European Journal for Philosophy of Science (2021) 11:21
https://doi.org/10.1007/s13194-020-00337-2

PAPER IN GENERAL PHILOSOPHY OF SCIENCE

Explication as a Three-Step Procedure: the case of the Church-Turing Thesis

Matteo De Benedetto[1]

Received: 17 March 2020 / Accepted: 24 November 2020 / Published online: 06 January 2021

Abstract
In recent years two different axiomatic characterizations of the intuitive concept of effective calculability have been proposed, one by Sieg and the other by Dershowitz and Gurevich. Analyzing them from the perspective of Carnapian explication, I argue that these two characterizations explicate the intuitive notion of effective calculability in two different ways. I will trace back these two ways to Turing's and Kolmogorov's informal analyses of the intuitive notion of calculability and to their respective outputs: the notion of computorability and the notion of algorithmability. I will then argue that, in order to adequately capture the conceptual differences between these two notions, the classical two-step picture of explication is not enough. I will present a more fine-grained three-step version of Carnapian explication, showing how with its help the difference between these two notions can be better understood and explained.

Keywords Explication · Church-turing thesis · Foundational analyses of computability · Turing · Kolmogorov

1 Introduction

The Church-Turing Thesis (CTT) expresses the proposition that our intuitive notion of effective calculability is captured by the formal notion of mechanical computabil-

✉ Matteo De Benedetto
matteo.debenedetto@lrz.uni-muenchen.de

[1] Fakultät für Philosophie, Wissenschaftstheorie und Religionswissenschaft, Munich Center for Mathematical Philosophy (MCMP), Ludwig-Maximilians-Universität München, Geschwister-Scholl-Platz 1, D-80539, München, Germany

ity[1]. The notion of mechanical computability can be instantiated by one of the many different formal notions of computability, all of which are extensionally equivalent to each other: Turing-computability, general recursiveness, Post-computability, λ-definability, and many others. This plethora of extensionally equivalent formal notions designed to capture a single intuitive concept naturally presents itself as a fantastic case study for philosophers interested in understanding the processes of conceptual change and conceptual formalization in mathematics.

A philosophical question that naturally arises in connection with CTT is then: how can one be sure that a precise formal notion adequately captures an intuitive informal notion? In philosophical literature, one can find several frameworks that aim to conceptualize this process of capturing an intuitive notion via a formal one. Amongst them, Carnap's concept of explication (Carnap 1950; Carus 2007) is particularly salient and useful. Recently, the significance of explication as a philosophical method has been appreciated, both in the context of Carnap's mature thought (Carus 2007; Wagner 2012; Leitgeb and Carus 2020) and in a broader meta-philosophical context (Kuipers 2007; Justus 2012; Brun 2016).

In this work, I will analyze several instances of mechanical computability from the perspective of Carnapian explication. Specifically, I will focus on two recent axiomatic approaches to CTT: Sieg's axioms for computors (Sieg 2013) and Dershowitz's & Gurevich's axioms for computations (Dershowitz and Gurevich 2008). In recent years, building upon Robin Gandy's seminal work(s) in the 1980s, Wilfried Sieg has proposed axioms for human and mechanical computers. In 2008, Dershowitz & Gurevich, coming from a completely different background, proposed another axiomatic characterization of computability. Following Gurevich, I will refer to these axiomatizations as "Foundational Analyses of Computability" (Gurevich 2012).

These axiomatic characterizations of computability will be analyzed in a purely conceptual way, focusing on how they explicate the intuitive notion of effective calculability. Consistently with the pluralism intrinsic to a Carnapian perspective, there will be no discussion about the provability of CTT and related debates in the philosophy of computability.

We will see that, from the perspective of Carnapian explication, these two foundational analyses of computability differ in how they clarify and restrict the boundaries of the intuitive notion of computability. These two different ways of analyzing the notion of effective calculability can be traced back to two pioneers of computability, Turing (1936) and Kolmogorov (1953). I will argue that the main conceptual difference between them lies in the different outputs of their respective informal analyses of effective calculability, i.e. two informal axiomatizations of what is effectively calculable. These axiomatizations implicitly define two semi-formal notions of computability: *computorability* and *algorithmability*.

[1] I use 'mechanical computability' as a neutral term of art to denote all the extensionally equivalent notions for which one can state a version of CTT, e.g. general recursiveness, Turing computability, λ-definability, etc. 'Effective calculability' denotes instead the intuitive notion of calculability.

I will then argue that, in order to adequately capture the conceptual differences between these two semi-formal notions, the classical two-step picture of explication is not enough. A more fine-grained three-step version of Carnapian explication will thus be presented. I will call the new mid-level step of this refined version of explication the *semi-formal sharpening of the clarified explicandum*. Thanks to this three-step version of Carnapian explication the notions of computorability and algorithmability can be understood as two different ways of semi-formally sharpening the clarified notion of effective calculability.

The main aim of this work is then, two-fold: to improve the understanding of explications of effective calculability and to sharpen the received view of Carnapian explication. I will show how the refined three-step version of Carnapian explication achieves a double pay-off corresponding to these two aims.

First, the three-step Carnapian explication allows a more fine-grained conceptual treatment of foundational analyses of computability. Specifically, the new mid-level step explains the conceptual differences between Sieg's axioms for computors and Dershowitz's & Gurevich's axioms for computations, explaining their differences in terms of the incompatibility between Turing's and Kolmogorov's ways of semi-formally sharpening the clarified explicandum. This explanation clarifies also some related philosophical debates over these analyses and stresses the conceptual originality of Kolmogorov's seminal work on computability.

Secondly, as the case of CTT shows, the three-step version of Carnapian explication seems a better tool than the original two-step version for treating complex cases of explications with several explicata for a given explicandum. By adding the new mid-level step of the semi-formal sharpening of a clarified notion between the clarification of the intuitive concept and the formulation of a new one, this refined version of Carnapian explication allows more fine-grained distinctions between different explications of a given concept. Even two explications that clarify the same intuitive concept in the same way (i.e. they output the same clarified explicandum after the clarification step), such as Turing's and Kolmogorov's "analyses" of effective calculability, can be semi-formally sharpened in different ways (i.e. they output two different concepts after the semi-formal sharpening step). The new mid-level step also provides further evidence for the importance of the intermediate steps in the evolution of an explication (cf. Quinon 2019). These intermediate steps have often been overlooked in philosophical discussions about explication, which have often focused only on the explicandum and the explicatum, but the case of CTT shows that without analyzing these steps it is sometimes impossible to properly understand the conceptual differences between two explicata.

In Section 2, the notion of Carnapian explication in its original two-step version will be presented. In Section 3, I will present, discuss, and compare Sieg's axioms for computors and Dershowitz's & Gurevich's axioms for computations through the lens of Carnapian explication. Their conceptual differences will be traced back, respectively, to Turing's and Kolmogorov's seminal ways of explicating effective calculability and more specifically to the notions of computorability and algorithmability. In Section 4, I will argue that the two-step canonical explication cannot capture the differences between these two notions. I will then present a refined three-step

version of Carnapian explication and I will show how this more fine-grained version is able to adequately conceptualize the differences between the notion of computorability and the notion of algorithmability. Finally, I will draw some general conclusion on the significance of this refined three-step version of explication for philosophical debates about CTT and explication.

2 The Carnapian concept of explication

In Carnap's own words:

> "By the procedure of *explication* we mean the transformation of an inexact, prescientific concept, the *explicandum*, into a new exact concept, the *explicatum*." Carnap (1950, p. 3. Original emphases).

Explication is a procedure involving two concepts. On one side, there is the explicandum, belonging to natural language (or more generally an evolved language), the scope of which thus contains arguably an amount of vagueness and ambiguity. On the other side, there is the explicatum, belonging to a (more) precise language, the scope of which is rigidly characterized by explicit and precise rules of use.

In recent years there has been a renewal of interest in explication as a philosophical method. Significant parts of the philosophical debate over explication have focused on the differences between explication and other philosophical methods (Floyd 2012; Gustafsson 2014), on the desiderata that a good explicatum ought to exhibit (Brun 2017; Dutilh Novaes and Reck 2017), on the alleged shortcomings of explication as a general method for philosophy (Strawson 1963; Reck 2012; Dutilh Novaes and Reck 2017), and on the possibility of explicating the concept of explication itself (Hanna 1967; Brun 2016; De Benedetto 2020). In all these discussions, most of the focus has been put onto the explicandum and the explicatum, i.e. the starting and the ending point of the procedure of explicating a given concept. In contrast, only few scholars have tried to analyze what happens during the transformation of a given explicandum into an explicatum (Shepherd and Justus 2015; Quinon 2019). In this paper I will focus on the breakdown of explication as the step-by-step transformation of an intuitive concept into a more precise one, in order to understand where exactly in this transformation process the conceptual differences between different foundational analyses of computability lie. In order to do this, I will take an almost non-committal stance towards the aforementioned recent debates on explication, assuming just that explication is a useful philosophical method and that all the explicata of effective calculability that I analyze are at least minimally satisfactory explications of the intuitive concept.

Explication is traditionally seen as a two-steps procedure. First of all, one has to clarify the explicandum, trying to explicitly state the intended meaning of the concept that one wants to explicate. Since the explicandum is still expressed in a natural language, an exact definition is not required. What (Carnap 1950, pp. 3-5) requires from the explicator, instead, is to state some positive and negative instances of the explicandum, together with some description or (partial) rules of use.

This step clarifies and (if necessary) disambiguates the concept that one seeks to explicate. It is, in fact, possible that in trying to clarify the explicandum, the explicator realizes that there are two or more different concepts that are ambiguously grouped in natural language within a single notion. A famous example of this phenomenon occurred in Carnap's (1950) explication of probability. In clarifying this concept, Carnap in fact realized that behind the intuitive understanding of probability lie two different notions, the logical and the frequentist concepts of probability. In clarifying the explicandum, the explicator also freely chooses the context of the explicandum that she wants to explicate. It is, in fact, often the case that a given explication wants to replace only some contexts or uses of the intuitive notion. A classical example of this decision of context are Tarski's opening remarks (Tarski 1933) before his explication of the concept of truth, where he states that he is interested in explicating the context of truth-assertions like "'snow is white' is true" and not in explicating uses such as "you are a true friend".

Then, there is the second step of the explication, i.e. the formulation of the explicatum in a certain target theory via an explicit definition or by stating its rules of use (Fig. 1).

The purpose of explication is the substitution, relative to a specific function-context, of an inexact concept with a (more) precise one. Explication is an inherently pragmatic procedure, i.e. its adequacy is not a matter of right or wrong, but of what is more or less satisfactory for the task that the explicator has in mind. Judging this adequacy is then never an all or nothing matter. The explicator has always a certain degree of freedom in choosing the explicatum for substituting a given concept. In Carnap's late terminology, as Stein stressed, questions about explication adequacy are thus external questions:

> "The explicatum, as an exactly characterized concept, belongs to some formalized discourse – some 'framework'. The explicandum (...) belongs ipso facto to a mode of discourse outside that framework. Therefore any question about the relation of the explicatum to the explicandum is an 'external' question; this holds, in particular, of the question whether an explication is adequate." Stein (1992, p. 280).

Even though explication is not a matter of right or wrong, one can still judge whether an explication is a good one or a bad one. In fact, external questions for Carnap can still be objects of rational discourse, although of the pragmatic kind of rationality that is often called instrumental. Relative to a specific purpose or function, one can state certain pragmatic meta-principles that a concept has to respect in order to qualify as a good explicatum for a certain explicandum. Carnap (1950, pp. 5-8) stated four desiderata that a good explicatum has to respect:

- Similarity: to the extent to which the other desiderata allow it, the explicatum ought to be as much as similar to the explicandum (exact similarity, i.e. identity, is explicitly not required).

Fig. 1 The two-step structure of Carnapian explication

- Fruitfulness: the explicatum ought to be connected with other scientific concepts, in order to make as many generalizations as possible expressible within the theory in which it is framed.
- Exactness: rules of use of the explicatum ought to be stated in an exact form (e.g. definitions, axioms).
- Simplicity: the explicatum ought to be as simple as the other desiderata allow it to be.

These principles give a hint of the virtues that a good explicatum has to possess, but they are intrinsically pluralist in their intent. Carnap, in fact, stresses how it is always possible to have different explicata that are equally adequate in respect to a given explicandum. In matters of explication, there is no absolutely correct answer to the problem of capturing an informal notion with a (more) formal one.

3 Two foundational analyses of computability

I am now going to apply the Carnapian notion of explication to the case of CTT, more specifically to the case of two recent foundational analyses of computability, one due to Sieg and the other by Gurevich and Dershowitz. Carnapian explication has been previously applied to two classical explications of effective calculability, namely Turing computability (Floyd 2012) and general recursiveness (Hanna 1967; Quinon 2019), but not to more recent models of computability[2]. In what follows, I will focus mostly on how the concept of effective calculability gets transformed in the explication process.

3.1 The Turing-Gandy-Sieg explications

In the introduction, I said that Sieg presented axioms for calculability. More exactly, he gave axioms for calculators, both human and mechanical ones. Why did he state his explication in terms of calculators? In Sieg's own words, "to investigate calculations is to analyze symbolic processes carried out by calculators; that is a lesson we owe to Turing" (Sieg 2002a, p. 390). Then, in order to properly understand Sieg's work one has to grasp the significance of concepts such as human calculator, mechanical device, and symbolic process.

3.1.1 Turing's analysis and Gandy's principles for mechanisms

Turing was the first to put the notion of an abstract calculator at the center of an explication of effective calculability. In fact, before Turing, with the important exception of Post's unpublished drafts (Post 1941), one cannot find any (significant) occurrence of the term 'computor' or 'calculator' in any seminal work on computability. Even

[2]Considering the robustness and the success of the concept of mechanical computability in capturing an intuitive notion, one may wonder why Carnap does not mention CTT as a paradigmatic example of explication. For an analysis of this issue and a possible answer to it, see Quinon (2019).

though the scope of their explications was human effective calculations, pre-Turing explications of calculability, such as Church's (Church 1936), Gödel's (1934), and Kleene's (1936) do not take in account any sort of calculator[3]. Before Turing, the problem of adequately defining mechanical computability was centered around the possible ways in which numeric functions can be deduced in a suitable formalism, i.e. 'calculability in a logic' (Church 1936; Sieg 1997).

Turing changed the level of analysis, understanding that in order to properly clarify the intuitive notion of effective calculability, one has to work not on the superficial calculus of the numerical functions, but instead on the symbolic processes underneath:

> "The real question at issue is 'What are the possible processes which can be carried out in computing a number?'" (Turing 1936, p. 135)

Changing the question allowed Turing to put the computor at the center of his explication of effective calculability. For Turing, a function is computable if and only if it can be computed by an idealized human being working in a clerical fashion. Restrictions on the calculation process can then be stated as bounds on the possible actions of the abstract computor, motivated by human physical-cognitive limitations.

Turing imagined a computor working on a one-dimensional paper, something like 'a child's arithmetic book', printing only a finite number of symbols. Infinity is forbidden because "if we were to allow an infinity of symbols, then there would be symbols differing to an arbitrary small extent" (Turing 1936, p. 135). Ruling out any form of ingenuity, the behavior of the abstract computor is fully determined by the symbols at which she is looking at the moment, together with his state of mind. Actions of the Turing computor have, then, to meet some natural, physical and cognitional, limitations: only a fixed number of symbols can be observed at one glance by the computor, only a fixed number of states of mind can be involved in the calculation, all the actions of the computor can be divided into elementary operations, and so on.

In Sieg's reconstruction of Turing's analysis, these limitations are summarized by three general bounds[4]:

- "(B) (Boundedness) There is a fixed bound on the number of configurations a computor can immediately recognize.
- (L) (Locality) A computor can change only immediately recognizable (sub-) configurations.
- (D) (Determinacy) The immediately recognizable (sub-)configuration determines uniquely the next computational step (and id)." (Sieg 2002a, pp. 249-250).

Turing concluded his explication of effective computability formulating the fundamental notion of a Turing machine. One of Sieg's explicit aims is then to sharpen

[3]Following Gandy, I take the term 'computor' to denote an idealized human calculator. I use the terms 'computer' and 'calculator' to refer to any kind of computing agent, being it a human or a mechanical device of some sort.

[4]Sieg, following Post, imposes these bounds on instantaneous descriptions (ids). For my analysis, this makes no difference.

Turing's informal argument via the axiomatic method. This is why he stated his foundational analysis in terms of calculations by abstract calculators. The other explicit aim of Sieg's axioms is to axiomatically characterize another, more general class of calculators, i.e. mechanical devices. Where in the case of computors his work builds upon Turing's analysis, in the case of mechanical devices Sieg attacked the problem by improving and simplifying Robin Gandy's treatment of machine computability[5].

The term machine is understood by Gandy "with its nineteenth century meaning" (Gandy 1980, p. 125). More specifically, he thinks that machines are discrete deterministic mechanical devices. With this wording, he excludes analogue machines and other devices the calculations of which cannot be described in discrete terms or that are not entirely deterministic. Examples of machines in Gandy's restricted sense are Turing machines, Von Neumann's crystalline automata and John Conway's 'the game of life'. According to Gandy, the main difference between a computor and a mechanical device is that the latter is able to act in parallel, performing a finite though unbounded number of bounded computations at the same time. Any machine of this kind must satisfy four principles.

The first principle states how one should be able to describe a machine. Gandy chooses (isomorphic classes of) hereditarily finite sets to suitably describe each state of the machine. From a given machine state, a structural operation gives us the next state of the machine. After having stated the form of description of his machines, Gandy imposes three different bounds on their working. These limitations should be understood as aiming to avoid the possibility of an omniscient device. Principle II expresses the requirement that the hierarchy of structures describing the machine has a maximum height. Principle III instead requires that any device can be uniquely reassembled from parts of bounded size. Gandy himself remarked that almost any kind of machine (in the general sense of the term) can be described in a way for which Principle II and III are satisfied. The most important principle is then the fourth, the Principle of Local Causality, which contains Gandy's core idea of how a machine computes:

> "Principle IV. (Preliminary version [of the Principle of Local Causality]) The next state, Fx, of a machine can be reassembled from its restrictions to overlapping 'regions' s and these restrictions are locally caused. That is, for each region s of Fx there is a 'causal neighborhood' $t \subseteq TC(x)$ of bounded size such that $Fx \uparrow s$ depends only on $x \uparrow t$." Gandy (1980, p. 135).

This principle forbids globally instantaneous signal propagation in the machine, allowing it only for locally determined regions from which a given state depends. This principle mirrored, in a more general way, both the locality and the boundedness conditions imposed by Turing on human calculations. Where in the case of human computors those restrictions were motivated by an appeal to human memory and intellectual natural limitations, Gandy motivates his fourth principle appealing

[5]Note that the focus of this work will remain the notion of effective calculability and its explications. I will talk about machine computability only in relation to mechanical computability, since Gandy's and Sieg's work on machine calculators is intertwined with their explication of effective calculability.

to physics, specifically to the ban of instantaneous action at distance contained in General Relativity theory. This impossibility, together with Gandy's second physical assumption, namely that there is a lower bound on the size of distinguishable atomic components of the machine, provides the pivotal finiteness of the causal neighborhood on which the machine at every step of the computation operates[6].

We have now everything that we need in order to properly understand Sieg's axioms for calculators. Let us turn to them, now.

3.1.2 Sieg's axioms for computors and the notion of computorability

Sieg has presented his axioms in various works throughout the last two decades Sieg (2002a, b, 2009). Throughout the years, Sieg has changed the presentation of his axioms. Technically speaking, he repeatedly simplified his set-theoretic framework by changing some secondary definitions of operations and sets. Philosophically speaking, the latest presentations of his axioms place more emphasis on Turing's symbolic view of calculation than on the boundedness and the locality conditions, thereby making Turing's ideas appear closer to Post than they are usually considered to be[7]. However, the conceptual core of his work has remained steadily the same for almost two decades and this is what matters for my analysis.

For the computor case, Sieg reformulates Turing's informal limitations, emphasizing the pivotal role of working with finite symbolic configurations, à la Post. According to him, the following aspects of the computors' work are what characterize them:

- (i) "they operate deterministically on finite configurations
- (ii) they recognize in each configuration exactly one pattern (from a bounded number of different kinds of such)
- (iii) they operate locally on the recognized patterns
- (iv) they assemble the next configuration from the original one and the result of the local operation." Sieg (2009, p. 587).

These four semi-formal bounds axiomatically define the notion that I call *computorability*, i.e. the mid-level notion of computability obtained via Turing analysis.

The mathematical formulation of Sieg's explicata on suitably defined systems uses the concept of 'discrete dynamical system', i.e. a system described by a class of syntactical configurations D together with an operation $F : D \to D$[8]. States are presented within the same set-theoretic framework of Gandy's work, i.e. hereditarily finite sets. Abstracting from specific representations, states of the computational processes are represented by structural classes. A *structural class* S is a class of states closed under $\in$-isomorphisms. In order to achieve a real abstract description of

[6]For a more detailed discussion of Gandy's constraints and his underlying physical assumptions, see Sieg and Byrnes (1999).

[7]That said, it should not be thought that Sieg equates Turing's and Post's takes on computability. See, in this respect, Sieg et al. (2016).

[8]For a full presentation of the technical details of Sieg's axioms, see Sieg (2002a).

the system, Sieg makes all operations G work structurally. Thus, a computor can be described with a pair $\langle S, G\rangle$, where S is a structural class and G a structural operation on S. In order to express the ability of the computor to assemble a new state from the actual state, a process I believe to be philosophically crucial in Sieg's take on computability, and the two Turing bounds imposed on the actions of the computor, Sieg introduces two important notions: causal neighborhood (Cn) and determined regions (Dr). A causal neighborhood is a member of a fixed finite class of isomorphism types, such that there does not exist another member into which it is $\in$-embeddable. The determined regions of a state z ($Dr(z, x)$) are, instead, a set of states which are, roughly speaking, isomorphic over a certain causal neighborhood of a given state x and their new atoms structurally correspond to each other. We can finally see the definition of a Turing computor, suitably representing Turing locality and boundedness conditions:

> "$M = \langle S; T, G\rangle$ is a *Turing computor on* S, where S is a structural class, T a finite set of stereotypes, and G a structural operation on $\cup T$, if and only if, for every $x \in S$ there is a $z \in S$, such that:
>
> (LC.0) $(\exists!y)y \in Cn(x)$
>
> (LC.1) $(\exists!v \in Dr(z, x))v \cong_x G(Cn(x))$
>
> (A.1) $z = (x \setminus Cn(x)) \cup Dr(z, x)$." Sieg (2009, p. 589).

Sieg proposes an analogous set-theoretic definition of a Gandy machine, straightforwardly extending this framework to the case of a mechanical device (Sieg 2009, p. 591).

One can now see how Sieg's explication achieves its two intended aims. He manages to axiomatically characterize the informal bounds imposed by Turing in his analysis of the intuitive concept of effective calculability. The boundedness and the locality conditions are expressed through a uniqueness restriction to the causal neighborhood of the state at which the computor is looking and to the determined regions considerable by the computor in the process of assembling the next state. In the case of a Gandy machine, these conditions are loosened up in order to allow the computer to perform an unbounded number of bounded computations at the same time. This looseness shows how the concept of a discrete deterministic mechanical device generalizes the concept of a computor.

I can then reconstruct this group of explications of effective calculability in terms of Carnapian explication. The pivotal first-step, what is usually known as Turing analysis, is the clarification of the explicandum. The intuitive notion of effective calculability gets clarified and disambiguated, the context of uses and the scope of the concept becomes clearer. The output of this clarification is what is defined by Sieg's semi-formal axioms for computors, what I have called computorability. This was firstly achieved informally by Turing, recognized (and used as a basis for the case of machine calculability) by Gandy, and finally made explicit by Sieg. Then, from the notion of computorability, different explicata can be formulated, such as Turing machines or the Gandy-Sieg set-theoretical notions of human calculators. Here is a diagram of the conceptual structure of this group of explications (Fig. 2):

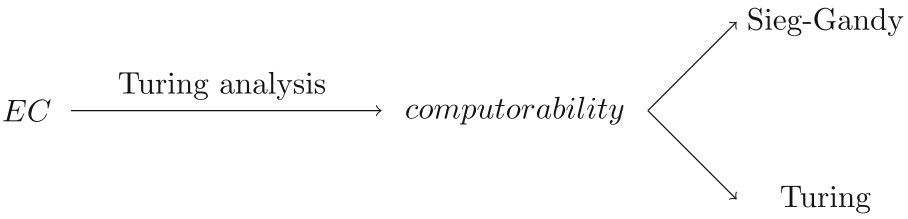

Fig. 2 The two-step structure of the Turing-Gandy-Sieg explications of effective calculability

I have thus presented Sieg's foundational analysis of computability, together with the related works of Turing and Gandy, showing how it can be understood as a Carnapian explication. I will now turn to Dershowitz's & Gurevich's alternative foundational analysis.

3.2 The Kolmogorov-Dershowitz-Gurevich explications

As in Sieg's case, Dershowitz's & Gurevich's axiomatization (Dershowitz and Gurevich 2008) of the intuitive properties of calculability is the result of decades of work on computability. I will thus recall Gurevich's previous work on computability, specifically focusing on the history, the ideas, and the aims of the ASM project[9], the results of which led to Dershowitz's & Gurevich's axiomatization.

According to Gurevich's own reconstruction, the original idea behind the ASM project was to provide an operational semantics for algorithms by elaborating on what he calls the "implicit Turing thesis" (Gurevich 1991, p. 267), i.e. the proposition stating that every algorithm can be simulated by an appropriate Turing machine. It was the aim of finding a more efficient simulation of algorithms, something closer and more faithful to computer scientists' actual view of algorithms, that moved him to start the ASM project.

In order to achieve a better simulation of algorithms, the main conceptual idea is to work at an arbitrary abstraction level, thereby considering any algorithm as an independent entity, without imposing a specific data-representation level on its simulation. This conceptual approach to explicating effective calculability can be traced back to Kolmogorov's take on computability. Gurevich repeatedly highlighted the significance of Kolmogorov's work for his own approach to computability. I take the connection between Gurevich's and Kolmogorov's work to be crucial to understand Dershowitz's & Gurevich's axiomatization of calculability. Since Kolmogorov's take on computability is not so well known, I will briefly present his ideas on computability.

3.2.1 Kolmogorov's hidden analysis

If Turing's approach to the problem of explicating the notion of effective calculability was to look at the possible actions of the computor performing the symbolic

[9] ASM stands for Abstract State Machines, previously known also as Evolving Algebras. As the name tells us, these are idealized machines designed to emulate various classes of algorithms.

calculation process, Kolmogorov instead tried to impose restrictions onto the evolution of the computation in a different way. Together with his own student Uspenski, he seemed to be conceptually dissatisfied with the approach of Turing and the other pioneers of the 1936 confluence to computability. He wanted to find a new approach, trying to apprehend the concept of an algorithmic process in all its generality (Kolmogorov and Uspenski 1958, pp. 62, 68; Uspensky and Semyonov 1993, pp. 253, 258).

Uspenski tells us that a fundamental idea in Kolmogorov's work on computability was to view calculation steps as objects, analyzable independently from any computer whatsoever (Uspensky and Semyonov 1993, pp. 253-254; Uspensky 1992, p. 395). Komogorov tried to outline the structure of an arbitrary algorithm, without reference to the specific agent computing it[10].

Kolmogorov proposed what Uspenski calls a "philosophical scheme" (Uspensky 1992, p. 394), i.e. a semi-formal axiomatic characterization of what an algorithm is:

1. "An algorithm Γ applied to any 'condition' ('initial state') A from some set $D(\Gamma)$ ('domain of applicability' of the algorithm Γ) gives a 'solution' ('concluding state') B.
2. The algorithmic process may be subdivided into separate steps of apriori bounded complexity; each step consists of an 'immediate processing' of the state S (that occurs at this step) into the state $S^* = \Omega_\Gamma(S)$.
3. The processing of $A^0 = A$ into $A^1 = \Omega_\Gamma(A^0)$, A^1 into $A^2 = \Omega_\Gamma(A^1)$, A^2 into $A^3 = \Omega_\Gamma(A^2)$, etc., continues until either a nonresultative stop occurs (if the operator Ω_Γ is not defined for the state that just appeared) or until the signal saying that the 'solution' has been obtained occurs. It is not excluded that the process will continue indefinitely (if the signal for the solution's appearance never occurs).
4. Immediate processing of S into $S^* = \Omega_\Gamma(S)$ is carried out only on the basis of information on the form of an apriori limited 'active part' of the state S and involves only this active part." (Kolmogorov 1953).

At first glance, the philosophical originality of Kolmogorov's take is not evident. Kolmogorov's philosophical scheme seems another definition of what I called computorability, i.e. the output of Turing's praised analysis. After all, does not Kolmogorov, as Turing did first, sharpen effective calculability by imposing a boundedness and a locality condition? Is his proposal, then, just another example of Turing's praised way of thinking about computability? With the help of the previous philosophical considerations, a second look at Kolmogorov's axiomatization shows that this is not the case.

Terminological similarities between the two formulations should not trick one into equating these two informal characterizations. In fact, if Turing framed the two bounds by imposing constraints on the possible actions of the computor, in

[10]Gurevich reported that Leonid Levin, in private conversation, told him that Kolmogorov viewed computations as physical processes. See Gurevich (2012, p. 6, 2015, p. 197). For my analysis, what matters is that Kolmogorov saw computations as objects that can be treated independently from any computer whatsoever, besides them being either abstract or physical entities.

Kolmogorov's axiomatization what is bounded and local is every step of the computational process, which is seen as the structural evolution from the input to the (possible) output. We can better appreciate Kolmogorov's originality by noting that no specific set of initial states or elementary operations is singled out by his definition of an algorithm. The restrictions imposed by these four intuitive considerations are structural limitations on the computational process. Specifically, all the steps of the algorithm must be bounded in their complexity and each of these steps must consist of an elementary transformation that depends only on a limited active part of the state under consideration. Both the boundedness and the locality constraint, then, are understood in a completely different way than in Turing's analysis. Thus, in stating his informal bounds, Kolmogorov focuses on the computational process and not, as Turing did, on the actions of the computor. From the point of view of the computational process, then, effectiveness is achieved by restricting our attention to those processes whose steps satisfy the bounded complexity and the limited active part requirements.

This focus on the computational process is connected with Kolmogorov's aforementioned programmatic aims of characterizing the general structure of an arbitrary algorithm. Kolmogorov and Uspenski stressed this point by stating that the Kolmogorov thesis is stronger than the Turing thesis, claiming that Kolmogorov did not want to just reduce computations to his definition, but also to capture their actual structure (Kolmogorov and Uspenski 1958, p. 74; Uspensky 1992, p. 396, 1993, p. 251). Coherently with this more abstract aim, Kolmogorov and his students repeatedly stressed how Kolmogorov's approach to computability is more general than previous ones.

I will then refer to the hidden process that led Kolmogorov to his 1953 philosophical scheme as *Kolmogorov analysis*. Despite not being explicit, we have seen that there is evidence in the writings of Kolmogorov and his own students that Kolmogorov analysis is philosophically different from Turing's one. Kolmogorov analysis wants to capture the structure of an arbitrary algorithm, stating the non-trivial restrictions characterizing classical computability on the general evolution of the computational process.

Note that here I stress a generality of Kolmogorov's approach to computability that, coherently with the Carnapian perspective of this paper, is purely conceptual and lies in its semi-formal axiomatization of an arbitrary algorithm. Kolmogorov's work on computability can also be said to be technically more general than Turing's, because Kolmogorov machines work on bounded graphs and are known to be able to implement more algorithms than classical Turing machines. This is a historically interesting fact, but should not be confused or merged with the conceptual generality stressed above. Also, Sieg and Byrnes have shown that Turing's approach to computability can be suitably generalized to bounded graphs, achieving with the so-called K-graph machines the same level of technical generality as Kolmogorov machines (Sieg and Byrnes 1996).

Kolmogorov achieved a semi-formal axiomatization of effective computability which is conceptually different from Turing's one. In the next subsection I will show how Dershowitz's & Gurevich's axiomatization of computability recovers the conceptual approach of Kolmogorov's work on computability.

3.2.2 Dershowitz's and Gurevich's axioms for computations and the notion of algorithmability

Dershowitz & Gurevich presented their characterization of effective calculability (Dershowitz and Gurevich 2008), adding a restriction to the initial states allowed by Gurevich's postulates for sequential algorithms (Gurevich 2000)[11]. Gurevich thinks about algorithms in terms of objects independent from any computer: "in our view, rather common in computer science, algorithms are not humans or devices; they are abstract entities" (Gurevich and et al. 2014, p. 38). What Gurevich wants to stress with this wording is not the (quite trivial) fact that algorithms should not be identified with their computing agents, something that every computability pioneer would obviously agree with. Gurevich, following Kolmogorov's footsteps, stresses instead that his explication of calculability treats algorithms and effective computations independently from their calculators. This philosophical standpoint justifies the technical motto of the ASM project of simulating algorithms at an arbitrary abstraction level, which can be traced back to Kolmogorov's and Uspenski's aforementioned programmatic aims.

According to Gurevich, then, it is possible to informally characterize any classical (i.e. sequential) algorithm by three non-trivial constraints on the evolution of the computational process, called the *Sequential Postulates*. For any sequential algorithm A:

> "POSTULATE I (Sequential time). An algorithm is a state transition system. Its transitions are partial functions." Dershowitz and Gurevich (2008, p. 313).

The first postulate tells us that a sequential algorithm, taking this wording in its vague acceptation, is a sequence of discrete computational steps. Gurevich excludes from the scope of his analysis continuous (analog) processes, transfinite computation sequences (involving limits), nondeterministic transitions, and nonprocedural input-output specifications.

> "POSTULATE II (Abstract state). States are structures, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute." (Dershowitz and Gurevich 2008, p. 317).

The second postulate tells us how algorithmic states are represented. It states that any algorithm, on his native level of abstraction, can be represented by (and actually, according to Gurevich, is) a (series of) first-order logical structure(s). There is obviously a certain degree of speculation in this thesis and Gurevich did not present any kind of justification for this proposition except for a classical argument by example: "logic experience shows that any kind of static mathematical situation can be adequately described as a first-order structure" (Gurevich 1999, p. 10). This representational tenet is shared also by Uspenski (1992, p. 395).

[11] Classically, a sequential algorithm (sometimes called sequential-time algorithm) is an algorithm that executes determined, isolated computations bounded step after bounded step.

Moreover, this postulate tells us that neither the vocabulary nor the base set of the algorithm states change during the computation. The evolution of the computational process is thereby given by suitable changes in the values of the functions of a given state[12]. The isomorphic closure is then imposed on the states and transitions of the algorithm for the same reasons adduced by Gandy and Sieg.

> "POSTULATE III (Bounded exploration). Transitions are determined by a fixed finite 'glossary' of 'critical' terms. That is, there exists some finite set of (variable-free) terms over the vocabulary of the states, such that states that agree on the values of these glossary terms, also agree on all next-step state changes." Dershowitz and Gurevich (2008, p. 319).

The third postulate expresses the effectiveness requirement, a fundamental limitation imposed on transitions. It can be seen as a generalization of Kolmogorov's bounded complexity and limited active part ideas. This restriction is pivotal to characterize sequential algorithms and, a fortiori, effectively calculable functions. Specifically, this postulate bounds the number of terms that have to be considered in order to make a transition from a given state to the next one. Unbounded searches and infinitary rules are therefore forbidden.

Philosophically speaking, this postulate is informally justified by what Gurevich calls the "Accessibility Principle", i.e. the assertion that, from the point of view of process-evolution, the only way in which a given algorithm A can access an element a of a given state X is to produce a ground term suitably evaluating that element. Then, the finite set T, which this postulate is about, is nothing but the set of all these terms.

The accessibility principle shows how Dershowitz's & Gurevich's postulates conceptually differ from Sieg's. This principle makes explicit a philosophical tenet of Kolmogorov analysis, namely the idea of explicating effective calculability independently from any calculator. If the central concept in Sieg's formal treatment is the notion of assembly (together with the related notions of causal neighborhood and determined regions), i.e. the formal representation of the actions that the computing agent can perform during the process of assembling a new state from a given one, in Dershowitz's & Gurevich's account the computing agent is completely out of the conceptual picture. Technically, the absence of a specific data-representation level makes it impossible to say something about how the computation process is carried out by the computing agent. Gurevich stresses this inability:

> "Imagine yourself being an executor of A at a state X. Since you cannot take advantage of a particular representation of the elements, the only way to access elements of X is to use the basic functions of X. *Essentially you use X as an oracle*."[13] (Gurevich 1999, p. 14, my emphasis).

[12] This encompasses also formulas usually expressed by relations. In fact, in Gurevich's formal framework relations are represented by functions and boolean values. Therefore first-order logical formulas are expressed in a quantifier-free way.

[13] Note that I have changed the symbols used by Gurevich to denote an algorithm and its state in order to make the quote coherent with the present terminology.

Gurevich presented these three postulates, proving a representation theorem between processes satisfying these axioms and a particular type of abstract transition system, called *abstract state machine* (ASM) (Gurevich 2000). ASMs are a particular version of static algebras, i.e. first-order structures without relations, including in their language three Boolean values as primitive (true, false, undef), as well as the usual Boolean operations. Basic transition rules are recursively constructed by two constructors working on basic update instructions. The problem of representing the addition of new atomic elements, something common in many sequential algorithms, is solved by using a Reserve universe, a naked set from which an appropriate constructor can take an element when it is needed. A sequence of rules is called a program. An ASM is then defined in the following way[14]:

> "An *abstract state machine* (ASM) is given by: a set (or proper class) S of algebraic states, closed under isomorphism, sharing a vocabulary F; a set (or proper class) $I \subseteq S$ of initial states, closed under isomorphism; a *program* P, consisting of finitely many *commands*, each taking the form of a *guarded assignment*: if p then $t := u$ for terms t and u over F and conjuction p of equalities and disequalities between terms." (Dershowitz and Gurevich 2008, p. 321, original emphases).

Gurevich proved a representation theorem, known as the "ASM Theorem" (Dershowitz and Gurevich 2008) or the "Main Theorem" (Gurevich 2000), ensuring that every algorithm that satisfies the three sequential postulates is step-by-step equivalent to a certain ASM program.

In order to adequately characterize computable functions, Dershowitz & Gurevich added another postulate to this characterization. In fact, one cannot be sure that a process satisfying the sequential postulates computes only calculable functions, because initial states are not restricted to computable ones and the other bounds work at a high level of abstraction. Recall that the computing agent is conceptually out of the picture and it technically acts as a kind of oracle. If, then, the initial state of a certain process is an oracle-like non-computable input, the resulting output of the computation can be outside the scope of Turing computable functions. Thus, Dershowitz & Gurevich need to impose, differently from Sieg's axiomatization, an explicit limitation on the set of possible initial states:

> "POSTULATE IV (Arithmetical State). Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values." Dershowitz and Gurevich (2008, p. 325).

The fourth postulate restricts the set of initial states only to arithmetical blank states[15], thereby restricting the domain of the algorithmic processes allowed only to

[14] For a full technical presentation of ASMs see Gurevich (1995) or Gurevich (2000).

[15] The authors definition of an arithmetical state is quite long. Roughly speaking, an arithmetical state is a combination of natural numbers, truth values (true, false and undefined), dynamic functions and arithmetical operations. Such an arithmetical state is considered blank when all operations, except the input, have an undefined value.

the numerical ones. A process satisfying both the sequential postulates and this fourth postulate is called an *arithmetical algorithm*. An ASM that satisfies the fourth postulate is called an *arithmetical ASM*. Using the concept of arithmetical ASM the authors proved that every arithmetical algorithm can be emulated by a certain arithmetical ASM and the co-extensiveness of arithmetical ASMs and partial recursiveness.

Then, Dershowitz & Gurevich recovered Kolmogorov's and Uspenski's forgotten conceptual approach to the problem of explicating effective calculability. They gave a new characterization of Kolmogorov's semi-formal axiomatization. Their postulates make explicit the philosophical elements behind Kolmogorov's philosophical scheme, such as the aim of capturing the general structure of an arbitrary algorithm, the emphasis on a more direct simulation of computations, the pivotal restrictions imposed on the evolution of the computing process. I will refer to the semi-formal notion of calculability implicitly defined by the Kolmogorov-Dershowitz-Gurevich postulates as *algorithmability*.

Let me reconstruct the conceptual structure of this group of explications of effective calculability in terms of Carnapian explication. The clarification of the explicandum was implicitly achieved by Kolmogorov in the process that led him to his 1953 philosophical scheme, i.e. Kolmogorov analysis. I called the conceptual offspring of his efforts *algorithmability*, i.e. what is defined by the Kolmogorov-Dershowitz-Gurevich semi-formal axioms for computations. From this notion, different explicata can be formulated, such as the classical Kolmogorov machines or the more recent ASMs. Here is a conceptual diagram of this second group of explications (Fig. 3):

I have thus presented Dershowitz's & Gurevich's foundational analysis of computability, together with Kolmogorov's seminal take, explaining how it can be understood as a Carnapian explication. In what follows I will conceptually compare this group of explications with the Turing-Gandy-Sieg ones.

4 Foundational analyses of computability as three-step explications

We have seen how Sieg's and Dershowitz's & Gurevich's axiomatizations build upon different approaches to computability and how some of the differences between their technical frameworks depend on their different conceptual background. Specifically, Sieg proposed axioms for calculators, building upon Turing's and Gandy's seminal works on computability. The conceptual core of his axioms for computors and mechanical devices is the degree of freedom that the calculator is allowed to have.

Fig. 3 The two-step structure of the Kolmogorov-Dershowitz-Gurevich explications of effective calculability

Dershowitz & Gurevich, following Kolmogorov's footsteps, instead proposed axioms for computations, considering them independently from any calculator whatsoever. The core of their axioms are the abstract state and the bounded work postulates, i.e. non-trivial representational and procedural limitations of the calculation process itself.

These approaches can be traced back, respectively, to Turing's praised analysis of calculability and Kolmogorov's seminal ideas about the intuitive notion of algorithm. In explication terms, it seems *prima facie* that Turing and Kolmogorov clarified the notion of effective calculability in two different ways, arriving respectively at the notions of computorability and algorithmability.

However, where exactly lies the conceptual difference between these two groups of explications? Several possibilities should be considered. These authors could for instance have explicated different intuitive concepts. Alternatively, there could be two possible disambiguations of effective calculability, as in the case of logical and statistical probability (Carnap 1950). Another possibility could be that these two groups of explications differ in the way in which they clarify their common explicandum or in the way in which they formulate their explicatum. In other words, where does the branching between these two groups of explications happen? Does it happen at the stage of the clarification of the explicandum or does it happen at the stage of the formulation of the explicatum? My proposal is that this difference lies neither in the first clarification step nor in the final step of the formulation of the explicatum, but rather in an additional step between the two.

The difference between these two approaches to computability cannot lie in the final step of the formulation of the explicatum. After all, there are many other instances of mechanical computability, extensionally equivalent to Turing computability, that do not show any such conceptual difference. Take Post's explication of effective calculability, for instance. Post's notion of normal system generability is indeed a different formalization of the intuitive concept of calculability than Turing's one. According to Post, a function is effectively calculable if and only if it(s symbolic representation) can be generated via the iterative application of certain substitution rules (Post 1943). This characterization of computability is then by no means conceptually reducible to Turing's one. However, Post's work can be completely subsumed under the conceptual approach of the Turing-Gandy-Sieg explications (cf. Sieg 2018). Even though Post does not specify a semi-formal notion of computability, his work seems to implicitly make use of the notion of computorability. In fact, what matters in Post's systems are the possible moves that one is allowed to make in order to solve the specific symbolic substitution puzzle, i.e. in order to calculate a certain function. In order to explicate the notion of effective calculability, Post then pays attention to the symbolic processes that can be carried out by an idealized human being working in a clerical way, just as Turing did[16]. Thus, the difference between Turing's and Kolmogorov's approaches must be sought elsewhere.

[16]In order to better appreciate this connection, see Turing's late take on solvable and unsolvable puzzles in Turing (1954).

Does perhaps the difference between these two groups of explications lie in the first step of these explications, i.e. in the clarification of the explicandum? Are these two foundational analyses two distinctive way of disambiguating or clarifying the intuitive notion of effective calculability? Even though it is certainly tricky to give arguments for the uniqueness of an intuitive, vague concept that different people tried to capture, some evidence can be provided. Despite their conceptual differences, Kolmogorov and Turing were trying to explicate the same informal notion: effective calculability. Compare the examples and the initial discussion contained in Kolmogorov and Uspenski (1958) with the ones in Turing (1936). Moreover, if one looks at the two foundational analyses of computability here treated, both Sieg and Dershowitz & Gurevich explicitly state that the aim of their work is to capture the intuitive notion of effective calculability, as traditionally understood and clarified (Sieg 2009; Dershowitz and Gurevich 2008). Turing and Kolmogorov explicated also the same contexts and uses of the term, i.e. the same clarified explicandum. Just like Turing, Kolmogorov was interested in explicating classical calculability in all its symbolic generality, abstracting away from the actual limitations of any computer or any domain. The difference between these two groups of explications is thus not explainable as a difference in the clarification of effective calculability such as the one at play in Church's work on general recursiveness (Quinon 2019, pp. 22-25).

Where then does the conceptual difference between Turing's and Kolmogorov's approaches to computability lie? It seems that the traditional, two-step version of Carnapian explication is not fine-grained enough to capture the differences between the semi-formal notions of computorability and algorithmability. In what follows I will present a refined three-step version of Carnapian explication that is able to capture the conceptual differences between these two groups of explications.

4.1 Explication as a three-step procedure: the semi-formal sharpening of the clarified explicandum

As recalled in Section 2, Carnapian explication is originally a two-steps procedure. I am now going to propose a refinement of the explication with the help of which I will achieve a better analysis of the two groups of explications of effective calculability treated above. This refinement adds another mid-level step, the *semi-formal sharpening of the clarified explicandum*, to the procedure, making it thus become a three-step method.

In the first step of Carnapian explication, i.e. the clarification of the explicandum, the explicator ought to clarify and (possibly) disambiguate the concept that she seeks to explicate, relative to a given context of use. My claim is that after this first, classically recognized step, in some cases the explicator sharpens this notion into a semi-formal one, a mid-level ExplicanDum**. Only after this sharpening, the explicator passes to the final third step of the procedure of explication, i.e. the formulation of the explicatum (Fig. 4).

The process of getting from ED^* to this semi-formal mid-level notion ED^{**}, i.e. the semi-formal sharpening of the clarified explicandum, is not a theoretically neutral step. Here the explicator has to make a theoretical choice amongst various possible directions in the sharpening. Every sharpening must in fact possess a given

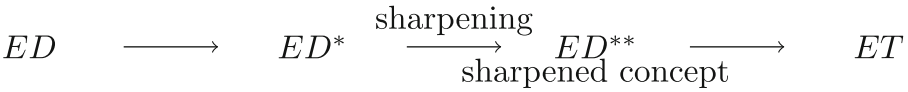

Fig. 4 The structure of the refined three-step version of explication

theoretical focus, which is given by the core aspects of the concept on which the sharpening focuses. The explicator highlights certain aspects of the concept, while she leaves other aspects at the borders of the conceptual sharpening.

Using a visual analogy, think about the action of shooting a picture of a landscape with a reflex camera. In order to take a good picture of the landscape, one has to focus the reflex camera on certain parts of the landscape, inevitably blurring the rest. It is impossible to focus on every element of the landscape at the same time. In order to take a good picture one has to choose what to focus on. Perhaps one wants to get a clear shot of the background of the landscape or perhaps one wants to put into focus the foreground. You cannot have both at the same time.

This is completely analogous to what happens in the process of semi-formally sharpening the clarified explicandum. The succession of shots of the same landscape in (Fig. 5) represents the possible evolution of a given explication. In (a) we can see a completely blurred landscape with unclear boundaries. This represents the starting

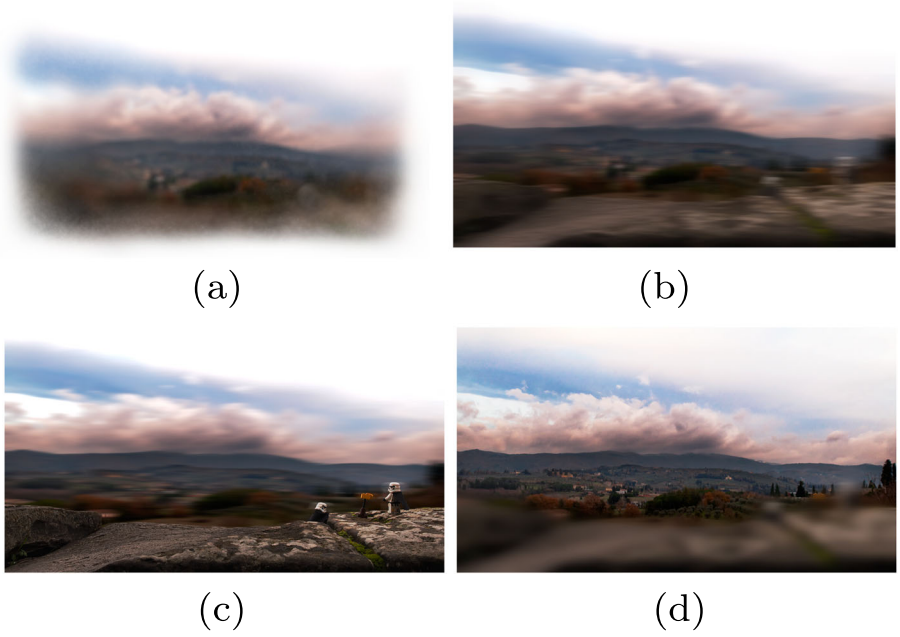

Fig. 5 Succession of shots of the same landscape with different focuses, representing the evolution of a given explicandum during the first two steps of the refined three-step version of explication

point of any explication, i.e. a given intuitive explicandum ED, the scope of which is undefined and that perhaps contains a great deal of ambiguity and vagueness. In (b) we can instead see the same blurred landscape, this time framed inside clear boundaries. This represents the clarified explicandum ED^*, which is obtained from the original explicandum through the clarification and the disambiguation of the uses and the contexts for which the notion is being explicated. Pictures (c) and (d) represent instead two different ways of putting the landscape into focus, respectively highlighting the foreground and the background of the picture, while inevitably blurring the rest of the image. The passage from (b) to one of these two pictures represents the semi-formal sharpening of the clarified explicandum. In this second step, the explicator sharpens the clarified explicandum ED^* onto a semi-formal sharpened ED^{**}. By stating semi-formal axioms or definitions, the explicator freely chooses the parts and features of the clarified explicandum that she wants to highlight. The parts and features to highlight and therefore the direction onto which the clarified explicandum is sharpened is never a philosophically neutral step. As represented by Pictures (c) and (d), multiple choices are always possible[17]. The semi-formal sharpening step thus changes neither the uses nor the context in which the explicandum is explicated, but it makes more precise the notion via a theoretically-laden semi-formal definition.

Finally, there is the last step of the explication, i.e. the formulation of the explicatum. I left this step out of the landscape-picture analogy because this step is different from all the others, due to its focus on the explicatum and not on (a disambiguated or clarified or sharpened version of) the explicandum. The explicatum can be a fully formalized notion or even an informal one in some cases (e.g. the famous fish-piscis example in Carnap 1950), but it is always a whole other concept, fully detached from the explicandum.

Let me stress that one should not expect the mid-level step of the sharpening of the clarified explicandum to occur in every explication whatsoever, but only in certain complex cases where many different formal explicata are meant to replace a single explicandum. Apart from the CTT case, other examples of concepts that may exhibit different sharpenings of the same clarified explicandum and for which the three-step explication seems an appropriate tool of analysis are formal theories of truth (Horsten 2011; Halbach 2014), different conceptions of set (Incurvati 2020), theories of informal proofs (Leitgeb 2009; Sjögren 2011), notions of logical consequence (Etchemendy 1990), and mathematical conceptions of infinity (Mancosu 2009).

In the next subsection, I will show how this refined three-step version of explication is able to account for the conceptual differences between the two foundational analyses of computability seen in Section 3. Specifically, the step of the semi-formal

[17] Some scholars outside the Carnapian tradition have also highlighted this possibility of sharpening concepts in multiple directions. See for instance (Smith 2011, pp. 27-29). Shapiro also stressed, in a Waismannian fashion, the possibility of this multiple sharpening in Shapiro (2013). However, it should be noted that despite some similarities, my concept of sharpening is a technical term that has to be understood inside the proposal of the three-step version of explication.

sharpening of the clarified explicandum will prove itself to be pivotal into adequately capturing the difference between the notions of computability and algorithmability.

4.2 Computorability vs Algorithmability

Applying my refined three-step version of Carnapian explication to the two aforementioned groups of explications of effective calculability gives us this conceptual picture (Fig. 6):

The branching between these two groups of explications happens neither in the first step of clarifying the explicandum nor in the final step of the formulation of the explicatum, but instead on the mid-level. The second-step, the semi-formal sharpening of the clarified explicandum, is where the two foundational analyses of computability differ. From the perspective of this refined version of explication, then, the notion of computorability and the notion of algorithmability are two semi-formal sharpenings of the clarified notion of effective calculability. Despite sharpening the same clarified notion, namely the classical concept of calculability clarified in terms of symbolic processes, they do that in two different ways, paradigmatically exemplified by Turing's praised analysis and Kolmogorov's hidden analysis of computability.

Let me state more precisely what these two semi-formal sharpenings of the clarified notion of effective calculability underlie:

- **Computorability**.
  The sharpening is achieved in terms of the possible actions of the computor. Bottom-up bounds are imposed on the freedom of the agent of the computation. The conceptual focus of the analysis is at the calculation level.
- **Algorithmability**.
  The sharpening is achieved in terms of non-trivial structural features of the process of computation. Top-down restrictions are imposed on the evolution of the computational objects. The conceptual focus of the analysis is at the abstract algorithmic level.

Then, the difference between the notion of computorability and the notion of algorithmability can be understood in terms of two different semi-formal sharpenings of the same clarified explicandum.

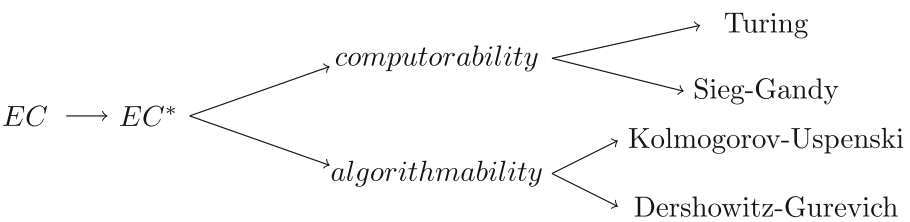

Fig. 6 The three-step structure of the Turing-Gandy-Sieg and the Kolmogorov-Dershowitz-Gurevich explications of effective calculability

Turing was the first one to sharpen the clarified notion of effective calculability, in what is usually called Turing analysis. Technically, I want to stress that this name is misleading. This is not a conceptual analysis, this is conceptual engineering of an intuitive notion into, first, a clarified notion and then into a sharpened one. Turing merged these two steps in his famous informal exposition of 1936. In his 'analysis', he fixed the use and the context of his explication of effective calculability, abstracting the notion of effective calculability from practical limitations, ruling out infinity and any kind of ingenuity, and focusing on the symbolic processes underneath any calculation process. This disambiguation and clarification of effective calculability belongs to the explication step of the clarification of the explicandum. At the same time, Turing sharpened this clarified explicandum by arguing that what is effectively calculable has to be computable by an abstract human calculator respecting in his actions the bounds that we are now all familiar with, thereby implicitly giving a semi-formal definition of the notion of computorability. This implicit semi-formal axiomatization of effective calculability in terms of actions of a computor belongs instead to the mid-level step of the sharpening of the clarified explicandum.

Turing's approach has been the prevailing way of semi-formally sharpening effective calculability since his 1936 analysis. After Turing, Gandy improved and made (some) assumptions underlying Turing's approach explicit, using them in order to attack the general problem of machine computability and thereby generalizing Turing's analysis of an abstract computor with his four 'principles for mechanisms'. Sieg continued this tradition with a detailed historical analysis of the conceptual background of the 1936 confluence and by axiomatically improving Turing's and Gandy's foundational analyses of computability.

The first occurrence of the Kolmogorov's approach to explicating effective calculability was instead Kolmogorov's own 1953 informal axiomatization of an algorithm. Kolmogorov, like Turing, implicitly clarified and sharpened effective calculability, merging these two steps into the hidden analysis behind his semi-formal definition of algorithmability. Together with Uspenski, he improved the 1953 treatment, arriving at the definition of a Kolmogorov machine, the first explicatum obtained via the sharpened notion of algorithmability. Despite the fruitfulness of Kolmogorov's model of computation (Uspensky 1992; Sieg and Byrnes 1996), the originality of his conceptual take has not been equally recognized. Gurevich and the ASM project revitalized Kolmogorov's approach to explicate effective calculability and the notion of algorithmability. As I stressed in Section 3, the ASM project approach to computability was mainly motivated by technical reasons, but nevertheless it recovered Kolmogorov's forgotten conceptual approach. A remarkable offspring of this project is Gurevich's axiomatic characterization of sequential algorithms, which makes evident the originality of the Kolmogorov-focus based view of computation. Together with Dershowitz, Gurevich then restricted his axioms in order to capture effective computations, thereby achieving a foundational analysis of computability deeply rooted in Kolmogorov's approach.

The refined three-step version of explication is also able to explain some debates about these two foundational analyses of computability. Recall that the semi-formal sharpening of the clarified explicandum always implies a theoretical choice. Two different semi-formal sharpenings of the same clarified explicandum are therefore

conceptually incompatible. This can be seen in some remarks made by Sieg and Gurevich on each other's foundational analysis.

Sieg doesn't recognize the originality of Dershowitz's & Gurevich's axioms, objecting to their claim of their approach being more general than the Gandy-Sieg one (Sieg 2013, pp. 119-121). We have seen that the generality stressed by Dershowitz & Gurevich is of a conceptual nature, being namely the fact that Kolmogorov's semi-formal sharpening of the clarified notion of calculability into algorithmability focuses on the abstract algorithmic level and not on the level of computation. This cannot be captured nor fully understood from the perspective of Turing's sharpening and thus within Sieg's framework[18].

This conceptual tenet of Turing's way of semi-formally sharpening effective calculability also has a formal correlate in the technical framework of Sieg's explicatum, in its focus on how a certain state of the computation is assembled, i.e. the input-output behavior on the specific level of data-representation at which the calculator works. Symmetrically, in Dershowitz's & Gurevich's treatment there is no place for any concept of calculator whatsoever. There is no trace of a computor in their formal framework or in their postulates, and Gurevich's accessibility principle explicitly forbids any meaningful notion of calculator. The technical correlate of this principle is the abstraction from any specific data-representation that makes the evolution of the state work via an oracle-like ground-term production. The assumption of Kolmogorov's way of semi-formally sharpening effective calculability explains their aforementioned claims of conceptual generality and also their critiques of Gandy's machines as a model of parallel computation. From the perspective of the notion of algorithmability, since the calculator is out of the conceptual picture, a Gandy machine seems then an unnatural layering of mechanical devices (Gurevich 2012, pp. 271-272). More generally, since both Gandy's and Gurevich's treatment of machine calculability heavily builds on their respective explications of effective calculability, conceptual differences between their formal models of machine computability can be explained by their opposite way of semi-formally sharpening effective calculability.

Another philosophical difference between these two groups of explications that can be explained by my refined three-step version of explication is the one stressed by Smith in the context of his idea of a squeezing argument for effective calculability (Smith 2013, pp. 357-364). He stressed how Kolmogorov imposed limitations on the concept of effective computation as top-down restrictions, in sharp contrast to Turing's bottom-up bounds on the actions of the computor. This difference is a consequence of the different semi-formal sharpening of the clarified explicandum achieved by these two pioneers of computability. Turing, in order to arrive at the notion of computorability, had to impose bottom-up bounds on the possible actions of the computor, having chosen to highlight the agent of the computation. Kolmogorov,

[18]It should be noted that Sieg also criticizes Dershowitz's & Gurevich's proposal on other, technical, grounds, such as the alleged technical generality of ASMs or the alleged proof of CTT that Dershowitz & Gurevich claim to have. Despite finding these objections very serious, I will, as already mentioned, leave out from the present work these kinds of discussions.

instead, highlighted the abstract structure of an arbitrary algorithm and its evolution process and thus, in order to define the notion of algorithmability, imposed the pivotal limitations as top-down restrictions.

5 Conclusion

Let me recall the main steps of the present work. I have conceptually analyzed two groups of explications of effective calculability: the Turing-Gandy-Sieg and the Kolmogorov-Dershowitz-Gurevich ones. I have argued that, from the perspective of Carnapian explication, their conceptual differences can be traced back to Turing's and Kolmogorov's different 'analyses' of effective calculability and specifically to the different semi-formal notions of computorability and algorithmability. We have then seen how the classical two-step version of Carnapian explication is not able to capture the difference between these two notions, tying it neither to the step of the clarification of the explicandum nor to the step of the formulation of the explicatum. I have thus proposed a refined three-step version of Carnapian explication which adds the mid-level step of the semi-formal sharpening of the clarified explicandum. With the help of this three-step version of explication, it is possible to adequately capture and conceptualize the difference between algorithmability and computorability in terms of two different semi-formal sharpenings of the same clarified explicandum. The three-step version of explication is also able to give a novel perspective on some philosophical debates related to these foundational analyses of computability.

The present work naturally presents itself as open to various possibilities of future work. First, the example of CTT shows the significance of the intermediate steps in the evolution of a given explication. Since, as I mentioned in Section 2, recent philosophical discussions over explication do not take intermediate steps of the procedure into much consideration, it would be interesting to see what the three-step version of explication can give to these debates. The new mid-level step could perhaps allow more fine-grained readings of some classical desiderata of explication such as similarity and might help understand better the differences between Carnapian explication and other philosophical methods. Secondly, it seems natural to apply the three-step version of explication to other examples of explications similar to CTT, such as formal theories of truth, logical consequence, informal provability, conceptions of set, and infinity. I expect to find at least in some of these cases conceptual differences analogous to the one between Turing's and Kolmogorov's approaches to computability. Finally, if in the future other different foundational analyses of computability will be proposed, it would be interesting to see whether they implicitly of explicitly define a semi-formal notion of computability different from both algorithmability and computorability and therefore another way of sharpening the clarified notion of effective calculability.

Acknowledgments I am indebted to Joe Dewhurst, Norbert Gratzl, Hannes Leitgeb, and two anonymous referees for helpful comments on previous drafts of this paper. I am also indebted to Filippo Gatteschi who took the beautiful pictures of our hometown used in Fig. 5.

Funding Open Access funding enabled and organized by Projekt DEAL. This publication was funded by the LMU Munich's Institutional Strategy LMUexcellent within the framework of the German Excellence Initiative.

References

Brun, G. (2016). Explication as a method of conceptual Re-Engineering. *Erkenntnis*, *81*(6), 1211–1241.

Brun, G. (2017). Conceptual Re-Engineering: From explication to reflective equilibrium. *Synthese*, *197*, 925–954.

Carnap, R. (1950). *Logical foundations of probability*. Chicago: The University of Chicago Press.

Carus, A.W. (2007). *Carnap and Twentieth-Century thought: Explication as enlightenment*. Cambridge: Cambridge University Press.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, *58*, 345–363.

De Benedetto, M. (2020). Explicating 'Explication' via Conceptual Spaces. Erkenntnis, https://doi.org/10.1007/s10670-020-00221-8.

Dershowitz, N., & Gurevich, Y. (2008). A natural axiomatization of computability and proof of Church's Thesis. *Bulletin of Symbolic Logic*, *14*, 299–350.

Dutilh Novaes, C., & Reck, E. (2017). Carnapian explication, formalisms as cognitive tools, and the paradox of adequate formalization. *Synthese*, *194*(1), 195–215.

Etchemendy. (1990). *The Concept of Logical Consequence*. Cambridge: Harvard University Press.

Floyd, J. (2012). Wittgenstein, Carnap, and Turing: Contrasting Notions of Analysis. In Wagner, P. (Ed.) *Carnap's Ideal of Explication and Naturalism* (pp. 34–46). UK: Palgrave Macmillan.

Gandy, R.O. (1980). Church's thesis and principles for mechanisms. In Barwise, J., Keisler, H.K., Kunen, K. (Eds.) *The Kleene Symposium* (pp. 123–145). North-Holland: Amsterdam.

Gödel, K. (1934). On undecidable propositions of formal mathematical systems. Mimeographed lecture notes by S. C. Kleene and J. B. Rosser, reprinted with revisions in Davis, M. (Ed.), The Undecidable: Basic papers on Undecidable Propositions, Unsolvable Problems And Computable Functions (pp. 39–74). Hewlett: Raven Press.

Gurevich, Y. (1991). Evolving algebras: an attempt to discover semantics. In Rozenberg, G., & Salomaa, A. (Eds.) *Current Trends in Theoretical Computer Science* (pp. 266–292): World Scientific.

Gurevich, Y. (1995). Evolving algebra 1993: Lipari guide. In Börger, E. (Ed.) *Specification and Validation Methods* (pp. 9–36): Oxford University Press.

Gurevich, Y. (1999). The sequential ASM thesis. *Bulletin of european association for theoretical computer science*, 1–32.

Gurevich, Y. (2000). Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, *1*, 77–111.

Gurevich, Y. (2012). Foundational Analyses of Computation. In Cooper, S.B., et al. (Eds.) *How the World Computes. Turing Centennial Conference* (pp. 264–275). Berlin: Springer.

Gurevich, Y., et al. (2014). What is an algorithm? (revised). In Olszewski, A. (Ed.) *Church's Thesis: Logic, Mind and Nature*: Copernicus Center Press.

Gurevich, Y. (2015). Semantics-to-Syntax Analyses of Algorithms. In Sommaruga, G., & Strahm, T. (Eds.) (pp. 187–206): Turing's revolution.

Gustafsson, M. (2014). Quine's Conception of Explication - and Why It Isn't Carnap's. In Harman, G., & Lepore, E. (Eds.) *A Companion to W.V.O. Quine* (pp. 508–525): Wiley.

Halbach, V. (2014). *Axiomatic theories of truth revised edition*. Cambridge: Cambridge University Press.

Hanna, J.F. (1967). An Explication of 'Explication'. *Philosophy of Science*, *35*, 28–44.
Horsten, L. (2011). *The Tarskian Turn. Deflationism and Axiomatic Truth*. MIT Press.
Incurvati, L. (2020). *Conceptions of set and the foundations of mathematics*. Cambridge: Cambridge University Press.
Justus, J. (2012). Carnap on concept determination: Methodology for philosophy of science. *European Journal for Philosophy of Science*, *2*, 161–179.
Kleene, S.C. (1936). General recursive functions of natural numbers. *Mathematische Annalen*, *112*(5), 727–742.
Kolmogorov, A.N. (1953). On the notion of algorithm. *Uspekhi Mat Nauk.*, *8*(4), 175–176.
Kolmogorov, A.N., & Uspenski, V.A. (1958). To the definition of an algorithm. *Uspehi Math. Nauk.*, *13*(4), 3–28.
Kuipers, T.A.F. (2007). Introduction. Explication in philosophy of science. In Kuipers, T.A.F. (Ed.) *handbook of the philosophy of science, General Philosophy of Science - Focal Issues* (pp. vii–xxiii): Elsevier.
Leitgeb, H. (2009). On Formal and Informal Provability. In Bueno, O., & Linnebo, Ø. (Eds.) *New Waves in Philosophy of Mathematics* (pp. 263–299). London: Palgrave Macmillan.
Leitgeb, H., & Carus, A. (2020). Rudolf Carnap Zalta, E.N. (Ed.)
Mancosu, P. (2009). Measuring the size of infinite collections of natural numbers: Was cantor's theory of infinite number inevitable?. *Review of Symbolic Logic*, *2*(4), 612–646.
Post, E.L. (1941). *Absolutely Unsolvable Problems and Relatively Undecidable Propositions: Account of an Anticipation. Unpublished draft. Reprinted in Davis, M. (Ed.), The Undecidable: Basic papers on Undecidable Propositions, Unsolvable Problems And Computable Functions*, (pp. 340–433). Hewlett: Raven Press.
Post, E.L. (1943). Formal reductions of the general combinatorial decision problem. *Amer. J. Math.*, *65*, 197–215.
Quinon, P. (2019). Can Church's thesis be viewed as a Carnapian explication?. Synthese, https://doi.org/10.1007/s11229-019-02286-7.
Reck, E. (2012). Carnapian Explication: a case study and critique. In Wagner, P. (Ed.) *Carnap's Ideal of Explication and Naturalism* (pp. 96–116): Palgrave Macmillan.
Shapiro, S. (2013). The open-texture of computability. In Copeland, J., Posy, C., Shagrir, O. (Eds.) *Computability: Gödel, Turing, Church, and Beyond* (pp. 153–181). Cambridge: The MIT Press.
Shepherd, J., & Justus, J. (2015). X-Phi And carnapian explication. *Erkenntnis*, *80*, 381–402.
Sieg, W., & Byrnes, J. (1996). K-graph Machines: generalizing Turing's machines and arguments. In Hajek, P. (Ed.) *Gödel '96, Lecture Notes in Logic*, (Vol. 6 pp. 98–119): Springer.
Sieg, W. (1997). Step by Recursive Step: Chuch's Analysis of Effective Calculability. *Bulletin of Symbolic Logic*, *3*(2), 154–180.
Sieg, W., & Byrnes, J. (1999). An Abstract Model for Parallel Computations: Gandy's Thesis. *The Monist*, *82*(1), 150–164.
Sieg, W. (2002a). Calculations by man and machine: Mathematical presentation. In *Proceedings of the Cracow International Congress of Logic, Methodology and Philosophy of science, Synthese Series* (pp. 245–260). Dordrecht: Kluwer Academic Publishers.
Sieg, W. (2002b). Calculations by man and machine: Conceptual analysis. In Sieg, W., Sommer, R., Talcott, C. (Eds.) *Reflections on the Foundations of Mathematics. Essays in honor of Solomon Feferman, Lecture Notes in Logic 115, Assoc. for Symbolic Logic* (pp. 390–409). Natick: A. K. Peters, Ltd.
Sieg, W. (2009). On Computability. In Irvine, A. (Ed.) *Handbook of the Philosophy of Mathematics* (pp. 535–630): Elsevier.
Sieg, W. (2013). Axioms for Computability: Do they allow a proof of Church's Thesis?. In Zenil, H. (Ed.) *A Computable Universe: Understanding and exploring nature as computation* (pp. 99–123). Singapore: World Scientific Publishing.
Sieg, W., Szabó, M., McLaughlin, D., Omodeo, E.G., Policriti, A. (2016). Why Post did [not] have Turing's Thesis. In *Martin Davis on Computability, Computational Logic, and Mathematical Foundations* (pp. 175–208): Springer.
Sieg, W. (2018). What is the Concept of Computation?. In Manea, F., Miller, R.G., Nowotka, D. (Eds.) *CiE 2018: Sailing Routes in the World of Computation*, (Vol. 10936 pp. 386–396): Lecture Notes in Computer Science.
Sjögren, J. (2011). A note on the relation between formal and informal proof. *Acta Analytica*, *25*, 447–458.
Smith, P. (2011). Squeezing arguments. *Analysis*, *71*(1), 22–30.

Smith, P. (2013). *An Introduction to gödel's Theorems*, 2nd edn. Cambridge: Cambridge University Press.
Stein, H. (1992). Was carnap entirely wrong after all?. *Synthese*, *93*, 275–295.
Strawson, P.F. (1963). Carnap's Views on Constructed Systems versus Natural Languages in Analytic Philosophy. In Schilpp, P.A. (Ed.) *The Philosophy of Rudolf Carnap* (pp. 503–518). Chicago: Open Court.
Tarski, A. (1933). Der Wahrheitsbegriff in den formalisierten Sprachen. English translation, (1956) The concept of truth in formalized languages. In Tarski, A. (Ed.) *Logic, Semantics: Metamathematics*. 2nd edn. (pp. 152–278). Oxford: Oxford University Press.
Turing, A.M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, ser. 2 42*(Parts 3 and 4), 230–265; Turing, A.M. (1937): A correction, *ibid. 43*, 544–546.
Turing, A.M. (1954). Solvable and unsolvable problems. *Science News*, *31*, 7–23.
Uspensky, A.V. (1992). Kolmogorov and mathematical logic. *The Journal of Symbolic Logic*, *57*(2), 385–412.
Uspensky, A.V., & Semyonov, A.L. (1993). Kolmogorov's Algorithms or Machines. In Shiryayev, A.N. (Ed.) *Selected Works of A. N. Kolmogorov, Volume III, Mathematics and Its Applications (Soviet Series)*, (Vol. 27 pp. 251–260): Springer.
Wagner, P. (Ed.) (2012). *Carnap's ideal of explication and naturalism*. UK: Palgrave Macmillan.