Other Approaches to Computability
Church's Thesis*

Xiaofeng Gao

Department of Computer Science and Engineering
Shanghai Jiao Tong University, P.R.China

CS363-Computability Theory

*Special thanks is given to Prof. Yuxi Fu for sharing his teaching materials.

Outline

Recursive Function

Three Basic Functions:

- The *zero function* $\mathbf{0}$.
- The *successor function* $x+1$.
- For each $n \geq 1$ and $1 \leq i \leq n$, the *projection function* U_i^n given by $U_i^n(x_1, \ldots, x_n) = x_i$.

Three Operations:

- *Substitution*: $h(\mathbf{x}) \simeq f(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$.
- *Recursion*: $\begin{cases} h(\mathbf{x}, 0) \simeq f(\mathbf{x}), \\ h(\mathbf{x}, y+1) \simeq g(\mathbf{x}, y, h(\mathbf{x}, y)). \end{cases}$
- *Minimalisation*: $\begin{cases} \text{Bounded: } \mu z{<}y(f(\mathbf{x}, z) = 0), \\ \text{Unbounded: } \mu y(f(\mathbf{x}, y) = 0). \end{cases}$

Primitive Recursive Function

The class $\mathscr{PR}$ of primitive recursive functions is the smallest class of partial functions that contains the basic functions $\mathbf{0}$, $x+1$, U_i^n and is closed under the operations of substitution and recursion.

Note: $\mathscr{PR}$ includes the operations of bounded minimalisation, since it can be rephrased as the combinations of substitution and recursion.

$$\mu z{<}y(f(\mathbf{x}, z) = 0) \simeq \sum_{v<y}\Big(\prod_{u \leq v} \mathrm{sg}(f(\mathbf{x}, u))\Big).$$

Partial Recursive Functions (Gödel-Kleene, 1936)

The class $\mathscr{R}$ of partial recursive functions is the smallest class of partial functions that contains the basic functions $\mathbf{0}$, $x+1$, U_i^n and is closed under the operations of substitution, recursion and minimalisation.

Notice that there is no totality restriction placed on the use of the μ-operator.

Partial Recursive Functions (Gödel-Kleene, 1936)

Gödel and Kleene originally defined the set $\mathscr{R}_0$ of μ-recursive functions.

In the definition of the μ-recursive functions, the μ-operator is allowed to apply only if it produces a total function.

In fact $\mathscr{R}_0$ is the set of all the total functions in $\mathscr{R}$.

Partial Recursive Functions are Computable Functions

Theorem. $\mathscr{R} = \mathscr{C}$.

Proof. We have proved that $\mathscr{R} \subseteq \mathscr{C}$. We have to show the reverse inclusion.

Partial Recursive Functions are Computable Functions

Suppose that $f(\mathbf{x})$ is a URM-computable function, computed by a program $P = I_1, \ldots, I_s$.

$$c(\mathbf{x}, t) = \begin{cases} r_1, & \text{the content of } R_1 \text{ after } t \text{ steps of } P(\mathbf{x}), \\ & \text{if } P(\mathbf{x}) \text{ has not stopped after } t{-}1 \text{ steps;} \\ r_1, & \text{the final content of } R_1 \text{ if } P(\mathbf{x}) \text{ stops} \\ & \text{in less than } t \text{ steps.} \end{cases}$$

$$j(\mathbf{x}, t) = \begin{cases} k, & k \text{ is the number of the next instruction after} \\ & t \text{ steps of } P(\mathbf{x}) \text{ have been performed;} \\ 0, & \text{if } P(\mathbf{x}) \text{ has stopped after } t \text{ steps or fewer.} \end{cases}$$

Fact. Both $c(\mathbf{x}, t)$ and $j(\mathbf{x}, t)$ are primitive recursive.

Partial Recursive Functions are Computable Functions

If $f(\mathbf{x})$ is defined, then $P(\mathbf{x})$ converges after exactly t_0 steps, where

$$t_0 = \mu t(j(\mathbf{x}, t) = 0), \text{ and } f(\mathbf{x}) = c(\mathbf{x}, t_0).$$

Else $f(\mathbf{x})$ is undefined $\Rightarrow P(\mathbf{x}) \uparrow \Rightarrow j(\mathbf{x}, t) \neq 0$ and $\mu t(j(\mathbf{x}, t) = 0)$ is undefined.

Thus function $f(\mathbf{x})$ defined by $P(\mathbf{x})$:

$$f(\mathbf{x}) \simeq c(\mathbf{x}, \mu t(j(\mathbf{x}, t) = 0)).$$

is partial recursive.

Corollary

Corollary. Every total function in $\mathscr{R}$ belongs to $\mathscr{R}_0$.

Proof: Suppose $f(\mathbf{x})$ is total in $\mathscr{R}$, then f is URM-computable by a program P.

Let c and j be the same definitions, which can be obtained without any use of minimalisation, so they are in $\mathscr{R}$.

Further, since f is total, $P(\mathbf{x})$ converges for every x, so the function $\mu t(j(\mathbf{x}, t) = 0)$ is total and belongs to $\mathscr{R}$.

Now $f(\mathbf{x}) = c(\mathbf{x}, \mu t(j(\mathbf{x}, t) = 0))$, so f is also in $\mathscr{R}$.

Predicate

A predicate $M(\mathbf{x})$ whose characteristic function c_M is recursive is called a *recursive predicate*.

A recursive predicate is the same as decidable predicate.

Alan Turing (23 Jun. 1912 - 7 Jun. 1954)

- An English student of Church
- Introduced a machine model for effective calculation in "On Computable Numbers, with an Application to the Entscheidungsproblem", Proc. of the London Mathematical Society, 42:230-265, 1936.
- Turing Machine, Halting Problem, Turing Test

Motivation

What are necessary for a machine to calculate a function?

- The machine should be able to interpret numbers
- The machine must be able to operate and manipulate numbers according to a set of predefined instructions

and

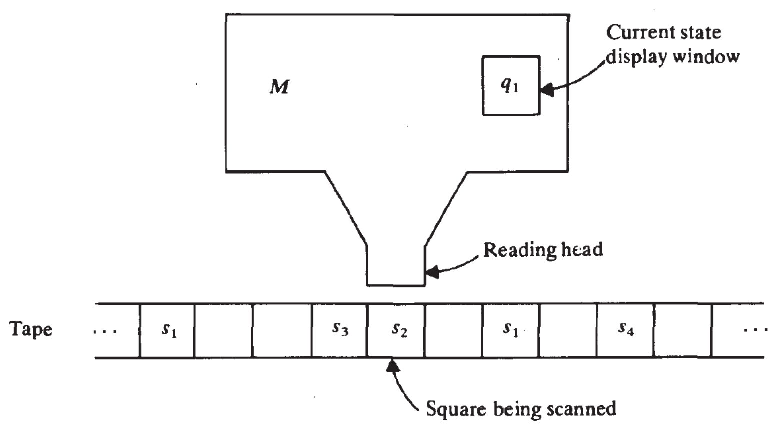
- The input number has to be stored in an accessible place
- The output number has to be put in an accessible place
- There should be an accessible place for the machine to store intermediate results

One-Tape Turing Machine

A Turing machine has five components:

1. A finite set $\{s_1, \ldots, s_n\} \cup \{\rhd, \sharp, \lhd\} \cup \{\Box\}$ of symbols.
2. A tape consists of an infinite number of cells, each cell may store a symbol.
$$\cdots \Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box \cdots$$
3. A reading head that scans and writes on the cells.
4. A finite set $\{q_S, q_1, \ldots, q_m, q_H\}$ of states.
5. A finite set of instructions (specification).

One-Tape Turing Machine



Turing Machines, Turing 1936

The input data

$$\rhd s_1^1 \ldots s_{i_1}^1 \Box \ldots \Box s_1^k \ldots s_{i_k}^k \lhd \Box \cdots$$

The reading head may write a symbol, move left, move right.

An instruction is of the following three forms:

$$q_i s_j s_k q_l$$
$$q_i s_j L q_l$$
$$q_i s_j R q_l$$

Notice that there are no instructions of the form $q_H s_j s_k q_l$.

An Example

Suppose a Turing machine M makes use of the alphabet $\{0,1\} \cup \{\triangleright, \Box, \triangleleft\}$.

$q_S \triangleright R q_1$
$q_1 0 R q_1$
$q_1 1 0 q_2$
$q_2 0 R q_2$
$q_2 1 R q_1$
$q_1 \triangleleft L q_3$
$q_2 \triangleleft L q_3$
$q_3 0 L q_3$
$q_3 1 L q_3$
$q_3 \triangleright R q_H$

						State
▷	1	1	0	1	◁	$\widehat{q_S}$
▷	**1**	1	0	1	◁	$\widehat{q_1}$
▷	**0**	1	0	1	◁	$\widehat{q_2}$
▷	0	**1**	0	1	◁	$\widehat{q_2}$
▷	0	1	**0**	1	◁	$\widehat{q_1}$
▷	0	1	0	**1**	◁	$\widehat{q_1}$
▷	0	1	0	**0**	◁	$\widehat{q_2}$
▷	0	1	0	0	**◁**	$\widehat{q_2}$
▷	0	1	0	**0**	◁	$\widehat{q_3}$
▷	0	1	**0**	0	◁	$\widehat{q_3}$
▷	0	1	0	0	◁	$\widehat{q_3}$
▷	**0**	1	0	0	◁	$\widehat{q_H}$

Turing-Computable Function

The partial recursive function $f(x)$ computed by M is

$$f(n) = \begin{cases} m, & m \text{ is the number of } 1\text{'s between } \triangleright \text{ and } \triangleleft, \\ & \text{if } M \text{ stops when the input number is } n; \\ \uparrow, & \text{otherwise.} \end{cases}$$

A Turing-Computable Function

The function $x + y$ is Turing-Computable by:

$q_S \triangleright R q_1$
$q_1 1 B q_1$
$q_1 B R q_2$
$q_2 1 B q_3$
$q_2 B R q_2$
$q_3 1 R q_3$
$q_3 B R q_3$
$q_3 \triangleleft L q_H$

Multi-Tape Turing Machine

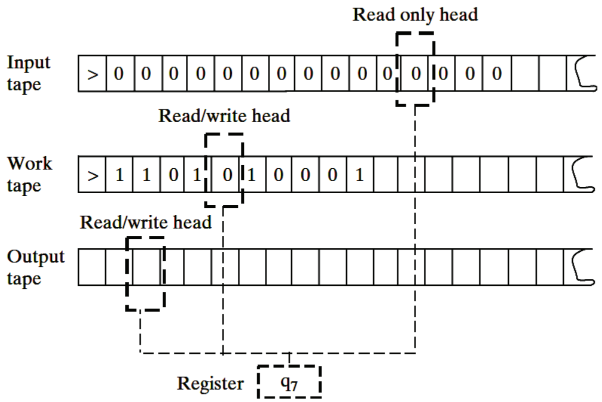
A multi-tape TM is described by a tuple (Γ, Q, δ) containing

- A finite set Γ called alphabet, of symbols. It contains a blank symbol $\Box$, a start symbol $\triangleright$, and the digits 0 and 1.
- A finite set Q of states. It contains a start state q_{start} and a halting state q_{halt}.
- A transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times L, S, R^k$, describing the rules of each computation step.

Example: A 2-Tape TM will have transition function (also named as specification) like follows:

$$\begin{array}{lcl} \langle q_s, \triangleright, \triangleright \rangle & \to & \langle q_1, \triangleright, R, R \rangle \\ \langle q_1, 0, 1 \rangle & \to & \langle q_2, 0, S, L \rangle \end{array}$$

Computation and Configuration



Computation, configuration, initial/final configuration

A 3-Tape TM for the Palindrome Problem

A palindrome is a word that reads the same both forwards and backwards. For instance:

ada, anna, madam, and nitalarbralatin.

Requirement: Give the specification of M with $k = 3$ to recognize palindromes on symbol set $\{0, 1, \rhd, \lhd, \Box\}$.

Preparation

To recognize palindrome we need to check the input string, output 1 if the string is a palindrome, and 0 otherwise.

Initially the input string is located on the first tape like "$\rhd$ 0110001 $\lhd \Box\Box\Box \cdots$", strings on all other tapes are "$\rhd\Box\Box\Box \cdots$".

The head on each tape points the first symbol "$\rhd$" as the starting state, with state mark q_S.

In the final state q_F, the output of the k^{th} tape should be "$\rhd$ 1 $\lhd \Box$" if the input is a palindrome, and "$\rhd$ 0 $\lhd \Box$" otherwise.

A 3-Tape TM for the Palindrome Problem

$Q = \{q_s, q_h, q_c, q_l, q_t, q_r\}$; $\Gamma = \{\Box, \rhd, \lhd, 0, 1\}$; two work tapes.

Start State:

$\langle q_s, \rhd, \rhd, \rhd\rangle \to \langle q_c, \rhd, \rhd, R, R, R\rangle$

Begin to copy:

$\langle q_c, 0, \Box, \Box\rangle \to \langle q_c, 0, \Box, R, R, S\rangle$
$\langle q_c, 1, \Box, \Box\rangle \to \langle q_c, 1, \Box, R, R, S\rangle$
$\langle q_c, \lhd, \Box, \Box\rangle \to \langle q_l, \Box, \Box, L, S, S\rangle$

Return back to the leftmost:

$\langle q_l, 0, \Box, \Box\rangle \to \langle q_l, \Box, \Box, L, S, S\rangle$
$\langle q_l, 1, \Box, \Box\rangle \to \langle q_l, \Box, \Box, L, S, S\rangle$
$\langle q_l, \rhd, \Box, \Box\rangle \to \langle q_t, \Box, \Box, R, L, S\rangle$

Begin to compare:

$\langle q_t, \lhd, \rhd, \Box\rangle \to \langle q_r, \rhd, 1, S, S, R\rangle$
$\langle q_t, 0, 1, \Box\rangle \to \langle q_r, 1, 0, S, S, R\rangle$
$\langle q_t, 1, 0, \Box\rangle \to \langle q_r, 0, 0, S, S, R\rangle$
$\langle q_t, 0, 0, \Box\rangle \to \langle q_t, 0, \Box, R, L, S\rangle$
$\langle q_t, 1, 1, \Box\rangle \to \langle q_t, 1, \Box, R, L, S\rangle$

Ready to terminate:

$\langle q_r, \lhd, \rhd, \Box\rangle \to \langle q_h, \rhd, \lhd, S, S, S\rangle$
$\langle q_r, 0, 1, \Box\rangle \to \langle q_h, 1, \lhd, S, S, S\rangle$
$\langle q_r, 1, 0, \Box\rangle \to \langle q_r, 0, \lhd, S, S, S\rangle$

Language System

Let $\Sigma = \{a_1, \ldots, a_k\}$ be the set of symbols, called alphabet.

A string (word) from Σ is a sequence $a_{i_1}, \cdots, a_{i_n}$ of symbols from Σ.

Σ^* is the set of all words/strings from Σ. (Kleene Star)

For example, if $\Sigma = \{a, b\}$, we have

$$\Sigma^* = \{a, b\}^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \ldots\}.$$

Λ is the empty string, that has no symbols. (ε)

$\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Fact: If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ by a TM M using the alphabet set Γ, then it is computable in time $4 \log |\Gamma| T(n)$ by a TM $\widetilde{M}$ using the alphabet $\{0, 1, \square, \triangleright\}$.

$\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Suppose M has k tapes with the alphabet Γ.

A symbol of M is encoded in $\widetilde{M}$ by a string $\sigma \in \{0,1\}^*$ of length $\log |\Gamma|$.

A state q in M is turned into a number of states in $\widetilde{M}$

- q,
- $\langle q, \sigma_1^1, \ldots, \sigma_1^k \rangle$ where $|\sigma_1^1| = \ldots = |\sigma_1^k| = 1$,
- $\cdots$,
- $\langle q, \sigma_{\log|\Gamma|}^1, \ldots, \sigma_{\log|\Gamma|}^k \rangle$, the size of $\sigma_{\log|\Gamma|}^1, \ldots, \sigma_{\log|\Gamma|}^k$ is $\log |\Gamma|$.

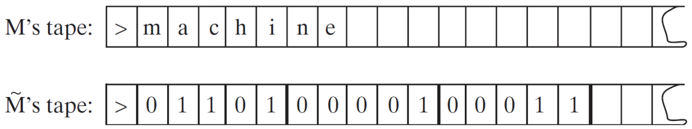
$\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

To simulate one step of M, the machine $\widetilde{M}$ will

1. use $\log |\Gamma|$ steps to read from each tape the $\log |\Gamma|$ bits encoding a symbol of Γ,
2. use its state register to store the symbols read,
3. use M's transition function to compute the symbols M writes and M's new state given this information,
4. store this information in its state register, and
5. use $\log |\Gamma|$ steps to write the encodings of these symbols on its tapes.

$\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Example: $\{0, 1, \square, \triangleright\}$ vs. English Alphabets



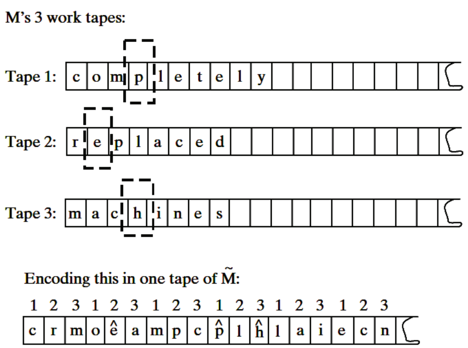
Single-Tape vs. Multi-Tape

Define a single-tape TM to be a TM that has one read-write tape.

Fact: If $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is computable in time $T(n)$ by a TM M using k tapes, then it is computable in time $5kT(n)^2$ by a single-tape TM $\widetilde{M}$.

Single-Tape vs. Multi-Tape

- The basic idea is to interleave k tapes into one tape.
- The first $n + 1$ cells are reserved for the input.



- Every symbol a of M is turned into two symbols $a, \hat{a}$ in $\widetilde{M}$, with $\hat{a}$ used to indicate head position.

Single-Tape vs. Multi-Tape

The outline of the algorithm:

The machine $\widetilde{M}$ places $\triangleright$ after the input string and then starts copying the input bits to the imaginary input tape. During this process whenever an input symbol is copied it is overwritten by $\triangleright$.

$\widetilde{M}$ marks the $n+2$-cell, ..., the $n+k$-cell to indicate the initial head positions.

$\widetilde{M}$ Sweeps $kT(n)$ cells from the $(n+1)$-th cell to right, recording in the register the k symbols marked with the hat $\hat{}$.

$\widetilde{M}$ Sweeps $kT(n)$ cells from right to left to update using the transitions of M. Whenever it comes across a symbol with hat, it moves right k cells, and then moves left to update.

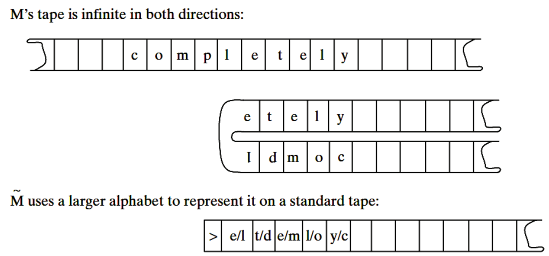
Unidirectional Tape vs. Bidirectional Tape

Define a bidirectional Turing Machine to be a TM whose tapes are infinite in both directions.

Fact: If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ by a bidirectional TM M, then it is computable in time $4T(n)$ by a TM $\widetilde{M}$ with one-directional tape.

Unidirectional Tape vs. Bidirectional Tape

- The idea is that $\widetilde{M}$ makes use of the alphabet $\Gamma \times \Gamma$.



- Every state q of M is turned into $\overline{q}$ and $\underline{q}$.

Unidirectional Tape vs. Bidirectional Tape

Let H range over $\{L, S, R\}$ and let $-H$ be defined by

$$-H = \begin{cases} R, & \text{if} \quad H = L, \\ S, & \text{if} \quad H = S, \\ L, & \text{if} \quad H = R. \end{cases}$$

$\widetilde{M}$ contains the following transitions:

$\langle \overline{q}, (\rhd, \rhd) \rangle \to \langle \underline{q}, (\rhd, \rhd), R \rangle$
$\langle \underline{q}, (\rhd, \rhd) \rangle \to \langle \overline{q}, (\rhd, \rhd), R \rangle$

$\langle \overline{q}, (a, b) \rangle \to \langle \overline{q'}, (a', b), H \rangle$ if $\langle q, a \rangle \to \langle q', a', H \rangle$
$\langle \underline{q}, (a, b) \rangle \to \langle \underline{q'}, (a, b'), -H \rangle$ if $\langle q, b \rangle \to \langle q', b', H \rangle$

Turing-Computability

Let $\mathscr{TC}$ be the set of Turing computable functions.

Theorem. $\mathscr{R} = \mathscr{TC} = \mathscr{C}$.

Proof. The proof of the inclusion $\mathscr{TC} \subseteq \mathscr{R}$ is similar to the proof of $\mathscr{C} \subseteq \mathscr{R}$. There could be many ways to show that $\mathscr{R} \subseteq \mathscr{TC}$.

Computability on Domains other than $\mathbb{N}$

URM that handle integers. We need a subtraction instruction.

(1). Each register contains an integer;

(2). There is an additional instruction $S^{-}(n)$ for each $n = 1, 2, 3, \cdots$ that has the effect of *subtracting* 1 from the contents of register R_n.

Alphabet Domain

Let $\Sigma = \{a_1, \ldots, a_k\}$ be the set of symbols, called alphabet.

Σ^* is the set of words/strings.

Λ is the empty string.

$\sigma\tau$ is the concatenation of σ and τ.

Computability on Alphabet Domain

Suppose $\Sigma = \{a, b\}$. The set $\mathscr{R}^\Sigma$ of partial recursive functions on Σ^* is the smallest set that satisfies the following properties:

- It contains the following basic functions:

$$\begin{aligned} f(\sigma) &= \Lambda, \\ f(\sigma) &= \sigma a, \\ f(\sigma) &= \sigma b, \\ U_i^n(\sigma_1, \ldots, \sigma_n) &= \sigma_i. \end{aligned}$$

- $\mathscr{R}^\Sigma$ is closed under substitution.

Computability on Alphabet Domain

- $\mathscr{R}^\Sigma$ is closed under recursion:

$$\begin{aligned} h(\sigma, \Lambda) &\simeq f(\sigma), \\ h(\sigma, \tau a) &\simeq g_1(\sigma, \tau, h(\sigma, \tau)), \\ h(\sigma, \tau b) &\simeq g_2(\sigma, \tau, h(\sigma, \tau)). \end{aligned}$$

- $\mathscr{R}^\Sigma$ is closed under minimalisation:

$$h(\sigma) \simeq \mu\tau(f(\sigma, \tau) = \Lambda).$$

Here $\mu\tau$ means the first τ in the natural ordering Λ, a, b, aa, ab, ba, bb, aaa, aab, aba, $\cdots$

Two Questions

1. How do different models of computation compare to each other?

2. How do these models characterize the informal notion of effective computability?

Other Approaches to Computability

1. Gödel-Kleene (1936): Partial recursive functions.
2. Turing (1936): Turing machines.
3. Church (1936): λ-terms.
4. Post (1943): Post systems.
5. Markov (1951): Variants of the Post systems.
6. Shepherdson-Sturgis (1963): URM-computable functions.

Fundamental Result: Each of the above proposals for a characterization of the notion of effective computability gives rise to the same class of functions.

Church-Turing Thesis

Question: How well is the informal intuitive idea of effectively computable function captured by the various formal characterizations?

Church-Turing Thesis.
The intuitively and informally defined class of effectively computable partial functions coincides exactly with the class $\mathscr{C}$ of URM-computable functions.

The functions definable in all computation models are the same. They are precisely the computable functions.

It was called Church Thesis by Kleene. Gödel accepted it only after he saw Turing's equivalence proof.

Church-Turing Thesis

Church-Turing thesis is not a *theorem*, but it has the status of a *claim* or *belief* which must be substantiated by evidence.

Evidence:

- ▷ The Fundamental result: many independent proposals for a precise formulation of the intuitive idea have led to the same class of functions $\mathscr{C}$.
- ▷ A vast collection of effectively computable functions has been shown explicitly to belong to $\mathscr{C}$.
- ▷ The implementation of a program P on the URM to compute a function is an example of an algorithm. Thus all functions in $\mathscr{C}$ are computable in the informal sense.
- ▷ No one has ever found a function that would be accepted as computable in the informal sense, that does not belong to $\mathscr{C}$.

Church-Turing Thesis

No one has come up with an intuitively computable function that is not recursive.

When you are convincing people of the computability of your functions, you are constructing an interpretation from your model to a well-known model.

Church-Turing Thesis is universally accepted. It allows us to give an informal argument for the computability of a function.

We can make use of a computable function without explicitly defining it.

How to prove the computability of a function f?

There are two methods open to us:

- Write a program that URM-computes f or prove by indirect means that such a program exists.
- Give an informal (though rigorous) proof that given informal algorithm is indeed an algorithm that serves to compute f, then appeal Church's thesis and conclude that f is URM-computable. (proof by church's thesis).

Example 1

Let P be a URM program; define a function f by

$$f(x,y,t) = \begin{cases} 1 & \text{if } P(x)\downarrow y \text{ after } t \text{ or fewer steps} \\ & \text{of the computation } P(x); \\ 0 & \text{otherwise.} \end{cases}$$

Prove the computability of f.

Informal Algorithm

Given (x,y,t), simulate the computation $P(x)$: carrying out t steps of $P(x)$ unless this computation stops after fewer than t steps.

If $P(x)$ stops after t or fewer steps, with y finally in R_1, then $f(x,y,t)=1$.

Otherwise ($P(x)$ stops in t or fewer steps with some number other than y in R_1, or if $P(x)$ has not stopped after t steps), we have $f(x,y,t)=0$.

Analysis

Simulation of $P(x)$ for at most t steps is clearly a mechanical procedure, which can be completed in a finite amount of time.

Thus, f is effectively computable.

Hence, by Church's Thesis, f is URM-computable.

Example 2

Suppose that f and g are unary effectively computable functions.

$$h(x) = \begin{cases} 1 & \text{if } x \in Dom(f) \text{ or } x \in Dom(g); \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Prove the computability of h.

Informal Algorithm

Given x, start the algorithms for computing $f(x)$ and $g(x)$ simultaneously. If and when one of these computations terminates, then stop altogether, and set $h(x) = 1$.

Otherwise, continue indefinitely.

Analysis

This algorithm gives $h(x) = 1$ for any x such that either $f(x)$ or $g(x)$ is defined; and it goes on for ever if neither is defined.

Thus, we have an algorithm for computing h, and hence, by Church's Thesis, h is URM-computable.

Example 3

Let $f(n)$ = the nth digit in the decimal expansion of π.

Prove the computability of f.

(So we have $f(0) = 3, f(1) = 1, f(2) = 4$, etc.)

Proof

We can obtain an informal algorithm for computing $f(n)$ as follows. Consider Hutton's series for π:

$$\begin{aligned}
\pi &= \frac{12}{5}\left\{1+\frac{2}{3}\left(\frac{1}{10}\right)+\frac{2\cdot 4}{3\cdot 5}\left(\frac{1}{10}\right)^2+\cdots\right\} \\
&\quad +\frac{14}{25}\left\{1+\frac{2}{3}\left(\frac{1}{50}\right)+\frac{2\cdot 4}{3\cdot 5}\left(\frac{1}{50}\right)^2+\cdots\right\} \\
&= \sum_{n=0}^{\infty}\frac{(n!2^n)^2}{(2n+1)!}\left\{\frac{12}{5}\left(\frac{1}{10}\right)^n+\frac{14}{25}\left(\frac{1}{50}\right)^n\right\} \\
&= \sum_{n=0}^{\infty} h_n \text{ (defined as)}
\end{aligned}$$

Proof (Cont.)

Let $s_k = \sum_{n=0}^{k} h_n$, by theory of infinite series $s_k < \pi < s_k + \frac{1}{10^k}$.

Since s_k is rational, the decimal expansion of s_k can be effectively calculated to any desired number of places using long division.

Thus the effective method for calculating $f(n)$ (given a number n) can be described as:

Find the first $N \geq n+1$ such that the decimal expansion $s_N = a_0.a_1a_2\cdots a_na_{n+1}\cdots a_N\cdots$ does not have all of $a_{n+1}\cdots a_N$ equal to 9. Then put $f(n) = a_n$.

Note: Such an N exists, for otherwise the decimal expansion of π would end in recurring 9, making π rational.

Proof (Cont.)

To see that this gives the required value, suppose that $a_m \neq 9$ with $n < m \leq N$. Then by the above

$$s_N < \pi < s_N + \frac{1}{10^N} \leq s_N + \frac{1}{10^m}.$$

Hence $a_0.a_1\cdots a_n\cdots a_m\cdots < \pi < a_0.a_1\cdots a_n\cdots(a_m+1)\cdots$. So the nth decimal place of π is indeed a_n.

Thus by Church's Thesis, f is computable.