What is Turing's Comparison between Mechanism and Writing Worth?

In Cooper, S. Barry; Dawar, Anuj; Löwe, Benedikt (Eds.) *How the World Computes. Proceedings of Turing Centenary Conference and 8th Conference on Computability in Europe,* CiE 2012, Cambridge, UK, June 18-23, 2012.

Springer, Series: Lecture Notes in Computer Science, Vol. 7318 ; Subseries: Theoretical Computer Science and General Issues, 2012, XVIII, 756 p., ISBN 978-3-642-30869-7.

What is Turing's Comparison between Mechanism and Writing Worth?

Jean Lassègue[1] and Giuseppe Longo[2]

[1] CREA, CNRS - Ecole Polytechnique, Paris, France
<jean.lassegue, giuseppe.longo@polytechnique.edu>
[2] CIRPHLES, Ecole Normale Supérieure, Paris, France
<invited and corresponding author: longo@ens.fr>

Abstract. In one of the many and fundamental side-remarks made by Turing in his 1950 paper (The Imitation Game paper), an analogy is made between Mechanism and Writing. Turing is aware that his Machine is a writing/re-writing mechanism, but he doesn't go deeper into the comparison. Striding along the history of writing, we will hint here at the nature and the role of alphabetic writing in the invention of Turing's (and today's) notion of computability. We will stress that computing is a matter of alphabetic sequence checking and replacement, far away from the physical world, yet related to it once the role of physical measurement is taken into account. Turing Morphogenesis paper, 1952, provides the guidelines for the modern analysis of "continuous dynamics" at the core Turing's late and innovative approach to bio-physical processes[3].

Keywords: Consonantal alphabet, phonograms, alphabet, combinators, λ-calculus, non-linear dynamics.

1 Introduction

In his 1950 philosophical article, Turing rather offhandedly used a comparison between Mechanism and Writing he didn't take time to develop: "(Mechanism and writing are from our point of view almost synonymous)", [[28]: 456]. The relationship between the two terms is far from being straightforward though. Mechanism seems to be understood by Turing as a branch of physics and, in a broader perspective, as the ideal deterministic world-view natural sciences should pursue. On the other hand, even if writing could be stretched to fit into linguistics it is neither a type of knowledge nor a scientific paradigm but rather a technology used for recording data in many different areas. Therefore how could mechanism and writing be compared? Turing's remark could of course be considered as a simple digression in the course of his article and the sentence just quoted is indeed put into brackets. But it is worth trying to stick to this comparison and follow it as far as it can lead us to. This is the goal of the next few pages. First of all, we should refine the comparison between the two terms to make it clearer.

2 Mechanism as a Scientific Paradigm

Resting upon the intellectual development of Turing himself, we know firstly that he extended Hilbert's formalist program as far as to come up against one of its inner limitations: the negative

[3] Longo's invited lecture, The Turing Centenary Conference (CiE 2012), *Computational Models After Turing: The Church-Turing Thesis and Beyond*, Isaac Newton Institute programme, Cambridge, June 18 - 23, 2012.

result of the halting problem in 1936 is a paradigmatic example of a limitation in decidability as well as the birth certificate of computer science. Now, as hinted below, there is a deep and non-trival relationship between (computational) undecidability and (deterministic) unpredictability in mathematical physics. Secondly, Turing managed to work out a new kind of non-predictive determinism in his 1952 article by studying symmetry breaking and non-linear processes in morphogenesis. In this article, Turing puts forward the notion of an "exponential drift" - what is nowadays called "sensitivity to initial conditions". Because of this "drif", a fluctuation / perturbation below measurement, may be amplified over time and may induce an *unpredictable*, yet major (i.e. *observable*) change in the genesis of forms. As is well known since Poincaré, but not much studied since then (with a few exceptions: Hadamard, Pontyagrin, Birkhoff...), unpredictability pops out as soon as non-linearity expresses dynamical interactions between components of a system, such as in the action/reaction/diffusion system Turing studies in 1952 (of course, technically, he works at the solutions given by the linear approximation of the system, yet he is aware of, and greatly interested by, the key properties of non-linearity). Of course, unpredictability is at the *interface* between the mathematical determination, e.g. equations, evolution functions..., and the physical processes which is being modeled by these written equations, functions etc..
In reference to this unpredictability, Turing makes the following observation in his 1950 paper, concerning his Discrete State Machine (DSM, as he then calls his Logical Computing Machine), ...[in a DSM], it is always possible to predict all future states... This is reminiscent of Laplace's view... The prediction which we are considering is, however, rather nearer to practicability than that considered by Laplace" [[28]: 440].
In fact, he explains, the Universe and its processes are subject to the exponential drift and gives the following example: "The displacement of a single electron by a billionth of a centimetre at one moment might make the difference between a man being killed by an avalanche a year later, or escaping.". On the contrary, and here lies the greatest effectiveness of his approach, "...It is an essential property of... [DSMs] that this phenomenon does not occur. Even when we consider the actual physical machines instead of the idealized machines...", prediction is possible, [[28]: 440]. Of course, stresses Turing, there may be a program which is so long that it is hard (practically impossible) to predict its behavior; yet, this is a practical issue, a very different one from the core theoretical property, that is deterministic unpredictability in non-linear dynamical systems, due in particular to the exponential drift.
In view of its developments, which originated from Turing's DSM, computer science appears to be predictive determinism made real, and no one should be allowed to belittle what an incredible feat it is to have made it possible since many (and perhaps most) physical processes precisely do not belong to predictive determinism but to the non-predictive one Turing explored in his last years. From this point of view, computer science came not so much as a surprise but as a kind of miracle, even if it became more and more clear that mechanism cannot be extended to most physical processes in nature (the "frictionless" simple pendulum and pulsars are some of the known exceptions), as was established as early as Poincaré's famous example in celestial mechanics concerning The Three Body Problem: the sun and one planet give a predictable Keplerian orbit, one more planet and the non-linear interactions let the system go, at equilibrium, along unstable orbits (almost always). Therefore, what should be kept in mind in Turing's comparison is not "mechanism" as such but the opposition between two paradigms in science: predictive (mechanistic) determinism on the one hand and non-predictive determinism on the other, the two of which Turing explored with close scrutiny and in which he produced fundamental results.
In the case of writing, a similar remark should be done. It is not writing proper that the attention should be focused on, but the opposition between explicit and implicit processes in the recording of data. It will therefore be argued below that computer science inherits a very specific writing structure which gives a compelling rationale to the comparison put forward by Turing: *explicit*

processes in mechanism and writing are of the same nature. But in order to reach this conclusion, we have to hark back to the history of writing.

3 The Long History of Writing

3.1 Writing Numbers and Languages

Turing does not specify which kind of writing should be compared to mechanism and we are left only with conjectures. Let us therefore start with the most primitive written system in the West as described in [24]. If we go back to this primitive use of writing, we find that it was first used in Mesopotamia around -3100 B.C. to *count* specific goods to be stored and that the recording of natural languages came only in second, at a later stage. The counting system itself which was utilized was much older.

Prehistoric Counting System In order to record quantities of goods which were presumably stocked for conservation and redistribution, Mesopotamian accountants made use of a prehistorical system of tokens that can be traced as far back as -7500 B.C. This system of tokens had three remarkable features. Firstly, tokens were geometrical forms, that bore no resemblance to the good which they referred to. Secondly, these tokens were used to refer to specific goods: each kind of geometrical token would refer to a single kind of good. Therefore, the system implied a great variety of tokens, in fact as many as the kinds of goods to be accounted for. Thirdly, tally was made possible through sheer repetition: using as many tokens as the number of pieces would indicate the quantity of the good in stock. By its geometrical form, each token was bestowed a qualitative meaning referring to the type of good and by its repetition, a quantitative one, referring to the number of pieces under consideration. It should be emphasized that this system is purely graphical and does not presuppose any use of natural language, even if it was presumably accompanied by verbal expression.

Involvement of Language Later on, the tally system evolved and verbal language became necessary to keep track of the name of the donators. The written recording of words in natural language capable of referring to names would follow the numerical system which was purely graphical. But the transformation of the writing system had a consequence on the writing of numbers in return. It was not thanks to repetition that the tokens were now capable of referring to numbers: it was through their *names* in natural language that they could play the role of predicates[1]. This meant a fantastic economy of means since from then on, no repetition of tokens was needed to express plurality and a single token interpreted as a quantity could now avoid the tedious and often mistaken repetition of the same token. But in return, it would also induce the use of natural language as the foundation for the whole counting system: writing was not entirely graphical anymore and would then oscillate between a visible, graphical part and an invisible, verbal one (for example the graphical token "ÅA" would *mean* five units), the relationship between the two being subject to technical innovations all along the history of writing.

3.2 Three Technical Innovations in the History of Verbal Writing

Three of these innovations are worth mentioning in order to specify in what sense mechanism and writing can be considered as synonyms.

[1] [[25]: 162-167]: "Remarkably, no new signs were created to express abstract numbers. Instead, the signs referring to measures of grain simply acquired another meaning: the wedge standing for a small unit of grain became "1", and the circle representing a large unit of grain became "10"."

Phonograms As we just noticed, as soon as it became necessary to go beyond graphical subitizing in order to keep a numerical track of what was in stock, *speech* was used in the up to then seemingly silent and only graphical process of recording. Phonograms (addition of syllables to form a new word like in a rebus) were devised and the way tokens standing for numbers and those standing for goods was therefore reorganized.
A phonogram uses the *first* syllable of a word chosen in advance (let us call is a "master-word") as a part of its own composition [2]. In order to compose a new phonogram from "master-words", the *first* syllable must therefore be recognized as such and be used at some place (the first or any other one fixed in advance) in the new word. This process follows therefore a numerical (ordinal) pattern in which the places (first, second and so on) taken by the syllables are crucial. The disassociation of a syllable (recognized as the first one in the master-word used as blueprint) plays the same role as a numerical predicate which is first dissociated and then placed next to a word to compose a sentence (like in the expression '3 jars'). This writing system implies that one has to know in advance which are the "master-words" new phonograms can be composed from: any phonographic composition presupposes that the speaker has a knowledge of the set of syllables used in the "master-words" of the language.

Alphabets Contrary to phonograms the construction of which depends on "master-words" used as blueprints for syllables, alphabets seem at first sight to form a finite list of tokens that record the sounds used for the formation of all the words of a given lexicon. Graphically speaking though, two different types of alphabets must be distinguished for they do not record the same phonological realities: a consonantal alphabet or "abjad[3]" tends to record the *syllables* of a given language and therefore carries the semantic value attached to its syllables whereas a vocalic-consonantal alphabet or "alphabet[4]" in the strict sense of the term tends to record the *phonemes* of a given language, apart from any semantic consideration. Let us make this difference clear.

Consonantal Alphabet A consonantal alphabet or "abjad", is a type of writing system where each symbol stands for a consonant, leaving the reader to supply the appropriate vowel and complete the syllable. It is particularly fit for languages (like Hebrew or Arabic) that have only a few vowels, for the semantic indetermination regarding the completion of the syllable is rather limited and easily supplemented by a reader, provided that he or she is knowledgeable in the language which is spoken. In this type of writing, the knowledge of the language is therefore a mandatory prerequisite, for no word can be read if the reader is not able to supply the vowels.

Vocalic-consonantal Alphabet Alphabets in the strict sense of the word are systems of writing that do not record syllables but phonemes, i.e. sounds that are considered as having the same *function* in speech but do not have any meaning in themselves. From this point of view, abjads and alphabets, even though they are sometimes both called alphabets, are not based on the same principle (see [10]) and they differ on at least three features. Firstly, alphabets are not restricted to languages the structure of which possesses only a few vowels: any language can be written with an alphabet because what is recorded is just phonemes, that is phonological realities and not syllables. Secondly, no previous knowledge of the language is required: anybody who can read alphabetically can read a foreign language which is alphabetically written, even without

[2] For example, [[25]: 162-167] uses as an example the composition of the modern phonogram 'Lucas' by the composition of the token for 'Lu' (retrieved from the first syllable of the Sumerian word for mouth "Lu") and of the token for 'Cas' (retrieved from the first syllable of the Sumerian word for man "Ka").
[3] Which stands for the first four letters in Arabic writing: 'Alif, Bā', Ǧīm, Dāl.
[4] Which stands for the first two letters in Greek writing: Alpha, Bêta.

understanding a word of what is written (try with Turk or Vietnamese if you are not familiar with these languages or just read something when you are tired until you eventually realize you have been reading without understanding a word of what was written... This is just impossible in Arabic or Hebrew). Thirdly, and as a consequence, no intervention is required from the reader, which means that *the process of alphabetical reading can be transferred to a machine*. This is the true reason for the comparison Turing put forward in his 1950 article: mechanism and alphabetic writing only involve entirely explicit processes of pattern recognition of characters that have an objective value in the sense that they do not depend either on the type of language (natural or artificial language as long as it is written) or on the type of reader (human or machine).
The oldest example of an alphabet is very well known: it is the Greek one, which appeared in the 8th century B.C. and from which all other alphabets derive. Its linguistic originality has been commented upon through and through but what is most striking from our point of view is just one fact: *alphabetical reading is potentially mechanisable*. That is why the Hilbertian formalist program as well as the works by Schönfinkel and Curry (see below) and all formalisms in the 30's *inherit a Greek alphabetical-mechanistic structure* which is the core of computability, after a long history we cannot fully describe in details here. This alphabetical-mechanistic structure is not, so to say, the alpha and omega of deterministic science, as Turing was well aware in his 1952 article, both in mathematical physics and in linguistics. For it is true that explicit processes deprived of any semantic structure lack what is most fundamental in spoken languages as well as in mathematical (and biological) structures: their plasticity, their versatility and, nonetheless, their incredible stability. But one has to make clear what this deterministic structure exactly is to be able to go beyond.

4 The Alphabetic Combinators

In the 1920's, Shönfinkel and Curry independently proposed an algebra of signs for expressing and computing logical predicates (see [8,9]) for a technical introduction and an historical account). Curry's formalism, which is still a reference, is based on two "combinators", K and S, that *act on (combine) signs* according to the following rules:

$$(KX)Y > X \qquad \text{(read: } KXY \text{ reduces to } X \text{)} \tag{1}$$

$$SXYZ > XZ(YZ) \qquad \text{(read: } SXYZ \text{ reduces to } XZ(YZ) \text{)} \tag{2}$$

By convention, association is intended to the left: $XYZ \equiv (XY)Z$ (the first sequence is *identical* to the second).
Surprisingly enough, by combining these "maipulators of signs" in a type-free way (there are no restrictions on what can be applied to what), one can compute a large class of functions from sequences of signs to sequences of signs. For example, $I \equiv SKK$ computes the identity function: $IX \equiv SKKX > KX(KX) > X$, while $SIIX > XX$, or $(S(K(SI))(S(KK)I))XY > YX$,
where, of course, X, Y, $Z \ldots$ are arbitrary signs or *sequences of signs* (within a parenthesis!). Truth values may also be encoded: $T \equiv K$, $F \equiv KI$. And, by a smart coding of numbers as combinators (take $0 \equiv KI$ and... keep going in a non-obvious way, see the references or below), one can compute all the Turing computable functions. Let's stress that Curry's Combinatory Logic, at the origin of the formal computability of the '30s, is a pure *calculus of alphabetic signs*. Uninterpreted, or meaningless (and this is crucial), sequences of alphabetic signs are formally copied, erased, iterated in a ... *potentially mechanisable way*, following Hilbert's request for formal systems. There is even no intended mathematical meaning, as there was no "interpretation" of this calculus on algebraic or geometric structures at the time. The point of course is to understand what "interpretation" means. Yet, this calculus is where the conception of the modern computing

machine began: by a "meaningless" action of signs over signs, formally codified by axioms and/or rules (the rules for K and S above). All what is needed is a "pattern matching of signs" (one should better say: sequence matching, i. e. checking that two sequences are identical) and *sequence replacement* (replace a sequence by another, according to the rules).
Shortly later, Church, also in Princeton, invented the λ-calculus [6]. This calculus has a more "mathematical flavor", since it provides a purely formal theory of functions, based on "functional abstraction" over variables ($\lambda y.X$ below is the functional analogous of $\{y/X\}$ in Set Theory). Consider a string of signs, X, possibly containing a (free) variable y, then write $\lambda y.X$ as the function that operationally behaves as follows, by "reduction" again, ">":

$$(\lambda y.X)Z > [Z/y]X \text{ where } [Z/y] \text{ is short for } Z \text{ replacing } y \text{ in all free occurrences of } y \text{ in } X \quad (3)$$

(a variable y is bound or "not free" when it occurs in the field of a $\lambda y.(...)$, see [3] for details). No more needs to be said: by borrowing for functions the notion of "abstraction" used for sets, one obtains an amazingly powerful calculus with just one axiom (and a few obvious rules for reduction, ">"). Integer numbers, for example, are easily coded by $0 \equiv \lambda x.\lambda y.y$ and $n \equiv \lambda x.\lambda y.x(x...(xy)...)$, where x occurs n times. Curry's and Church's alphabetic calculi are equivalent modulo a simple translation (yet, some subtleties are required, see [12,3]).
When these calculi and Turing's computable functions, jointly with Kleene's system [13], were proved equivalent as for their number-theoretic expressiveness (they compute the same class of functions), λ-calculus played a pivotal role: the proofs were given under the form of "equivalent λexpressiveness". The Turing-Church Thesis was then justified by these rather surprising equivalences: it was fair to assume that all formal systems a la Hilbert could at most define the same class of functions, the Turing computable ones. The meaningless manipulation of alphabetic strings is at the core of these formalisms, following Hilbert's notion of a formal system. Type Theory [7,11] made the link between *propositions* of Formal Logic and *types* that one can associate to typable terms of Curry-Church calculi.
Are these purely formal-alphabetic systems "in the (physical) world"? They are not. As we said earlier on, they are grounded on one of the most fantastic (and earliest) abstraction invented by humans: alphabetic writing. They stress the core idea of it: signs have no and must not have meaning. In the alphabetic systems of writing, meaning pops out phonetically: the sound of voice gives meaning to the sequences of letters.
However, let's make a distinction. In the formal systems for computation, a form of meaning is possible: the *operational* one given by the rules and axioms for manipulating them. Consider, say, the $KXY > X$ rule mentioned above. One may say that it gives a "meaning" to K as an *operator* on signs. No meaning yet, in no ordinary, nor mathematical sense, just an operational definition. The non obvious mathematical structures "interpreting" these calculi were invented in the late '60, [26] and were later given a general set-theoretic frame, see ([18,3]) for the type free calculus, and a categorical one, for both the type-free and the typed calculi [27,19,14,1].
The difference of these forms of "meaning" should be clear. The first is the purely mechanical processing of uninterpreted signs: (re-)writing as a mechanism. As explained above, it is the direct result of the amazing abstraction proposed by the greek invention of an uninterpreted alphabet, at the origin of our modern form of writing. These signs are so meaning independent that they may be used for a game of signs, a combinatorial mechanics which is at the true origin of modern computing: a machine can operate according to the rules for the K and S operators. "Mechanism and writing are from our point of view almost synonymous", says Turing. The so-called operational semantics is just the specification of rules to operate mechanically on signs. Indeed, once Turing made explicit the distinction between software and hardware, a vast area of programming styles were directly based on Lambda Calculus and Combinatory Logic: the functional languages (LISP, Edinburg ML...), the Haskell style's languages (from Haskell B. Curry), which are also

functional, but with a more combinatorial flavor. Theory of Programming is largely indebted to "term-rewriting", [5], the discipline that generalized Curry's and Church's calculi by the general analysis of alphabetic "sequence checking and sequence replacement systems of mechanical rules", at the core of Turing's intuition.
The mathematical semantics instead, deals with meaning in a rather different way. One has to interpret those signs and their operations over independently established mathematical structures, deriving their "meaning", in principle, from radically different conceptual constructions. It is like a "translation" into a different language that *per se* bears meaning by different or totally independent conceptual experiences and contexts. The challenge is not obvious. Let's see why. Any sequence of signs can be applied to any other. So XX is well formed, say. A way to understand that X may operate on X is to consider X as a function acting on ... X. Typically, in λ-calculus, $(\lambda x.M)N$ works even when $N \equiv (\lambda x.M)$, whose *mathematical meaning* is that of a function applied to itself. How to construct a non-trivial mathematical universe where this is possible? The universe must be non-trivial as it must contain all integer numbers and, as we know from Turing's equivalences, all computable functions on them. Some non obvious categorical properties allow this, in particular the existence of "reflexive objects", i. e. spaces D such that all the morphisms (functions) on D, call it $D \to D$, may be isomorphically embedded into D. This is set-theoretically impossible, unless $D = \{1\}$, since $D \to D$ is always provably larger than D. The construction of suitable categories is given in the references: they must have enough morphisms to interpret all computable functions on integers, but must not contain all set-theoretic functions. Here, we just want to stress that that categorical interpretation may be really considered as providing "mathematical meaning": one has to construct, by totally independent tools from the operatorial signs, a mathematical structure (a category, in its precise technical sense), where, typically, this challenging self-application is understood by an embedding of a space of morphisms (functions), $D \to D$, into its own domain of definition, D.
But why should these categories "add meaning"? First, as we said, a translation in another language does add meaning, per se. Second, categories are defined in terms of diagrams, that is of visual objects. Categorical diagrams may be always described by sets of equations, yet only their visual display makes the symmetries manifest, while they are just implicit in the equations. Dualities (like adjunctions, a deep concept in Category Theory) are symmetries and play a major role in the mathematics of categories. So, in the end, meaning is added to the mechanical signs' pushing, by their interpretation into geometric structures, including diagrams and their gestaltic power to make us understand. Symmetries are meaningful to us: they organize space, correlate structures and visualized concepts.
Of course, by adding structural meaning we lose effectiveness. In Combinatory logic and λ-calculus, the reduction operation, ">", is decidable and its transitive and symmetric closure, "=", is semi-decidable. That is, a machine can perform them and, given two terms, M and N, it may decide whether $M > N$ or semi-decide whether $M = N$. Instead, in all proper set-theoretic and categorical models, the interpretation of ">" and "=" are identified and are not semi-decidable. No machine can (semi-)decide the equality of infinite objects, like the functions or morphisms in the categories needed for the semantics of these type-free calculi. In a sense, one goes from the Laplacian universe where dynamics and computations are predictable (one may "decide" the future), as observed by Turing, to richer universes where meaning is grounded on manifolded conceptual experiences (mathematical structures, gestalts...), far away from the purely mechanical notion of "decidability" by an algorithm.

5 The Undecidable

There exist deep and non-obvious correlations between Gödel-Turing undecidability and Poincaré's deterministic impredictability, at the core of modern non-linear dynamics, see [21]. One thing should be clear though: our systems of equations generally yield computable solutions, if any. Writing and solving the most complex systems of (non-)linear equations is a matter of writing and rewriting, that is one has to write them and apply an *algorithm* that we invented to solve them, if any. This is entirely effective, including the result (see below for some exceptions), unless one is crazy enough as to put a non-computable real, Chaitin's Ω, say, as a coefficient or an exponent in the equations. Some computationalists derive from this that "the world is computable" (against Turing's claim: "The displacement of a single electron by a billionth of a centimetre... " may induce major unpredictable/incomputable changes, ([28]: 440). As he saw perfectly well, it is the question of physical measurement that changes everything, i.e. the relation between mathematics and the physical world by way of a measurement (the "small error... about the size", ([28]: 451). It is our alphabetic writing of equations and solutions that is computable. And most of the time mathematics "misses the world": impredictability means that the solution of the intended equational/functional system for a physical process diverges from that process, i. e. very soon it does not describe it anymore, or "prediction becomes impossible", as observed since Poincaré. The key issue is "how to associate a number to a process", whether it is an integer or a computable real, that is, the problem is *measurement*, the only way by which we access to the physical processes.

Take a physical double pendulum: the two equations determining it, if implemented on a digital computer (they have computable solutions, of course), do not allow to predict this chaotic physical process, in view of the inevitable choice of an input number *and* of (Turing's) "exponential drift": two (computable real) numbers within the best measurement interval *may very soon yield radically different trajectories.* In [16], unpredictability is proved for the planetary system (many more equations), at a reasonable astronomical time scale (about 1 million years). There is no way to compute and predict with arbitrary precision processes that are enough sensitive to initial conditions. The approximation in (classical!) measurement has lower bounds of various sorts: thermal fluctuation, gravitational effects, elasticity... Thus most *physical* processes, far from "computing" non-computable *functions*, just do not *(well) define* input-output functions, [20,21]: starting on the same input as an interval provided by measurement, a physical process the *mathematical* representation of which is chaotic, will soon lead to diverging trajectories, thus to very different measurable outputs. As for actual implementations, the Shadowing Lemmas, [22], at most guarantee (and only for hyperbolic non-linear systems), that, *for any* discrete space-time trajectory, *there exists* a continuous one approximating it (and not the converse!). So far, we dealt with the unpredictability of *physical* processes modeled by *computable functions.*

There exist two remarkable exceptions to the computability of the mathematical solutions of interesting mathematical systems. One is given in [4], the other in [23]. These effectively written systems yield *only* non-computable solutions. Yet, by looking closely at their proofs, one sees that this doesn't say much, per se, about the physical world, but only about the "mechanisms of writing" used.

The first proof reduces, by a complex argument, to Turing's halting problem, thus to a purely diagonal argument within the formalism. As for the second, the issue is more delicate and it has been extensively discussed, see [31]. The very elegant system used for computability over real numbers, [30], (there are many and they are not equivalent, a challenge for the extension of Church Thesis to computable "continua"), uses as input values the computable reals. One then shows that *computability implies continuity* (over the Cantor-set topology). Then the proof goes by proving the non-continuity of the unique solution, under computable initial conditions.

This non-continuity is clearly a relevant physico-mathematical fact, to be discussed first. But... what is exactly its *physical* sense? Poincaré, from a purely mathematical analysis to the non-analyticity (almost everywhere) of the solution of the Three (planetary) Body Problem, gave a physico-mathematical meaning to the existence of diverging coefficients in the approximating series: minor fluctuations in the initial conditions (below measurement) could mathematically yield divergent trajectories. Predicting is a matter of "pre-dicere" (to say or, better, to write in advance), that is to *measure* firstly, then *compute* by (re-)writing and compare written numerical results and measurement obtained at the end of the process.
We lack, so far, a close analysis of the physical relevance of the incomputability (due to a discontinuity) of the wave equations (by the way, are these as sound, for actual waves, as Newton-Laplace equations for planets?). This should be discussed in reference to actual physical measurement as the only way we have to "extract" numbers from a natural process. We can only say, so far, that the computations lead to a "mathematical singularity" (a discontinuity), an issue *within* the mechanism of term (re-)writing and solving equations.
And so we are back to our founding father. Alphabetic writing and its formal re-writing are effective (they are "synonymous to mechanism"): incomputability is their internal challenge. Each time, the relations between mathematical writing and the world must be closely analyzed, as Poincaré did in reference to physical measurement: there is no way we can deduce the existence of physical super-computers from internal (yet very deep) mathematical games of signs. The fact that the world does not compute does not need to mean that it "super-computes", i.e. it computes non-computable functions, as some wrongly claim. As we said, it may simply mean that it doesn't even (well) define a function. This is so for a double pendulum, the planets or any system to be mathematically described as sensitive to initial conditions. It may also be the case that our mathematical system badly models the world; typically, Navier-Stokes and wave equations badly apply close to borders - what is the relevance of this fact, as for measurement? More generally, what is the relationship between (effective) mathematical writing and physical processes which may be only related by means of physical measurement?

References

1. Asperti A., Longo G.: Categories, Types and Structures; Category Theory for the working computer scientist. M.I.T. Press, 1991 (out of print, downloadable from: http://www.di.ens.fr/users/longo/)
2. Bailly G, Longo G.: Mathematics and Natural Sciences: the Physical Singularity of Life. Imperial Coll. Press, London (2011)
3. Barendregt H. The Lambda Calculus: its syntax and semantics. North Holland, Amsterdam (1984)
4. Braverman M., Yampolski M.: Non-computable Julia sets. Journ. Amer. Math. Soc. 19, 551-578 (2006)
5. Bezem M., Klop J.W., Roel de Vrijer R.: Term Rewriting Systems. Cambridge Univ. Press, Cambridge (2003)
6. Church A.: A set of postulates for the foundation of Logic, Annals of Math., (2) 33, pp. 346 - 366 and 34, pp. 37-54 (1932/33)
7. Church A.: A formalisation of the simple theory of types. JSL. 5, 56-58 (1940)
8. Curry H. B., Feys E.: Combinatory Logic vol. I. North-Holland, Amsterdam (1968)
9. Curry H. B., Hindley J. R., Seldin J.: Combinatory Logic vol. II. North-Holland, Amsterdam (1972)
10. Herrenschmidt C., Les trois écritures; langue, nombre, code. Gallimard, Paris (2007)
11. Hindley R.: Basic Simple Type Theory. Cambridge University Press, Cambridge (1997)
12. Hindley R., Longo G.: Lambda-calculus models and extensionality. Zeit. für Mathemathische Logik und Grundlagen der Mathematik, 26(2), 289–310 (1980)
13. Kleene S. C. Lambda definability and recursiveness. Duke Math. J. 2, 340-353, (1936)
14. Lambek J., Scott P.J.: Introduction to higher order Categorial Logic. Cambridge Univ. Press, Cambridge (1986)
15. Lassègue's page (downloadable articles), http://www.lassegue.net

16. Laskar J., "Large scale chaos in the Solar System", Astron. Astrophysics, 287, L9 L12, (1994)
17. Longo's page (downloadable articles), http://www.di.ens.fr/users/longo
18. Longo G.: Set-theoretical models of lambda-calculus: Theories, expansions, isomorphisms. Annals of Pure and Applied Logic 24, 153–188 (1983)
19. Longo G., Moggi E.: A category-theoretic characterization of functional completeness. Theoretical Computer Science 70 (2), 193–211 (1990)
20. Longo G.: Critique of Computational Reason in the Natural Sciences. In: E. Gelenbe et al. (eds.) Fundamental Concepts in Computer Science. Imperial College Press, London (2008)
21. Longo G.: Incomputability in Physics and Biology. To appear in MSCS, 2012 (see Longo's web page). Preliminary version: Computability in Europe. LNCS vol. 6158, Springer, Heidelberg (2010)
22. Pilyugin S. Yu.: Shadowing in dynamical systems, Springer, 1999.
23. Pour-El M.B., Richards J.I.: Computability in analysis and physics. Perspectives in Mathematical Logic, Springer, Berlin, (1989)
24. Schmandt-Besserat D.: How Writing Came About. University of Texas Press, Austin (1992)
25. Schmandt-Besserat D.: From Tokens to Writing: the Pursuit of Abstraction. Bull. Georg. Natl. Acad. Sci. 175(3), 162-167 (2007)
26. Scott D.: Continuous lattices. In: Lawvere (ed.), Toposes, Algebraic Geometry and Logic, SLNM 274, pp. 97-136. Springer, Berlin (1972)
27. Scott D.: Data types as lattices. SIAM Journal of Computing 5, 522-587 (1976)
28. Turing A. M.: Computing Machinery and Intelligence. Mind LIX, 433-460 (1950)
29. Turing A. M.: The Chemical Basis of Morphogenesis, Philo. Trans. Royal Soc. B237, 37-72 (1952)
30. Weihrauch K., Computable analysis, Springer, Berlin (2000)
31. Weihrauch K., Zhong N.: Is wave equation computable or computers can beat the Turing Machine?, Proc. London Math. Soc. (3) 85, 312-332 (2002)