Knuth 1981].)

It is not clear to us how Kolmogorov thought of the tape graph. One hypothesis is that edges reflect physical closeness. This hypothesis collides with the fact that our physical space is finite-dimensional. As one of us remarked earlier [Gurevich 2000, p. 81], "In a finite-

dimensional Euclidean space, the volume of a sphere of radius n is bounded by a polynomial of n. Accordingly, one might expect a polynomial bound on the number of vertices in any vicinity of radius n (in the graph theoretic sense) of any state of a given KU machine, but in fact such a vicinity may contain exponentially many vertices."

Another hypothesis is that edges are some kind of channels. This hypothesis too collides with the fact that our physical space is finite-dimensional.

Probably the most natural approach would be to think of informational rather than physical edges. If vertex a contains information about the whereabouts of b, draw an edge from a to b. It is reasonable to assume that the amount of information stored at every single vertex a is bounded, and so the out-degree of the tape graph is bounded. It is also reasonable to allow more and more vertices to have information about b as the computation proceeds, so that the in-degree of the tape graph is unbounded. This brings us to Schönhage machines. These can be seen as Kolmogorov machines (in the version with directed edges) except that only the out-degrees are required to be bounded. The in-degrees can depend on the input and, even for a particular input, can grow during the computation.

"In 1970 the present author introduced a new machine model (cf. [Schönhage 1970]) now called *storage modification machine* (SMM)," writes Schönhage in [1980], "and posed the intuitive thesis that this model possesses extreme flexibility and should therefore serve as a basis for an adequate notion of time complexity." In article [Schönhage 1980], Schönhage gave "a comprehensive presentation of our present knowledge of SMMs". In particular, he proved that SMMs are "real-time equivalent" to successor RAMs (random access machines whose only arithmetical operation is $n \mapsto n + 1$). The following definitions appear in [Schönhage 1980, p. 491].

Definition. A machine M' is said to *simulate* another machine M "in real time", denoted $M \xrightarrow{r} M'$, if there is a constant c such that for every input sequence x the following holds: if x causes M to read an input symbol, or to print an output symbol, or to halt at time steps $0 = t_0 < t_1 < \cdots < t_l$, respectively, then x causes M' to act in the very same way with regard to those external actions at time steps $0 = t'_0 < t'_1 < \cdots < t'_l$ where $t'_j - t'_{j-1} \leq c(t_j - t_{j-1})$ for $1 \leq j \leq l$. For machine classes $\mathcal{M}, \mathcal{M}'$ *real time reducibility* $\mathcal{M} \xrightarrow{r} \mathcal{M}'$ is defined by the condition that for each $M \in \mathcal{M}$ there

exists an $M' \in \mathcal{M}'$ such that $M \xrightarrow{r} M'$. *Real time equivalence* $\mathcal{M} \overset{r}{\leftrightarrow} \mathcal{M}'$ means $\mathcal{M} \xrightarrow{r} \mathcal{M}'$ and $\mathcal{M}' \xrightarrow{r} \mathcal{M}$. □

Dima Grigoriev proved that Turing machines cannot simulate Kolmogorov machines in real time [Grigoriev 1980].

Schönhage introduced a precise language for programming his machines and complained that the Kolmogorov–Uspensky description of Kolmogorov machines is clumsy. For our purposes here, however, it is simplest to describe Schönhage machines as generalized Kolmogorov machines where the in-degree of the tape graph may be unbounded. It is still an open problem whether Schönhage machines are real time reducible to Kolmogorov machines.

Schönhage states his thesis as follows: "$\mathcal{M} \xrightarrow{r}$ SMM holds for all *atomistic* machine models $\mathcal{M}$."

Schönhage writes that Donald E. Knuth "brought to his attention that the SMM model coincides with a special type of 'linking automata' briefly explained in volume one of his book (cf. [Knuth 1968, pp. 462–463]) in 1968 already. Now he suggests calling them 'pointer machines' which, in fact, seems to be the adequate name for these automata." Note that Kolmogorov machines also modify their storage. But the name "pointer machine" fits Knuth–Schönhage machines better than it fits Kolmogorov machines.

A successor RAM is a nice example of a pointer machine. Its tape graph consists of natural numbers and a couple of special registers. Each special register has only one pointer, which points to a natural number that is intuitively the content of the register. Every natural number n has only a pointer to $n + 1$, a pointer to another natural number that is intuitively the content of register n, and a pointer to every special register.

The notion of pointer machine seems an improvement over the notion of Kolmogorov machine to us (and of course the notion of Kolmogorov machine was an improvement over the notion of Turing machine). And the notion of pointer machine proved to be useful in the analysis of the time complexity of algorithms. In that sense it was successful. It is less clear how much of an advance all these developments were from the point of view of absolute definitions. The pointer machine reflected the computer architecture of real computers of the time. (The modern tendency is to make computers with several CPUs, central processing units, that run asynchronously.)

Remark. In an influential 1979 article, Tarjan used the term "pointer machine" in a wider sense [Tarjan 1979]. This wider notion of pointer machines has become better known in computer science than the older notion.

4. Related Issues

We mention a few issues touched upon in the talk that was the precursor of this paper. It is beyond the scope of this paper to develop these issues in any depth.

4.1. Physics and Computations

What kind of computations can be carried out in our physical universe? We are not talking about what functions are computable. The question is what algorithms are physically executable. We don't expect a definitive answer soon, if ever. It is important, however, to put things into perspective. Many computer science concerns are above the level of physics. It would be great if quantum physics allowed us to factor numbers fast, but this probably will not greatly influence programming language theory.

Here are some interesting references.

- Robin Gandy attempted to derive Turing's thesis from a number of "principles for mechanisms" [Gandy 1980]. Wilfried Sieg continues this line of research [Sieg 1999].

- David Deutsch [1985] designed a universal quantum computer that is supposed to be able to simulate the behavior of any finite physical system. Gandy's approach is criticized in [Deutsch, Ekert and Lupaccini 2000, pp. 280–281]. Deutsch's approach and quantum computers in general are criticized in [Levin 2003b (sec. 2)].

- Charles H. Bennett and Rolf Landauer pose in [1985] important problems related to the fundamental physical limits of computation.

- Marian Boykan Pour–El and Ian Richards [1989] investigate the extent to which computability is preserved by fundamental constructions of analysis, such as those used in classical and quantum theories of physics.

4.2. Polynomial Time Turing's Thesis

There are several versions of the polynomial time Turing's thesis discussed in theoretical computer science. For simplicity, we restrict attention to decision problems.

To justify the interest in the class P of problems solvable in polynomial time by a Turing machine, it is often declared that a problem is feasible (=practically solvable) if and only if it is in P. Complexity theory tells us that there are P problems unsolvable in time n^{1000}. A more reasonable thesis is that a "natural problem" is feasible if and only if it is in P. At the 1991 Annual Meeting of the Association of Symbolic Logic, Steve Cook argued in favor of that thesis, and the second author argued against it. Some of the arguments can be found in [Cook 1991] and [Gurevich 1993] respectively.

A related but different version of the polynomial time Turing thesis is that a problem is in P if it can be solved in polynomial time at all, by any means. The presumed reason is that any polynomial time computation can be polynomial time simulated by a Turing machine (so that the computation time of the Turing machine is bounded by a polynomial of the computation time of the given computing device). Indeed, most "reasonable" computation models are known to be polytime equivalent to the Turing machine. "As to the objection that Turing machines predated all of these models," says Steve Cook [2003], "I would reply that models based on RAMs are inspired by real computers, rather than Turing machines."

Quantum computer models can factor arbitrary integers in polynomial time [Shor 1997], and it is not believed that quantum computers can be polynomial time simulated by Turing machines. For the believers in quantum computers, it is more natural to speak about probabilistic Turing machines. We quote from [Bernstein and Vazirani 1997].

> Just as the theory of computability has its foundations in the Church–Turing thesis, computational complexity theory rests upon a modern strengthening of this thesis, which asserts that any "reasonable" model of computation can be efficiently simulated on a probabilistic Turing Machine (an efficient simulation is one whose running time is bounded by some polynomial in the running time of the simulated machine). Here, we take reasonable to mean in principle physically realizable.

Turing's analysis does not automatically justify any of these new theses. (Nor does it justify, for example, the thesis that polynomial time interactive Turing machines capture polynomial time interactive algorithms.) Can any of the theses discussed above be derived from first principles? One can analyze Turing's original justification of his thesis and see whether all the reductions used by Turing are polynomial time reductions. But one has to worry also about algorithms not covered directly by Turing's analysis.

4.3. Recursion

According to Yiannis Moschovakis, an algorithm is a "recursor", a monotone operator over partial functions whose least fixed point includes (as one component) the function that the algorithm computes [Moschovakis 2001]. He proposes a particular language for defining recursors. A definition may use various givens: functions or recursors.

Moschovakis gives few examples and they are all small ones. The approach does not seem to scale to algorithms interacting with an unknown environment. A posteriori the approach applies to well understood classes of algorithms. Consider for example non-interactive sequential or parallel abstract state machines (ASMs) discussed below in Sections 5 and 6. Such an ASM has a program for doing a single step. There is an implicit iteration loop: repeat the step until, if ever, the computation terminates. Consider an operator that, given an initial segment of a computation, augments it by another step (unless the computation has terminated). This operator can be seen as a recursor. Of course the recursion advocates may not like such a recursor because they prefer stateless ways.

We are not aware of any way to derive from first principles the thesis that algorithms are recursors.

5. Formalization of Sequential Algorithms

Is it possible to capture (=formalize) sequential algorithms on their natural levels of abstraction? Furthermore, is there one machine model that captures all sequential algorithms on their natural levels of abstraction? According to [Gurevich 2000], the answer to both questions is yes. We outline the approach of [Gurevich 2000] and put forward a slight but useful generalization.

As a running example of a sequential algorithm, we use a version Euc of Euclid's algorithm that, given two natural numbers, computes their greatest common divisor d.

```
1.  Set a = Input1, b = Input2.
2.  If a = 0 then set d = b and go to 1
    else set a, b = b mod a, a respectively and go to 2.
```

Initially Euc waits for the user to provide natural numbers Input1 and Input2. The assignment on the last line is simultaneous. If, for instance, $a = 6$ and $b = 9$ in the current state then $a = 3$ and $b = 6$ in the next state.

5.1. Sequential Time Postulate

A sequential algorithm can be viewed as a finite or infinite state automaton.

Postulate (Sequential Time). A sequential algorithm A is associated with

- a nonempty set $\mathcal{S}(A)$ whose members are called *states* of A,
- a nonempty[1] subset $\mathcal{I}(A)$ of $\mathcal{S}(A)$ whose members are called *initial states* of A, and
- a map $\tau_A : \mathcal{S}(A) \longrightarrow \mathcal{S}(A)$ called the *one-step transformation* of A.

The postulate ignores final states [Gurevich 2000 (sec. 3.3.2)]. We are interested in runs where the steps of the algorithm are interleaved with the steps of the environment. A step of the environment consists in changing the current state of the algorithm to any other state. In particular it can change the "final" state to a non-final state. To make the one-step transformation total, assume that the algorithm performs an idle step in the "final" states. Clearly Euc is a sequential time algorithm. The environment of Euc includes the user who provides input numbers (and is expected to take note of the answers).

[1] In [Gurevich 2000], $\mathcal{I}(A)$ and $\mathcal{S}(A)$ were not required to be nonempty. But an algorithm without an initial state couldn't be run, so is it really an algorithm? We therefore add "nonempty" to the postulate here.

This sequential-time postulate allows us to define a fine notion of behavioral equivalence.

Definition. Two sequential time algorithms are behaviorally equivalent if they have the same states, the same initial states and the same one-step transformation.

The behavioral equivalence is too fine for many purposes but it is necessary for the following.

Corollary. If algorithms A and B are behaviorally equivalent then B step-for-step simulates A in any environment.

The step-for-step character of simulation is important. Consider a typical distributed system. The agents are sequential-time but the system is not. The system guarantees the atomicity of any single step of any agent but not of a sequence of agent's steps. Let A be the algorithm executed by one of the agents. If the simulating algorithm B makes two steps to simulate one step of A then another agent can sneak in between the two steps of B and spoil the simulation.

5.2. Small-Step Algorithms

An object that satisfies the sequential-time postulate doesn't have to be an algorithm. In addition we should require that there is a program for the one-step transformation. This requirement is hard to capture directly. It will follow from other requirements in the approach of [Gurevich 2000].

Further, a sequential-time algorithm is not necessarily a sequential algorithm. For example, the algorithm P in subsection 2.3 is not sequential. The property that distinguishes sequential algorithms among all sequential-time algorithms is that the steps are of bounded complexity. The algorithms analyzed by Turing [1937] were sequential:

> The behavior 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 algorithms analyzed by Kolmogorov in [1953] are also sequential: "An algorithmic process is divided into separate steps of limited complexity."

These days there is a tendency to use the term "sequential algorithm" in the wider sense of the contrary of the notion of a distributed algorithm. That is, "sequential" often means what we have called "sequential-time". So we use the term "small-step algorithm" as a synonym for the term "sequential algorithms" in its traditional meaning.

5.3. Abstract State Postulate

How does one capture the restriction that the steps of a small-step algorithms are of bounded complexity? How does one measure the complexity of a single-step computation? Actually we prefer to think of bounded work instead of bounded complexity. The work that a small-step algorithm performs at any single step is bounded, and the bound depends only on the algorithm and does not depend on input. This complexity-to-work reformulation does not make the problem easier of course. How does one measure the work that the algorithm does during a step? The algorithm-as-a-state-automaton point of view is too simplistic to address the problem. We need to know more about what the states are. Fortunately this question can be answered.

Postulate (Abstract State).

- States of a sequential algorithm A are first-order structures.
- All states of A have the same vocabulary.
- The one-step transformation τ_A does not change the base set of any state.
- $\mathcal{S}(A)$ and $\mathcal{I}(A)$ are closed under isomorphisms. Further, any isomorphism from a state X onto a state Y is also an isomorphism from $\tau_A(X)$ onto $\tau_A(Y)$.

The notion of first-order structure is well-known in mathematical logic [Shoenfield 1967]. We use the following conventions:

- Every vocabulary contains the following *logic symbols:* the equality sign, the nullary relation symbols `true` and `false`, and the usual Boolean connectives.
- Every vocabulary contains the nullary function symbol `undef`.
- Some vocabulary symbols may be marked *static*. The remaining symbols are marked *external* or *dynamic* or both.[2] All logic symbols are static.
- In every structure, `true` is distinct from `false` and `undef`, the equality sign has its standard meaning, and the Boolean connectives have their standard meanings on Boolean arguments.

The symbols `true` and `false` allow us to treat relation symbols as special function symbols. The symbol `undef` allows us to deal with partial functions; recall that first-order structures have only total functions. The static functions (that is the interpretations of the static function symbols) do not change during the computation. The algorithm can change only the dynamic functions. The environment can change only the external functions.

It is easy to see that higher-order structures are also first-order structures (though higher-order logics are richer than first-order logic). We refer to [Gurevich 2000] for justification of the abstract-state postulate. Let us just note that the experience of the ASM community confirms that first-order structures suffice to describe any static mathematical situation [ASM].

It is often said that a state is given by the values of its variables. We take this literally. Any state of a sequential algorithm should be uniquely determined (in the space of all states of the algorithm) by the interpretations of the dynamic and external function symbols.

What is the vocabulary (of the states) of Euc? In addition to the logic symbols, it contains the nullary function symbols `0, a, b, d, Input1, Input2` and the binary function symbol `mod`. But what about labels `1` and `2`? Euc has an implicit program counter. We have

[2] This useful classification, used in [Gurevich 1991; 1995] and in ASM applications, was omitted in [Gurevich 2000] because it wasn't necessary there. The omission allowed the following pathology in the case when there is a finite bound on the size of the states of A. The one-step transformation may change the values of `true` and `false` and modify appropriately the interpretations of the equality relation and the Boolean connectives.

some freedom in making it explicit. One possibility is to introduce a Boolean variable, that is a nullary relational symbol, `initialize` that takes value `true` exactly in those states where Euc consumes inputs. The only dynamic symbols are `a, b, d, initialize`, and the only external symbols are `Input1, Input2`.

5.4. Bounded Exploration Postulate and the Definition of Sequential Algorithms

Let A be an algorithm of vocabulary Υ and let X be a state of A. A *location* ℓ of X is given by a dynamic function symbol f in Υ of some arity j and a j-tuple $\bar{a} = (a_1, ..., a_j)$ of elements of X. The *content* of ℓ is the value $f(\bar{a})$.

An *(atomic) update* of X is given by a location ℓ and an element b of X and denoted simply (ℓ, b). It is the action of replacing the current content a of ℓ with b.

By the abstract-state postulate, the one-step transformation preserves the set of locations, so the state X and the state $X' = \tau_A(X)$ have the same locations. It follows that X' is obtained from X by executing the following set of updates:

$$\Delta(X) = \{(\ell, b) \colon \ b = Content_{X'}(\ell) \neq Content_X(\ell)\}$$

If A is Euc and X is the state where $a = 6$ and $b = 9$ then $\Delta(X) = \{(a, 3), (b, 6)\}$. If Y is a state of A where $a = b = 3$ then $\Delta(Y) = \{(a, 0)\}$.

Now we are ready to formulate the final postulate. Let X, Y be arbitrary states of the algorithm A.

Postulate (Bounded Exploration). There exists a finite set T of terms in the vocabulary of A such that $\Delta(X) = \Delta(Y)$ whenever every term $t \in T$ has the same value in X and Y.

In the case of Euc, the term set {`true`, `false`, $0, a, b, d, b \ mod \ a$, `initialize`} is a bounded-exploration witness.

Definition. A *sequential algorithm* is an object A that satisfies the sequential-time, abstract-state and bounded-exploration postulates.

5.5. Sequential ASMs and the Characterization Theorem

The notion of a sequential ASM rule of a vocabulary Υ is defined by induction. In the following definition, all function symbols (including relation symbols) are in Υ and all terms are first-order terms.

Definition. If f is a j-ary dynamic function symbol and $t_0, ..., t_j$ are first-order terms then the following is a rule:

$f(t_1, ..., t_j) := t_0$.

Let φ be a Boolean-valued term, that is φ has the form $f(t_1, ..., t_j)$ where f is a relation symbol. If P_1, P_2 are rules then so is

`if` φ `then` P_1 `else` P_2`.`

If P_1, P_2 are rules then so is

`do in-parallel`
  P_1
  P_2

The semantics of rules is pretty obvious but we have to decide what happens if the constituents of the `do in-parallel` rule produce contradictory updates. In that case the execution is aborted. For a more formal definition, we refer the reader to [Gurevich 2000]. Syntactically, a sequential ASM program is just a rule; but the rule determines only single steps of the program and is supposed to be iterated. Every sequential ASM program P gives rise to a map $\tau_P(X) = Y$ where X, Y are first-order Υ-structures.

Definition. A sequential ASM B of vocabulary Υ is given by a sequential ASM program Π of vocabulary Υ, a nonempty set $\mathcal{S}(B)$ of Υ-structures closed under isomorphisms and under the map τ_Π, a nonempty subset $\mathcal{I}(B) \subseteq \mathcal{S}(B)$ that is closed under isomorphisms, and the map τ_B which is the restriction of τ_Π to $\mathcal{S}(B)$.

Now we are ready to formulate the theorem of this section.

Theorem [*] (ASM Characterization of Small-Step Algorithms). For every sequential algorithm A there is a sequential abstract state machine B behaviorally equivalent to A. In particular, B simulates A step for step.

If A is our old friend Euc, then the program of the desired ASM B could be this.

```
if initialize then
  do in-parallel
    a := Input1
    b := Input2
    initialize := false
else
  if a = 0 then
    do in-parallel
      d := b
      initialize := true
  else
    do in-parallel
      a := b mod a
      b := a
```

We have discussed only deterministic sequential algorithms. Nondeterminism implicitly appeals to the environment to make the choices that cannot be algorithmically prescribed [Gurevich 2000]. Once nondeterminism is available, classical ruler-and-compass constructions can be regarded as nondeterministic ASMs operating on a suitable structure of geometric objects.

A critical examination of [Gurevich 2000] is found in [Reisig 2003].

6. Formalization of Parallel Algorithms

Encouraged by the success in capturing the notion of sequential algorithms in [Gurevich 2000], we "attacked" parallel algorithms in [Blass and Gurevich 2003]. The attack succeeded. We gave an axiomatic definition of parallel algorithms and checked that the known (to us) parallel algorithm models satisfy the axioms. We defined precisely a version of parallel abstract state machines, a variant of the notion of parallel ASMs from [Gurevich 1995], and we checked that our parallel ASMs satisfy the definitions of parallel algorithms. And we proved the characterization theorem for parallel algorithms: every parallel algorithm is behaviorally equivalent to a parallel ASM.

The scope of this paper does not allow us to spell out the axiomatization of parallel ASMs, which is more involved than the ax-

iomatization of sequential ASMs described in the previous section. We just explain what kind of parallelism we have in mind, say a few words about the axioms, say a few words about the parallel ASMs, and formulate the characterization theorem. The interested reader is invited to read—critically!—the paper [Blass and Gurevich 2003]. More scrutiny of that paper is highly desired.

6.1. What Parallel Algorithms?

The term "parallel algorithm" is used for a number of different notions in the literature. We have in mind sequential-time algorithms that can exhibit unbounded parallelism but only bounded sequentiality within a single step. Bounded sequentiality means that there is an *a priori* bound on the lengths of sequences of events within any one step of the algorithm that must occur in a specified order. To distinguish this notion of parallel algorithms, we call such parallel algorithms *wide-step*. Intuitively the width is the amount of parallelism. The "step" in "wide-step" alludes to sequential time.

Remark. Wide-step algorithms are also bounded-depth where the depth is intuitively the amount of sequentiality in a single step; this gives rise to a possible alternative name *shallow-step algorithms* for wide-step algorithms. Note that the name "parallel" emphasizes the potential rather than restrictions; in the same spirit, we choose "wide-step" over "shallow-step".

Here is an example of a wide-step algorithm that, given a directed graph $G = (V, E)$, marks the well-founded part of G. Initially no vertex is marked.

```
1. For every vertex x do the following.
   If every vertex y with an edge to x is marked
   then mark x as well.
2. Repeat step 1 until no new vertices are marked.
```

6.2. A few Words on the Axioms for Wide-Step Algorithms

Adapt the sequential-time postulate, the definition of behavioral equivalence and the abstract-state postulate to parallel algorithms simply by replacing "sequential" with "parallel". The bounded-

exploration postulate, on the other hand, specifically describes sequential algorithms. The work that a parallel algorithm performs within a single step can be unbounded. We must drop the bounded-exploration postulate and assume, in its place, an axiom or axioms specifically designed for parallelism.

A key observation is that a parallel computation consists of a number of processes running (not surprisingly) in parallel. The constituent processes can be parallel as well. But if we analyze the computation far enough then we arrive at processes, which we call *proclets*, that satisfy the bounded-exploration postulate. Several postulates describe how the proclets communicate with each other and how they produce updates. And there is a postulate requiring some bound d (depending only on the algorithm) for the amount of sequentiality in the program. The length of any sequence of events that must occur in a specified order within any one step of the algorithm is at most d.

There are several computation models for wide-step algorithms in the literature. The two most known models are Boolean circuits and PRAMs [Karp and Ramachandran 1990]. (PRAM stands for "Parallel Random Access Machines".) These two models and some other models of wide-step algorithms that occurred to us or to the referees are shown to satisfy the wide-step postulates in [Blass and Gurevich 2003].

6.3. Wide-Step Abstract State Machines

Parallel abstract state machines were defined in [Gurevich 1995]. Various semantical issues were elaborated later in [Gurevich 1997]. A simple version of parallel ASMs was explored in [Blass, Gurevich and Shelah 1999]; these ASMs can be called BGS ASMs. We describe, up to an isomorphism, an arbitrary state X of a BGS ASM. X is closed under finite sets (every finite set of elements of X constitutes another element of X) and is equipped with the usual set-theoretic operations. Thus X is infinite but a finite part of X contains all the essential information. The number of *atoms* of X, that is elements that are not sets, is finite, and there is a nullary function symbol Atoms interpreted as the set of all atoms. It is easy to write a BGS ASM program that simulates the example parallel algorithm above.

```
forall x ∈ Atoms
  if {y: y ∈ Atoms: E(y,x) ∧¬ (M(y))} = ∅
then M(x) := true
```

Note that x and y are mathematical variables like the variables of first-order logic. They are not programming variables and cannot be assigned values. In comparison to the case of sequential ASMs, there are two main new features in the syntax of BGS ASMs:

- set-comprehension terms $\{t(x)\colon x \in r\colon \varphi(x)\}$, and
- `forall` rules.

In [Blass and Gurevich 2000], we introduced the notion of a background of an ASM. BGS ASMs have a set background. The specification language AsmL, mentioned in the introduction, has a rich background that includes a set background, a sequence background, a map background, etc. The background that naturally arises in the analysis of wide-step algorithms is a multiset background. That is the background used in [Blass and Gurevich 2003].

6.4. The Wide-Step Characterization Theorem

Theorem [**] (ASM Characterization of Wide-Step Algorithms). For every parallel algorithm A there is a parallel abstract state machine B behaviorally equivalent to A. In particular, B simulates A step for step.

Thus, Boolean circuits and PRAMs can be seen as special wide-step ASMs (which does not make them any less valuable). The existing quantum computer models satisfy our postulates as well [Grädel and Nowack 2003] assuming that the environment provides random bits when needed. The corresponding wide-step ASMs need physical quantum-computer implementation for efficient execution.

7. Toward Formalization of Distributed Algorithms

Distributed abstract state machines were defined in [Gurevich 1995]. They are extensively used by the ASM community [ASM] but the problem of capturing distributed algorithms is open. Here we concentrate on one aspect of this important problem: interaction

between a sequential-time agent and the rest of the system as seen by the agent. One may have an impression that this aspect has been covered because all along we studied runs where steps of the algorithm are interleaved with steps made by the environment. But this interleaving mode is not general enough.

If we assume that each agent's steps are atomic, then interleaving mode seems adequate. But a more detailed analysis reveals that even in this case a slight modification is needed. See Subsection 7.1.

But in fact an agent's steps need not be atomic because agents can interact with their environments not only in the inter-step fashion but also in the intra-step fashion. It is common in the AsmL experience that, during a single step, one agent calls on other agents, receives "callbacks", calls again, etc. It is much harder to generalize the two characterization theorems to intra-step interaction.

7.1. Trivial Updates in Distributed Computation

Consider a small-step abstract state machine A. In Section 5, we restricted attention to runs where steps of A are interleaved with steps of the environment. Now turn attention to distributed computing where the agents do not necessarily take turns to compute. Assume that A is the algorithm executed by one of the agents. Recall that an update of a location ℓ of the current state of A is the action of replacing the current content a of ℓ with some content b. Call the update *trivial* if $a = b$. In Section 5 we could ignore trivial updates. But we have to take them into account now. A trivial update of ℓ matters in a distributed situation when the location ℓ is shared: typically only one agent is allowed to write into a location at any given time, and so even a trivial update by one agent would prevent other agents from writing to the same location at the same time.

Recall that $\Delta(X)$ is the set of nontrivial updates computed by the algorithm A at X during one step. Let $\Delta^+(X)$ be the set of all updates, trivial or not, computed by A at X during the one step. It seems obvious how to generalize Section 5 in order to take care of trivial updates: just strengthen the bounded-exploration postulate by replacing Δ with Δ^+. There is, however, a little problem. Nothing in the current definition of a small-step algorithm A guarantees that there is a $\Delta^+(X)$ map associated with it. ($\Delta(X)$ is definable in terms of X and $\tau_A(X)$.) That is why we started this subsection by

assuming that A is an ASM. Euc also has a $\Delta^+(X)$ map: if X is the state where $a = 6$ and $b = 9$ then $\Delta^+(X) = \{(a, 3), (b, 6)\}$, and if Y is a state of A where $a = b = 3$ then $\Delta(Y) = \{(a, 0)\}$ and $\Delta^+(Y) = \{(a, 0), (b, 3)\}$.

To generalize Section 5 in order to take into account trivial updates, do the following.

- Strengthen the abstract-state postulate by assuming that there is a mapping Δ^+ associating a set of updates with every state X of the given algorithm A in such a way that the set of non-trivial updates in $\Delta^+(X)$ is exactly $\Delta(X)$.
- Strengthen the definition of behavioral equivalence of sequential algorithms by requiring that the two algorithms produce the same $\Delta^+(X)$ at every state X.
- Strengthen the bounded exploration postulate by replacing Δ with Δ^+.

It is easy to check that Theorem [*], the small-step characterization theorem, remains valid.

Remark. In a similar way, we refine the definition of wide-step algorithms and strengthen Theorem [**], the wide-step characterization theorem.

Remark. Another generalization of Section 5, to algorithms with the `output` command, is described in [Blass and Gurevich 2003]. The two generalizations of Section 5 are orthogonal and can be combined. The `output` generalization applies to wide-step algorithms as well.

7.2. Intra-Step Interacting Algorithms

During the execution of a single step, an algorithm may call on its environment to provide various data and services. The AsmL experience showed the importance of intra-step communication between an algorithm and its environment. AsmL programs routinely call on outside components to perform various jobs.

The idea of such intra-step interaction between an algorithm and its environment is not new to the ASM literature; external functions appear already in the tutorial [Gurevich 1991]. In simple cases, one

can pretend that intra-step interaction reduces to inter-step interaction, that the environment prepares in advance the appropriate values of the external functions. In general, even if such a reduction is possible, it requires an omniscient environment and is utterly impractical.

The current authors are preparing a series of articles extending Theorems [*] and [**] to intra-step interacting algorithms. In either case, this involves

- axiomatic definitions of intra-step interacting algorithms,
- precise definitions of intra-step interacting abstract state machines,
- the appropriate extension of the notion of behavioral equivalence,
- verification that the ASMs satisfy the definitions of algorithms,
- a proof that every intra-step interacting algorithm is behaviorally equivalent to an intra-step interacting ASM.

Acknowledgment

We thank Steve Cook, John Dawson, Martin Davis, Sol Feferman, Leonid Levin, Victor Pambuccian and Vladimir Uspensky for helping us with references. We thank John Dawson, Sol Feferman, Erich Grädel, Leonid Levin, Victor Pambuccian and Dean Rosenzweig for commenting, on very short notice (because of a tight deadline), on the draft of this paper.

References

ASM, ASM Michigan Webpage, <http://www.eecs.umich.edu/gasm/>, maintained by James K. Huggins.

ASML, The AsmL Webpage, <http://research.microsoft.com/foundations/ AsmL/>.

Bennett, C.H. and Landauer, R. [1985] "Fundamental Physical Limits of Computation", *Scientific American* **253**(1) (July), 48–56.

Bernstein, E. and Vazirani, U. [1997], "Quantum Complexity Theory", *SIAM Journal on Computing* **26**, 1411–1473.

Blass, A. and Gurevich, Y. [1997] "The Linear Time Hierarchy Theorem for Abstract State Machines and RAMs", *Springer Journal of Universal Computer Science* **3**(4), 247–278.

Blass, A. and Gurevich, Y. [2000] "Background, Reserve, and Gandy Machines", Springer Lecture Notes in Computer Science, 1862, pp. 1–17.

Blass, A. and Gurevich, Y. [2003], "Abstract State Machines Capture Parallel Algorithms", *ACM Transactions on Computational Logic* **4**(4), 578–651.

Blass, A., Gurevich, Y., and Shelah, S. [1999], "Choiceless Polynomial Time", *Annals of Pure and Applied Logic* **100**, 141–187.

Börger, E. and Stärk, R. [2003], *Abstract State Machines*, Springer.

Church, A. [1936], "An Unsolvable Problem of Elementary Number Theory", *American Journal of Mathematics* **58**, 345–363. Reprinted in [Davis 1965, pp. 88–107].

Cook, S.A. [1991], "Computational Complexity of Higher Type Functions", Proceedings of 1990 International Congress of Mathematicians, Kyoto, Japan, Springer-Verlag, pp. 55–69.

Cook, S.A. [2003], Private communication.

Davis, M. [1965], "The Undecidable", Raven Press.

Davis, M. [1982] "Why Gödel Didn't Have Church's Thesis", *Information and Control* **54**, 3–24.

Deutsch, D. [1985], "Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer", *Proceedings of the Royal Society*, A **400**, 97–117.

Deutsch, D., Ekert, A., and Lupaccini, R. [2000], "Machines, Logic and Quantum Physics", *The Bulletin of Symbolic Logic* **6**, 265–283.

Gandy, R.O. [1980], "Church's Thesis and Principles for Mechanisms", in *The Kleene Symposium*, (J. Barwise, H.J. Keisler, and K. Kunen eds.), North-Holland, pp. 123–148.

Gandy, R.O. [1988], "The Confluence of Ideas in 1936", in *The Universal Turing Machine: A Half-Century Story*, (R. Herken ed.), Oxford University Press, pp. 55–111.

Gödel, K. [1990], "Collected Works", vol. II, Oxford University Press.

Grädel, E. and Nowack, A. [2003], "Quantum Computing and Abstract State Machines", Springer Lecture Notes in Computer Science, 2589, pp. 309–323.

Grigoriev, D. [1980], "Kolmogorov Algorithms are Stronger Than Turing Machines", *Journal of Soviet Mathematics* **14**(5), 1445–1450.

Gurevich, Y. [1988], "Kolmogorov Machines and Related Issues", in *Current Trends in Theoretical Computer Science*, (G. Rozenberg and A. Salomaa eds.), World Scientific, 1993, pp. 225–234; originally in *Bull. EATCS* **35**(1988).

Gurevich, Y. [1991], "Evolving Algebras: An Attempt to Discover Semantics", in *Current Trends in Theoretical Computer Science*, (G. Rozenberg and A. Salomaa eds.), World Scientific, 1993, pp. 266–292; originally in *Bull. EATCS* **43**(1991).

Gurevich, Y. [1993], "Feasible Functions", *London Mathematical Society Newsletter* **206** (June), 6–7.

Gurevich, Y. [1995], "Evolving Algebra 1993: Lipari Guide", in *Specification and Validation Methods*, (E. Börger ed.), Oxford University Press, pp. 9–36.

Gurevich, Y. [1997], "May 1997 Draft of the ASM Guide", Technical Report CSE–TR–336–97, EECS Department, University of Michigan.

Gurevich, Y. [2000], "For Every Sequential Algorithm There Is an Equivalent Sequential Abstract State Machine", *ACM Transactions on Computational Logic* **1**(1), 77–111.

Gurevich, Y. and Huggins, J.K. [1993], "The Semantics of the C Programming Language", Springer Lecture Notes in Computer Science, 702, 274–308.

Gurevich, Y. and Shelah, S. [1989], "Nearly Linear Time", Springer Lecture Notes in Computer Science, 363, 108–118.

Gurevich, Y. and Spielmann, M. [1997], "Recursive Abstract State Machines", *Springer Journal of Universal Computer Science* **3**(4), 233–246.

Gurevich, Y. [2005], "Interactive Algorithms 2005", Springer Lecture Notes in Computer Science, 3618, pp. 26–38,

(J. Jedrzejowicz and A. Szepietowski eds.). An extended-with-an-appendix version is to be published in *Interactive Computation: The New Paradigm*, (D. Goldin, S. Smolka, and P. Wegner eds.), Springer-Verlag.

Karatsuba, A. and Ofman, Y. [1963], "Multiplication of Multidigit Numbers on Automata", *Soviet Physics Doklady* (English translation) **7**(7), 595–596.

Karp, R.M. and Ramachandran, V. [1990], "Parallel Algorithms for Shared-Memory Machines", in *Handbook of Theoretical Computer Science*, vol. A: *Algorithms and Complexity*, (J. van Leeuwen ed.), Elsevier and MIT Press, pp. 869–941.

Kijne, D. [1956], "Plane Construction Field Theory", Van Gorcum, Assen.

Kleene, S.C. [1938], "On Notation for Ordinal Numbers", *Journal of Symbolic Logic* **3**, 150–155.

Kleene, S.C. [1981], "Origins of Recursive Function Theory", *Annals of the History of Computing* **3**(1) (January), 52–67.

Kleene, S.C. [1988], "Turing's Analysis of Computability, and Major Applications of It", in *The Universal Turing Machine: A Half-Century Story*, (R. Herken ed.), Oxford University Press, pp. 17–54.

Knuth, D.E. [1968] *The Art of Computer Programming*, vol. 1: *Fundamental Algorithms*, Addison-Wesley, Reading, MA.

Knuth, D.E. [1981], *The Art of Computer Programming*, vol. 2: *Seminumerical Algorithms*, Addison-Wesley, Reading, MA.

Kolmogorov, A.N. [1953], "On the Concept of Algorithm", *Uspekhi Mat. Nauk* **8**(4), 175–176, Russian. An English translation is found in [Uspensky and Semenov 1993, pp. 18–19].

Kolmogorov, A.N. and Uspensky, V.A. [1958], "On the Definition of Algorithm", *Uspekhi Mat. Nauk* **13**(4), 3–28, Russian. Translated into English in *AMS Translations* **29**(1963), 217–245.

Levin, L.A. [1973], "Universal Search Problems", *Problemy Peredachi Informatsii* **9**(3), 265–266, Russian. The journal is translated into English under the name *Problems of Information Transmission*.

Levin, L.A. [2003a], Private communication.

Levin, L.A. [2003b], "The Tale of One-Way Functions", *Problemy Peredachi Informatsii* **39**(1), 92–103, Russian. The journal is translated into English under the name *Problems of Information Transmission*. The English version is available online at <http://arXiv.org/ abs/ cs.CR/ 0012023>.

Moschovakis, Y.N. [2001], "What Is an Algorithm?" in *Mathematics Unlimited*, (B. Engquist and W. Schmid eds.), Springer-Verlag, pp. 919–936.

Pour–El, M.B. and Richards, I. [1989], "Computability in Analysis and Physics", (Perspectives in Mathematical Logic), Springer-Verlag.

Reisig, W. [2003], "On Gurevich's Theorem on Sequential Algorithms", *Acta Informatica* **39**, 273–305.

Schönhage, A. [1970], "Universelle Turing Speicherung", in *Automatentheorie und Formale Sprachen*, (J. Dörr and G. Hotz eds.), Bibliogr. Institut, Mannheim, pp. 369–383. In German.

Schönhage, A. [1980], "Storage Modification Machines", *SIAM Journal on Computing* **9**, 490–508.

Shoenfield, J.R. [1967], "Mathematical Logic", Addison-Wesley.

Shor, P.W. [1997], "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *SIAM Journal on Computing* **26**(5), 1484–1509.

Sieg, W. [1997], "Step by Recursive Step: Church's Analysis of Effective Calculability", *The Bulletin of Symbolic Logic* **3**(2), 154–180.

Sieg, W. [1999], "An Abstract Model for Parallel Computations: Gandy's Thesis", *The Monist* **82**(1), 150–164.

Tarjan, R.E. [1979], "A Class of Algorithms Which Require Nonlinear Time to Maintain Disjoint Sets", *Journal of Computer and System Sciences* **18**, 110–127.

Turing, A.M. [1937], "On Computable Numbers, With an Application to the *Entscheidungsproblem*", *Proceedings of London Mathematical Society*, Series 2 **42**(1936–1937), 230–265; correction, *ibidem* **43**, pp. 544–546. Reprinted in [Davis 1965, pp. 155–222] and available online at <http://www.abelard.org/turpap2/tp2-ie.asp>.