The Church-Turing Thesis and its Look-Alikes

Amir M. Ben-Amram

May 14, 2006

> "A function is mechanically computable (that is: computable by means of a machine) if and only if it is Turing-computable."

I have recently seen the above claim in various contexts. Some researchers endorse it, proposing to view it as a law of nature. Others question its validity. Members of both groups sometimes mislabel it as "the Church-Turing Thesis," and this misconception prompted me to write this note. I would also like to draw the readers' attention to yet another "thesis," dealing with programming languages, and its relationship to the former ones. Understanding the difference between Church and Turing's thesis and other similarly-looking statements is important to anyone interested in the foundations of computability theory. So, let us refer to the above statement as *the Law of Mechanical Computability*, and proceed to contrast it with other similar statements, the first of which is Church and Turing's original thesis.

The goal of Church and Turing's work in the 1930's was to formalize the notion of *effective procedure* in order to enable proofs of negative statements, e.g., "there is no effective procedure to determine validity of predicate-calculus theorems" (Hilbert's *Entscheidungsproblem*). What Hilbert was asking for, with respect to the decision problem, is a "procedure" in the traditional mathematical sense, such as the classical Euclid's Algorithm. Such a procedure is meant to be read and executed by humans. In Turing's celebrated paper (Turing, 1936) he referred to the human executing the procedure as "the computer," possibly confusing future-day readers. What characterizes a procedure of this kind? According to Turing's analysis, the procedure should be deterministic and discrete, involve a finite set of symbols

This is an extended version of a note printed in *SIGACT News* (2005). author's email: `amirben@mta.ac.il`

and a finite set of basic steps that affect a finite number of symbols each [1]. Such restrictions do not obviously apply to mechanical devices. Therefore, the *Church-Turing Thesis*:

> "A function is effectively computable if and only if it is Turing-computable"

Does not imply the Law of Mechanical Computability, as it does not address machines. Neither is it formally implied by the latter (as long as we do not postulate that the human user of the effective procedure is a type of machine). It may be helpful to the modern reader (or student) to have the thesis reformulated as follows:

> "A certain procedure constitutes a (discrete) algorithm for computing a given function if and only if it can be implemented as a Turing Machine that computes that given function."

This version highlights the fact that the Thesis is concerned with mathematical modeling of the concept of discrete algorithm, a concept that would probably be familiar from previous exposure to programming and algorithmics. It also acknowledges the fact that the term *algorithm* has a wider sense; not all algorithms are discrete. I prefer not to explore this avenue further in this note and will omit the qualification in the sequel.

Another take on the effective procedure question is that a mathematical procedure is a *linguistic artifact*. The great logicians of the 19th and early 20th century were concerned with formalizing the language of proofs; it would have been natural to expect that the language of procedures should follow. In fact, powerful, effective languages for the expression of algorithms only appeared when the advances of electronic computers prompted developments in programming languages. But it is important to observe that the Turing Machine *is* a language for expressing algorithms, albeit a cumbersome one. So is Church's way of modeling computational procedures by λ-expressions. Turning to languages, we arrive at a third variant of the Thesis:

> "A certain language is a *reasonable programming language* if it can be automatically translated into the Turing Machine model. An algorithm is any program in a reasonable programming language."

[1] Turing observes: "We [humans] cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same," explaining why unbounded natural numbers should not be allowed as symbols in his model

We remark that the reference to automatic translation is quite important for justifying the transference of certain results in computability between different models. This assumption arises more naturally when discussing programming languages than when discussing either "mechanical devices" or "procedures to be used by humans." What is the relationship of this "linguistic" version to the previous statements? Well, the last version subsumes the original thesis if we postulate that procedures to be used by humans require a formal language for their specification, one that would qualify as a reasonable programming language. It will subsume the Law of Mechanical Computability if we assume that a mechanical computing device is provided with a *description language* that allows its simulation by a discrete algorithm. This seems to be the case for all devices that are known to obey the Law, which suggests that the law is really about amenability to such description (but we are moving here outside the scope of this note).

To conclude, the Church-Turing Thesis originally concerned procedures, or algorithms, intended to be humanly-executable (at least in principle). The ensuing belief in the universality of Church and Turing's computational models (and the numerous equivalent models that I have neglected to mention) has subsequently been extended to the computational power of hardware and software. The resulting propositions are inequivalent to the Thesis and should be distinguished from it. Further variants (not discussed in this note) relate to complexity; once again, it is important to understand that such formulations would not be equivalent to any of the above. To any reader interested in the issues of universal computational models and Turing's work I strongly recommend the volume *The Universal Turing Machine: A Half-Century Survey*, edited by Rolf Herken (2nd edition, Springer 1994). An argument that justifies the Law of Mechanical Computability (under certain restrictions on the class of machines considered—no claims regarding "every possible device"!) is given by (Gandy, 1980).

References

R. Gandy, Church's thesis and principles for mechanisms, in *The Kleene Symposium*, J. Barwise et al., eds., North-Holland, Amsterdam, 1980, pp.123-148.

Turing, Alan M. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society,* Series 2, 42, 230-265, 1936. Corrections ibid. 544-546 (1937).