The Church-Turing Thesis and Relative Recursion

Yiannis N. Moschovakis
UCLA and University of Athens

Amsterdam, September 7, 2012

The Church -Turing Thesis (1936) in a contemporary version:

CT: For every function $f : \mathbb{N}^n \to \mathbb{N}$ on the natural numbers,

f is computable by an algorithm

$\Longleftrightarrow$ *f is computable by a Turing machine*

which implies that for every relation R on $\mathbb{N}$

R can be decided by an algorithm

$\Longleftrightarrow$ *R can be decided by a Turing machine*

- Church said it first, Turing said it better!
- Turing machine $\sim$ computer with access to unlimited memory

Most often applied in its "non-trivial" direction:

If R cannot be decided by a Turing machine

then R is absolutely undecidable

First, motivating application: the Entscheidungsproblem

Theorem (Church,Turing, 1936)

No algorithm can decide whether an arbitrary sentence of First Order Logic is provable

First Order Language (a formal fragment of mathematical English):

- Symbols for constants, relations, functions and $=$
- Variables $v_0, v_1, \ldots$ and punctuation symbols ( ) ,
- Symbols for the propositional connective $\neg, \&, \vee, \rightarrow$
- Symbols for the quantifiers $\forall$ (for all), $\exists$ (there exists)
- (Formal) Sentences: grammatically correct strings of symbols, e.g.,

$$(\forall x)(\exists y)\text{Father}(y, x) \implies (\exists y)(\forall x)\text{Father}(y, x)$$

First Order Logic: A proof system (axioms and rules) for sentences

Every mathematical theorem can be formalized in FOL, $\underline{\text{Axioms} \Rightarrow \theta}$

Absolutely unsolvable problems in CS, mathematics, etc.

- Whether a given program in a "complete" programming language will terminate (given enough time and memory) (Turing's original Halting Problem, 1936)
- Whether two words represent the same element in a finitely generated, finitely presented cancellation semigroup (Post, 1940s)
- Whether two words represent the same element in a finitely generated, finitely presented group (Boone, Novikov, 1950s)
- Whether two compact, orientable manifolds of dimension ≥ 4 (given by triangulations) are homeomorphic (A. Markov)
- Hilbert's 10th problem: whether an arbitrary polynomial equation

  $$p(x_1, \ldots, x_n) = 0$$

  with integer coefficients has an integer root (Matiyasevich 1970, following Julia Robinson, Martin Davis and Putnam in the 1960s)

Why is the Church-Turing Thesis true?

CT: For every function $f : \mathbb{N}^n \to \mathbb{N}$ on the natural numbers,
f is computable by an algorithm
$\Longleftrightarrow$ *f is computable by a Turing machine*

- It is now universally accepted, partly because
  - of the analysis in Turing 1936 (and subsequent elucidations)
  - no counterexamples have been found in more than 70 years
  - the developments in Computer Science

But none of these is completely convincing, so

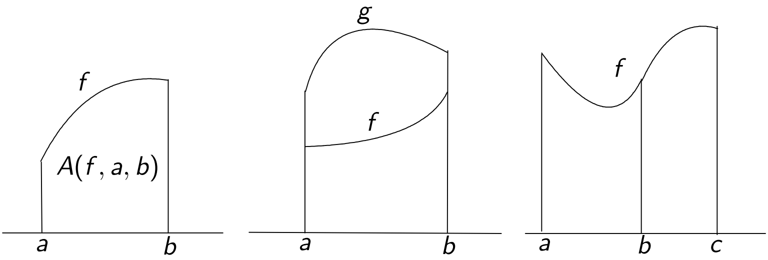
- Can we give a rigorous, mathematical proof of CT?

- Within mathematics, CT is used as a definition:

(imprecise) f is computable $\sim$ (precise) f is computable by a TM

And can one prove a definition?

Proving definitions!



$A(f, a, b) =$ the area above the axis, below f and between a and b
Assume that for all continuous f with the figures as in the drawing:

- $A(f, a, b) \geq 0, \quad A(f, a, b) \leq A(g, a, b)$
- $A(f, a, c) = A(f, a, b) + A(f, b, c)$
- Calibration: area of a rectangle = base × height

Thm For every continuous f,

$$A(f, a, b) = \int_a^b f(x)dx$$

Some points from Turing's analysis

- There is no mention of "algorithms" in Turing 1936

  - *"The computable numbers may be described as the real numbers whose decimal expansions*

    . . . are calculable by finite means

    . . . can be written down by a machine"

  - *"We may compare a man in the process of computing a real number to a machine which is only capable of . . . "*

  - *"It is my contention that these [his] operations include all those which are used in the computation of a number . . . "*

- Gandy (1980): TM's capture *routine computation by a clerk*, but CT holds for computability by mechanical devices

- *What mechanical devices might be available in 2112?*

Some comments on Church's formulation

- CT*: "Every function, an algorithm for the calculation of the values of which exists, is [Turing computable]"*
- *"An algorithm consists in a method by which, given any positive integer* n*, a sequence of expressions (in some notation)* $E_{n1}, E_{n2}, \ldots, E_{n,r_n}$ *can be obtained; . . . the fact that the algorithm has terminated becomes effectively known [proved] and the value of* $F(n)$ *is effectively calculable"*
- *"If this interpretation or some similar one is not allowed, it is difficult to see how the notion of an algorithm can be given any exact meaning at all"*

  (Kripke (2000) suggests that this argument practically proves CT)

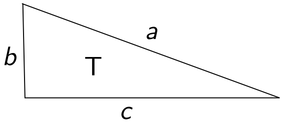
The analyses of Turing, Church (and most others) assume that:

- All computation is symbolic
- Input and output functions on $\mathbb{N}$ are needed to start and finish

What kind of a proposition is CT?

For any proposition A, we say that:

- A is empirical if it refers to the physical world
- A is mathematical if it is about mathematical objects
- A is logical if it is true or false *by logic alone*



PT: If T is a right triangle
then $a^2 = b^2 + c^2$

- PT is a mathematical truth (about lines, triangles, lengths, etc.)
- $\underline{\textit{Axioms of Euclidean geometry} \implies PT}$ is a logical truth
- If "lines" are the paths of light rays,
  then PT is empirical—true or false depending on your physics

CT is not a logical truth

CT: For every function $f : \mathbb{N}^n \to \mathbb{N}$ on the natural numbers,
f is computable by an algorithm
$\iff$ *f is computable by a Turing machine*

- If we allow algorithms to be implemented by "mechanical devices" as Gandy would like, then CT is empirical
- The operations that "a clerk might do" are mathematical operations (on natural numbers or strings of symbols); so if we only allow these, then CT is mathematical
- CT is not logical, because it depends on what the numbers are and how algorithms operate on them

- Obstructions to a proof:
- The relativization problem: distinguish absolute computation from computation relative to an oracle (missing "calibration")
- No intuitions for what is "non-computable"

The Euclidean algorithm (before 300 BC)

For $a, b \in \mathbb{N} = \{0, 1, \ldots\}$, $a \geq b \geq 1$,

$$\gcd(a, b) = \text{the largest number which divides both } a \text{ and } b$$

Basic mathematical fact about the greatest common divisor function:

$$(\varepsilon) \quad \boxed{\gcd(a, b) = \text{if } (\text{rem}(a, b) = 0) \text{ then } b \text{ else } \gcd(b, \text{rem}(a, b))}$$

where $a = qb + \text{rem}(a, b)$ (for some q and $0 \leq \text{rem}(a, b) < b$)

- (ε) expresses an algorithm from rem, $=_0$ for computing $\gcd(a, b)$
- The important facts about ε are its correctness and its complexity:

$$\text{calls}_\varepsilon(a, b) = \text{the number of divisions } \varepsilon \text{ makes to compute } \gcd(a, b)$$
$$\leq 2 \log_2(b) \qquad (a \geq b \geq 2)$$

- The Euclidean operates directly on numbers: there is no need for intermediate "syntactic expressions", "input representation", etc.

Two more algorithms from primitives in mathematics

- The Sturm algorithm (1829): Computes *the number of roots* of a polynomial

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n \qquad (*)$$

of degree $\leq n$ with real coefficients in a real interval (b, c)

  - Operates on tuples $(a_0, \ldots, a_n, b, c)$ of real numbers
  - Primitives: the field operations $+, -, \cdot, \div$ and the ordering $\leq$ in $\mathbb{R}$

- Horner's rule ($\sim$ 1250): Computes the value of a polynomial $(*)$ of degree $\leq n$ in an arbitrary field F
  - Operates on tuples $(a_0, \ldots, a_n, x)$ from F
  - Primitives: The field operations $0, 1, +, -, \cdot, \div$ of F
  - The basic mathematical fact used by Horner's Rule:

$$a_0 + a_1 x + \cdots + a_{n+1} x^{n+1} = a_0 + x\Big(a_1 + a_1 x + \cdots + a_{n+1} x^n\Big)$$

  - Optimal for generic inputs in $\mathbb{R}, \mathbb{C}$ (Pan 1966)

Computability from arbitrary primitives

$$\mathbf{A} = (A, \mathbf{\Phi}) = (A, c_0, \dots, c_{k-1}, R_1, \dots, R_{l-1}, \phi_1, \dots, \phi_{m-1})$$

Def A function $f : A^n \to A$ or an n-ary relation R on A is

recursive in $\mathbf{A}$ or from $\mathbf{\Phi}$

if is it is computed by a recursive (McCarthy) program

Recursive programs are systems of mutually recursive equations constructed using

- Variables over A and *partial functions and relations* on A
- Names for the primitives in $\mathbf{\Phi}$
- Composition (calls)
- Conditionals (branching)

$\sim$ programs in a programming language with full recursion,
interpreted over A and with access to unlimited memory and time

The Relative Recursion Thesis

$$\mathbf{A} = (A, \mathbf{\Phi}) = (A, c_0, \ldots, c_{k-1}, R_1, \ldots, R_{l-1}, \phi_1, \ldots, \phi_{m-1})$$

RRT: For every function $f : A^n \to A$ on an arbitrary set A,

f is computable from given primitives $\mathbf{\Phi}$ on A

$\iff$ f is recursive in the structure $\mathbf{A} = (A, \mathbf{\Phi})$

(and similarly with relations)

- Arguments in favor of RRT:
  - It covers all examples of algorithms in mathematics which compute functions from specified primitives
  - There are no known counterexamples
  - *Recursive programs can be implemented* (using oracles for $\mathbf{\Phi}$)
  - One can give an analysis of the notion of relative algorithm which supports RRT (as Turing's analysis supports CT)

RRT is logical (true or false by logic alone)

Tarski on logical notions (1986): A set in the type structure over a non-empty A is logical if it is invariant under all (automorphisms of the type structure induced by) permutations of A

Equality $=_A$ on A and the existential quantifier $\exists^A$ are logical because for every permutation $\pi : A \rightarrowtail\!\!\!\!\twoheadrightarrow A$

$$x = y \iff \pi(x) = \pi(y),$$
$$(\exists^A x)R(x) \iff (\exists^A x)R^\pi(x) \text{ with } R^\pi(y) \iff R(\pi(y))$$

- $\Big\{(f, \mathbf{\Phi}) : f \text{ is recursive in } (A, \mathbf{\Phi})\Big\}$ is logical (easy theorem)
- $\Big\{(f, \mathbf{\Phi}) : f \text{ is computable from } \mathbf{\Phi}\Big\}$ is logical (intuitively clear!)

  Basic intuition: an algorithm from $\mathbf{\Phi}$ uses only logical operations and calls to $\mathbf{\Phi}$

Thm *The Relative Recursion Thesis is a logical proposition*

Some advantages of relative over absolute computability

$\mathbf{rec}(A, \mathbf{\Phi}) =$ the set of all functions and relations on A
which are recursive from $\mathbf{\Phi}$

- Foundations: It is easier to understand a *general theory* with many models: RRT *is easier to understand than* CT
  - Examples: Many interesting ones with A other than the natural numbers, in algebra and computer science
  - Generalizations, e.g., Kleene's higher type recursion, Normann's set recursion, inductive definability.
    These theories have important applications in logic and set theory
- Complexity theory: Recursive programs from specified primitives carry a rich theory of computational complexity

RRT: The basic logical primitives of computation are

composition, branching and mutual recursion

A reduction of CT to RRT + the standard view

Claim *A function* $f : \mathbb{N}^n \to \mathbb{N}$ *is computable*
if and only if f *is computable in the structure* $(\mathbb{N}, 0, S, =)$

— because the structure $(\mathbb{N}, 0, S, =)$ is what the natural numbers are!

- $(\mathbb{N}, 0, S, =)$ is a Peano system, i.e., $S : \mathbb{N} \rightarrowtail\!\!\!\twoheadrightarrow \mathbb{N} \setminus \{0\}$ and *every* $X \subseteq \mathbb{N}$ *which contains* 0 *and is closed under the successor* S *exhausts* $\mathbb{N}$ (the Induction Axiom)
- Any two Peano systems are isomorphic (Dedekind)
- The standard view: The natural numbers are a Peano system
  — nothing less and nothing more

Theorem (Kleene, McCarthy)

For every $f : \mathbb{N}^n \to \mathbb{N}$

$$f \in \mathbf{rec}(\mathbb{N}, 0, S, =) \iff f \text{ is Turing computable}$$

Theorem: $\Big(\text{RRT} + \textit{the standard view}\Big) \implies \text{CT}$