Practical uses of the Church-Turing Thesis, revisited

Jason Zesheng Chen

Independent

zeshengc@uci.edu

2025/02/28

Overview

Take Home Message

- Using the Church-Turing Thesis (or its analogues elsewhere) in actual mathematical practice serves two slightly different purposes.

Overview

Take Home Message

- Using the Church-Turing Thesis (or its analogues elsewhere) in actual mathematical practice serves two slightly different purposes.
  1. **Rigor Assurance**

Overview

Take Home Message

- Using the Church-Turing Thesis (or its analogues elsewhere) in actual mathematical practice serves two slightly different purposes.
  1. **Rigor Assurance**
  2. **Coding Invariance**

Overview

Take Home Message

- Using the Church-Turing Thesis (or its analogues elsewhere) in actual mathematical practice serves two slightly different purposes.
  1. **Rigor Assurance**
  2. **Coding Invariance**

Overview

Take Home Message

- Using the Church-Turing Thesis (or its analogues elsewhere) in actual mathematical practice serves two slightly different purposes.
  1. **Rigor Assurance**
  2. **Coding Invariance**

Recognizing the distinction leads to nice philosophical payoff.

- Certain disagreements in the literature are naturally dissolved.

Disclaimer

What this talk is *not* about

Disclaimer

What this talk is *not* about

In this talk, we will set aside the most classical use of the CTT: that our formalization of the intuitive notion of effective procedure is *right*.

Disclaimer

What this talk is *not* about

In this talk, we will set aside the most classical use of the CTT: that our formalization of the intuitive notion of effective procedure is *right*.

Instead...

We will focus on the use of the CTT in the technical literature, in its capacity in facitlitating proofs and mathematical ideas.

The Church-Turing Thesis

The CTT

A procedure is effectively feasible if and only if it can be implemented by a Turing machine.

How the CTT often gets used

Nies (2012). *Computability and Randomness*

Many other formal definitions for the intuitive notion of a computable function were proposed.

How the CTT often gets used

Nies (2012). *Computability and Randomness*

Many other formal definitions for the intuitive notion of a computable function were proposed. All turned out to be equivalent. This lends evidence to the Church-Turing thesis which states that any intuitively computable function is computable in the sense of [Turing machines].

How the CTT often gets used

Nies (2012). *Computability and Randomness*

Many other formal definitions for the intuitive notion of a computable function were proposed. All turned out to be equivalent. This lends evidence to the Church-Turing thesis which states that any intuitively computable function is computable in the sense of [Turing machines]. More generally, each informally given algorithmic procedure can be implemented by a Turing program.

How the CTT often gets used

Nies (2012). *Computability and Randomness*

Many other formal definitions for the intuitive notion of a computable function were proposed. All turned out to be equivalent. This lends evidence to the Church-Turing thesis which states that any intuitively computable function is computable in the sense of [Turing machines]. More generally, each informally given algorithmic procedure can be implemented by a Turing program.We freely use this thesis in our proofs: we give a procedure informally and then take it for granted that a Turing program implementing it exists.

How the CTT often gets used

Nies (2012). *Computability and Randomness*

Many other formal definitions for the intuitive notion of a computable function were proposed. All turned out to be equivalent. This lends evidence to the Church-Turing thesis which states that any intuitively computable function is computable in the sense of [Turing machines]. More generally, each informally given algorithmic procedure can be implemented by a Turing program.We freely use this thesis in our proofs: we give a procedure informally and then take it for granted that a Turing program implementing it exists.

This is known in the literature as "proof by Church's Thesis (or proof by the Church-Turing Thesis)"

Point of the Nies quote

1. To recall a familiar use of the CTT in the technical literature.

Point of the Nies quote

1. To recall a familiar use of the CTT in the technical literature.
2. To illustrate a departure of the primary use of the CTT in actual mathematical practice from how it is used in more philosophical contexts.

Rigor Assurance

Rogers (1987). *Theory of recursive functions and effective computability*

Such methods ... permit us to avoid cumbersome detail and to isolate crucial mathematical ideas from a background of routine manipulation. We shall see that much profound mathematical substance can be discussed, proved, and communicated in this way ... Proofs which rely on informal methods have, in their favor, all the evidence accumulated in favor of Church's Thesis.

Outside of classical computability

Definition

For an admissible ordinal α and $A \subseteq \alpha$ the following are equivalent:

1. A is $\sum_1$-definable in L_α.
2. A is computably enumerable by Koepke's α-Turing machines.
3. A is semi-decidable by Koepke's α-register machines.

Outside of classical computability

Greenberg (2020). *Two applications of admissible computability*

In general, working in α-computability, with experience, we apply some kind of Church-Turing thesis to α-computable functions ... we eventually cease to write down precise Σ_1 formulas ... Instead, we develop an intuition as to what constitutes "legal" α-computable manipulations of α-finite objects (elements of L_α), and get a sense of the "time" that a process takes; if it takes fewer than α steps, then it "halts".

Outside of classical computability

Greenberg (2020). *Two applications of admissible computability*

In general, working in α-computability, with experience, we apply some kind of Church-Turing thesis to α-computable functions ... we eventually cease to write down precise Σ_1 formulas ... Instead, we develop an intuition as to what constitutes "legal" α-computable manipulations of α-finite objects (elements of L_α), and get a sense of the "time" that a process takes; if it takes fewer than α steps, then it "halts".

Notice

Greenberg is certainly not expecting the reader to develop an intuition for what is effective infinitarily (whatever that means). He is expecting the reader to develop an intuition for how to translate informal argument into formal ones.

One more example

Hamkins & Lewis, *Infinite Time Turing Machines*

"We will assume complete familiarity with the notions of Turing machines and ordinals and, in describing our algorithms, take the high road to avoid getting bogged down in Turing machine minutiae. We hope the readers will appreciate our saving them from reading what would otherwise resemble computer code." (Hamkins & Lewis, 2000)

Rigor Assurance

The fact that numerous formalizations of the same intuitive concept turn out to be equivalent provides a kind of assurance that intuitive, informal, natural language used in proofs can always be safely translated back to formal language, if one wishes.

Question

In the context of facilitating a proof, is this the only purpose that is served by appeals to the Church-Turing Thesis?

San Mauro's tension

The Standard View towards proofs by Church's Thesis (according to San Mauro (2018))

Proof by Church's Thesis = proof is left to the reader.

San Mauro's tension

The Standard View towards proofs by Church's Thesis (according to San Mauro (2018))

Proof by Church's Thesis = proof is left to the reader.

i.e., exactly the kind of time-saving practice familiar to mathematicians from any other field. No additional significance.

San Mauro's tension

The Standard View towards proofs by Church's Thesis (according to San Mauro (2018))

Proof by Church's Thesis = proof is left to the reader.

San Mauro: taking this attitude misses out on a key aspect of what makes computability unique.

San Mauro's illustration: construction arguments in computability

- Many computability arguments begin with enumerating (say) the r.e. sets.

San Mauro's illustration: construction arguments in computability

- Many computability arguments begin with enumerating (say) the r.e. sets.
- Using this enumeration we construct many other objects of interest.

San Mauro's illustration: construction arguments in computability

- Many computability arguments begin with enumerating (say) the r.e. sets.
- Using this enumeration we construct many other objects of interest.
- Taking the Standard View on this kind of constructions entails that one still cares about the enumeration, despite not being bothered to write it down.

San Mauro's illustration: construction arguments in computability

- Many computability arguments begin with enumerating (say) the r.e. sets.
- Using this enumeration we construct many other objects of interest.
- Taking the Standard View on this kind of constructions entails that one still cares about the enumeration, despite not being bothered to write it down.

San Mauro's illustration: construction arguments in computability

- Many computability arguments begin with enumerating (say) the r.e. sets.
- Using this enumeration we construct many other objects of interest.
- Taking the Standard View on this kind of constructions entails that one still cares about the enumeration, despite not being bothered to write it down. So the proof is technically a proof schema, giving a recipe of construction for each enumeration.
- But what makes computability unique is that the properties of interest remain invariant under different enumerations. They are in some sense absolute properties of the objects, not a result of the coding.

San Mauro's illustration: construction arguments in computability

- Many computability arguments begin with enumerating (say) the r.e. sets.
- Using this enumeration we construct many other objects of interest.
- Taking the Standard View on this kind of constructions entails that one still cares about the enumeration, despite not being bothered to write it down. So the proof is technically a proof schema, giving a recipe of construction for each enumeration.
- But what makes computability unique is that the properties of interest remain invariant under different enumerations. They are in some sense absolute properties of the objects, not a result of the coding.
- This dis-entanglement with formalisms is what makes computability unique and what proponents of the Standard View are missing out on.

San Mauro's conclusion

San Mauro (2018)

[The set constructed in the example] does not refer to any of its formal definitions ... The notion ... is better understood as an absolute one, i.e. independent from the chosen formalism

In other words

San Mauro's claim

Appealing to the CTT in a proof achieves more than, say, leaving an exercise to the reader. It commits us to dealing with the One True Notion of computation.
Proponents of the Standard View overlooks this aspect when they reduce the practice to exercise-left-to-reader.

My diagnosis

There's no serious disagreement between San Mauro and proponents of the Standard View. The apparent disagreement stems from a conflation between two different purposes served by appeals to CTT:

My diagnosis

There's no serious disagreement between San Mauro and proponents of the Standard View. The apparent disagreement stems from a conflation between two different purposes served by appeals to CTT:

- Having faith that my reader can fill in the formal details. (**Rigor Assurance**)
- versus
- Not caring how they choose to fill in the formal details. Because it doesn't matter. (**Coding Invariance**)

Coding Invariance

Kleene (1952)

"The notion of λ-definability has the variants λ-K-definability ... and λ-δ-definability ... also there is a parallel development, started by [Schönfinkel, Curry, and Rosser], which leads to a notion that we may call combinatory definability, proved equivalent to λ-definability by Rosser."

Coding Invariance

Kleene (1952)

"The notion of λ-definability has the variants λ-K-definability ... and λ-δ-definability ... also there is a parallel development, started by [Schönfinkel, Curry, and Rosser], which leads to a notion that we may call combinatory definability, proved equivalent to λ-definability by Rosser."

Notice

This is a different facet to the CTT than the equivalence between different formalisms like Turing machines and λ-calculus. This is invariance under different codifications of the same formalism.

Moschovakis (2016). *Hyperarithmetical Sets*

"Codings are useful for expressing succinctly uniform properties of coded sets. ... It is clear that propositions ... which hold uniformly for a certain coding also hold uniformly for every equivalent coding."

Moschovakis (2016). *Hyperarithmetical Sets*

"Codings are useful for expressing succinctly uniform properties of coded sets. ... It is clear that propositions ... which hold uniformly for a certain coding also hold uniformly for every equivalent coding."

Moschovakis (2016). *Hyperarithmetical Sets*

"For a classical example, consider the coding of recursive partial functions specified by [Kleene's Normal Form Theorem]. Its precise definition depends on the choice of computation model that we use, Turing machines, systems of recursive equations or whatever [that is, each choice of model gives rise to a different enumeration of the recursive partial functions], but all these codings are equivalent and so uniform propositions about them are coding invariant."

Failure of Coding Invariance

1. One of the earliest definitions of quantum Turing machines allowed for arbitrary transition amplitudes. (Bernstein & Vazirani, 1993)

Failure of Coding Invariance

1. One of the earliest definitions of quantum Turing machines allowed for arbitrary transition amplitudes. (Bernstein & Vazirani, 1993)
2. The amplitudes, being real numbers, can code all sorts other information. This enabled Adleman et al. (1997) to prove that the class of sets decidable with bounded error in polynomial time has uncountable cardinality and contains sets of all Turing degrees.

Failure of Coding Invariance

1. One of the earliest definitions of quantum Turing machines allowed for arbitrary transition amplitudes. (Bernstein & Vazirani, 1993)
2. The amplitudes, being real numbers, can code all sorts other information. This enabled Adleman et al. (1997) to prove that the class of sets decidable with bounded error in polynomial time has uncountable cardinality and contains sets of all Turing degrees.
3. Subsequent journal version of Bernstein and Vazirani (1997) corrected the definition to only allow efficiently computable transition amplitudes.

Failure of Coding Invariance

1. One of the earliest definitions of quantum Turing machines allowed for arbitrary transition amplitudes. (Bernstein & Vazirani, 1993)
2. The amplitudes, being real numbers, can code all sorts other information. This enabled Adleman et al. (1997) to prove that the class of sets decidable with bounded error in polynomial time has uncountable cardinality and contains sets of all Turing degrees.
3. Subsequent journal version of Bernstein and Vazirani (1997) corrected the definition to only allow efficiently computable transition amplitudes.
4. "we need to constrain the entries allowed in the transition function ... Otherwise, it is possible to smuggle hard-to-compute quantities into the transition amplitudes [such as the solution to the halting problem]."

Failure of Coding Invariance

1. One of the earliest definitions of quantum Turing machines allowed for arbitrary transition amplitudes. (Bernstein & Vazirani, 1993)
2. The amplitudes, being real numbers, can code all sorts other information. This enabled Adleman et al. (1997) to prove that the class of sets decidable with bounded error in polynomial time has uncountable cardinality and contains sets of all Turing degrees.
3. Subsequent journal version of Bernstein and Vazirani (1997) corrected the definition to only allow efficiently computable transition amplitudes.
4. Careful choices were then made in the paper to ensure that, in terms of computability (i.e., what are computable simplciter), the resulting machines are equivalent to classical Turing machines.
5. The resulting definition of quantum Turing machines is now the canon.

Example from logic: iterated consistency

Define $T_0 := \mathrm{ZFC}$, $T_{2^n} := T_n + \mathrm{Con}(T_n)$, $T_{3 \cdot 5^e} = \mathrm{ZFC} \cup \bigcup T_{\Phi_e(n)}$. Now coding peculiarity follows:

Example from logic: iterated consistency

Define $T_0 := \mathrm{ZFC}$, $T_{2^n} := T_n + \mathrm{Con}(T_n)$, $T_{3\cdot 5^e} = \mathrm{ZFC} \cup \bigcup T_{\Phi_e(n)}$. Now coding peculiarity follows:

Theorem (Turing's completeness theorem)

For every true Π^0_1 sentence φ, there exists a notation in d in Kleene's $\mathcal{O}$ such that φ is provable in T_d.

Example from logic: iterated consistency

Define $T_0 := \mathrm{ZFC}$, $T_{2^n} := T_n + \mathrm{Con}(T_n)$, $T_{3\cdot 5^e} = \mathrm{ZFC} \cup \bigcup T_{\Phi_e(n)}$. Now coding peculiarity follows:

Theorem (Turing's completeness theorem)

For every true Π^0_1 sentence φ, there exists a notation in d in Kleene's $\mathcal{O}$ such that φ is provable in T_d.

Lesson

If we want to talk about iterated consistency statements, then the theories in the limit are susceptible to coding peculiarities.

A quick comparison

Rigor Assurance

Confluence $\Rightarrow$ not having to worry about informal language breaking a proof.

Coding Invariance

Confluence $\Rightarrow$ not having to worry that different codifications end up making the proofs talk about something else entirely.

A very illuminating example

Invariant descriptive set theory

The abstract study of how difficult classification problems are. (E.g., "If I know how to tell whether these two quantities are identical, then can I tell whether these two structures are isomorphic?")

A very illuminating example

Invariant descriptive set theory

The abstract study of how difficult classification problems are. (E.g., "If I know how to tell whether these two quantities are identical, then can I tell whether these two structures are isomorphic?")

A worry

In IDST, we are not really talking about the mathematical structures themselves, but talk about coded versions of them.

A very illuminating example

Invariant descriptive set theory

The abstract study of how difficult classification problems are. (E.g., "If I know how to tell whether these two quantities are identical, then can I tell whether these two structures are isomorphic?")

A worry

In IDST, we are not really talking about the mathematical structures themselves, but talk about coded versions of them. E.g., a finitely generated countable group is coded by a function $f : \mathbb{N}^2 \to \mathbb{N}$, telling us how the group operation behaves; or as a subgroup of some universal group (say the free group F_ω).

A very illuminating example

Gao's Thesis, (Gao, 2008, p. 328)

For any class $\mathcal{H}$ of mathematical structures, if (X_1, Ω_1) and (X_2, Ω_2) are two standard Borel spaces naturally coding elements of $\mathcal{H}$, then there exists a Borel map $f : X_1 \to X_2$ such that $f(x)$ and x are isomorphic as mathematical structures for every $x \in \mathcal{H}$.

A very illuminating example

Gao's Thesis, (Gao, 2008, p. 328)

"Any choice of coding/presentation of a structure is as good as any other."

A very illuminating example

Gao's Thesis, (Gao, 2008, p. 328)

"Any choice of coding/presentation of a structure is as good as any other."

This is classic Coding Invariance. Compare it with Gao's earlier appeal to proof by the Church-Turing Thesis:

A very illuminating example

Gao's Thesis, (Gao, 2008, p. 328)

"Any choice of coding/presentation of a structure is as good as any other."

This is classic Coding Invariance. Compare it with Gao's earlier appeal to proof by the Church-Turing Thesis:

Gao (2008, p. 24)

All formal definitions of computability have been shown to be equivalent ... we will not deal with the details of the above definition, but will rather adopt the Church-Turing Thesis ... Thus if a function is intuitively computable by an informal algorithm then by the Church-Turing Thesis we may conclude that it is formally computable without checking the details of the formal definitions.

Rigor Assurance

Confluence $\Rightarrow$ not having to worry about invalidity brought by translating from informal to formal language.

Coding Invariance

Confluence $\Rightarrow$ not having to worry pseudo-insights smuggled in by the choice of coding.

Resolution of San Mauro's tension

- The Standard View is a view about the Rigor Assurance aspect of how CTT is used
- Proponents of SV are right that this is nothing more than leaving the exercise to the reader.
- This is different from Coding Invariance.
- San Mauro is right that the remarkable invariance under different codings is unique to the study of computability.

One closing disclaimer...

Maddy, $V = L$ and Maximize

Given that a naturalistic philosopher brings no special modes of argument from philosophy, every argument she gives must be based on modes of argument available to any mathematician qua mathematician; at best, she will make explicit what is already implicit. Unsatisfying as this may be in dramatic terms, a good naturalistic argument should not strike the practitioner as late- breaking news; at best, it will fall so far short of originality as to qualify as a commonplace. Given this goal, the best confirmation of success would be for the mathematician to shrug and say, 'Of course, everybody knows that.'

References I

- Adleman, L. M., DeMarrais, J., & Huang, M.-D. A. (1997).Quantum Computability. *SIAM Journal on Computing*, *26*(5), 1524–1540. https://doi.org/10.1137/S0097539795293639
- Bernstein, E., & Vazirani, U. (1993).Quantum complexity theory. *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, 11–20. https://doi.org/10.1145/167088.167097
- Bernstein, E., & Vazirani, U. (1997).Quantum Complexity Theory. *SIAM Journal on Computing*, *26*(5), 1411–1473. https://doi.org/10.1137/S0097539796300921
- Gao, S. (2008, September 3). *Invariant Descriptive Set Theory* (0th ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781584887942
- Hamkins, J. D., & Lewis, A. (2000).Infinite time Turing machines. *Journal of Symbolic Logic*, *65*(2), 567–604. https://doi.org/10.2307/2586556

References II

Kleene, S. C. (1952). *Introduction to Metamathematics*. Van Nostrand.

San Mauro, L. (2018). Church-Turing Thesis, in Practice. In M. Piazza & G. Pulcini (Eds.), *Truth, Existence and Explanation: FilMat 2016 Studies in the Philosophy of Mathematics* (pp. 225–248). Springer International Publishing. https://doi.org/10.1007/978-3-319-93342-9_13