VI. Church-Turing Thesis

Yuxi Fu

BASICS, Shanghai Jiao Tong University

Fundamental Question

How do computation models characterize the informal notion of effective computability?

Fundamental Result

Theorem. The set of functions definable in λ-Calculus (Turing Machine Model, Unlimited Random Access Machine Model) is precisely the set of recursive functions.

Proof.

We have already showed that
μ-definable $\Rightarrow$ λ-definable $\Rightarrow$ Turing definable $\Rightarrow$ URM-definable.

We have to show that URM-definable $\Rightarrow$ μ-definable. □

Synopsis

1. Gödel Encoding
2. Kleene's Proof
3. Church-Turing Thesis

Gödel Encoding

Godel's Insight

The set of syntactical objects of a formal system is denumerable.

More importantly, every syntactical object can be coded up effectively by a number in such a way that a unique syntactical object can be recovered from the number.

This is the crucial technique Gödel used in his proof of the Incompleteness Theorem.

Enumeration

An enumeration of a set X is a surjection $g : \omega \rightarrow X$;
this is often represented by writing $\{x_0, x_1, x_2, \ldots\}$.

It is an enumeration without repetition if g is injective.

Denumeration

A set X is denumerable if there is a bijection $f : X \to \omega$.
(denumerate = denote + enumerate)

Let X be a set of "finite objects".

Then X is effectively denumerable if there is a bijection $f : X \to \omega$ such that both f and f^{-1} are computable.

Encoding Pair

Fact. $\omega \times \omega$ is effectively denumerable.

Proof.

A bijection $\pi : \omega \times \omega \to \omega$ is defined by

$$\pi(m, n) \stackrel{\text{def}}{=} 2^m(2n+1) - 1,$$
$$\pi^{-1}(l) \stackrel{\text{def}}{=} (\pi_1(l), \pi_2(l)),$$

where

$$\pi_1(x) \stackrel{\text{def}}{=} (x+1)_1,$$
$$\pi_2(x) \stackrel{\text{def}}{=} ((x+1)/2^{\pi_1(x)} - 1)/2.$$

□

Encoding Tuple

Fact. $\omega^+ \times \omega^+ \times \omega^+$ is effectively denumerable.

Proof.

A bijection $\zeta : \omega^+ \times \omega^+ \times \omega^+ \to \omega$ is defined by

$$\zeta(m, n, q) \stackrel{\text{def}}{=} \pi(\pi(m-1, n-1), q-1),$$
$$\zeta^{-1}(l) \stackrel{\text{def}}{=} (\pi_1(\pi_1(l)) + 1, \pi_2(\pi_1(l)) + 1, \pi_2(l) + 1).$$

□

Encoding Finite String

Fact. $\bigcup_{k>0} \omega^k$ is effectively denumerable.

Proof.

A bijection $\tau : \bigcup_{k>0} \omega^k \to \omega$ is defined by

$$\begin{aligned}\tau(a_1, \ldots, a_k) \ &\stackrel{\text{def}}{=}\ 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \ldots \\ &\quad + 2^{a_1+a_2+a_3+\ldots,a_k+k-1} - 1.\end{aligned}$$

Now given x it is easy to find $b_1 < b_2 < \ldots < b_k$ such that

$$2^{b_1} + 2^{b_2} + 2^{b_3} + \ldots + 2^{b_k} = x + 1.$$

It is then clear how to calculate $a_1, a_2, a_3, \ldots, a_k$. Details are next. □

Encoding Finite String

A number $x \in \omega$ has a unique expression as

$$x = \sum_{i=0}^{\infty} \alpha_i 2^i,$$

where α_i is either 0 or 1 for all $i \geq 0$.

1. The function $\alpha(i, x) = \alpha_i$ is primitive recursive:

$$\alpha(i, x) = \mathrm{rm}(2, \mathrm{qt}(2^i, x)).$$

2. The function $\ell(x) = \mathit{if}\, x > 0\ \mathit{then}\ k\ \mathit{else}\, 0$ is primitive recursive:

$$\ell(x) = \sum_{i<x} \alpha(i, x).$$

Encoding Finite String

3. If $x > 0$ then it has a unique expression as

$$x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_k},$$

where $1 \leq k$ and $0 \leq b_1 < b_2 < \ldots < b_k$.

The function $\mathrm{b}(i, x) = \mathit{if}\,(x > 0) \wedge (1 \leq i \leq \ell(x))\ \mathit{then}\ b_i\ \mathit{else}\ 0$ is primitive recursive:

$$\mathrm{b}(i, x) = \begin{cases} \mu y{<}x \left(\sum_{k \leq y} \alpha(k, x) = i \right), & \text{if } (x > 0) \wedge (1 \leq i \leq \ell(x)); \\ 0, & \text{otherwise.} \end{cases}$$

Encoding Finite String

4. If $x > 0$ then it has a unique expression as

$$x = 2^{a_1} + 2^{a_1+a_2+1} + \ldots + 2^{a_l+a_2+\ldots+a_k+k-1}.$$

The function $\mathsf{a}(i, x) = a_i$ is primitive recursive:

$$\begin{array}{rcl} \mathsf{a}(i, x) & = & \mathsf{b}(i, x), \text{ if } i = 0 \text{ or } i = 1, \\ \mathsf{a}(i+1, x) & = & (\mathsf{b}(i+1, x) \dot{-} \mathsf{b}(i, x)) \dot{-} 1, \text{ if } i \geq 1. \end{array}$$

We conclude that $a_1, a_2, a_3, \ldots, a_k$ can be calculated by primitive recursive functions.

Encoding Programme

Let $\mathcal{I}$ be the set of all instructions.

Let $\mathcal{P}$ be the set of all programs.

The objects in $\mathcal{I}$, and $\mathcal{P}$ as well, are 'finite objects'.

Encoding Programme

Theorem. $\mathcal{I}$ is effectively denumerable.

Proof.

The bijection $\beta : \mathcal{I} \rightarrow \omega$ is defined as follows:

$$\begin{aligned}
\beta(Z(n)) &= 4(n-1), \\
\beta(S(n)) &= 4(n-1)+1, \\
\beta(T(m,n)) &= 4\pi(m-1,n-1)+2, \\
\beta(J(m,n,q)) &= 4\zeta(m,n,q)+3.
\end{aligned}$$

The converse β^{-1} is easy. □

Encoding Programme

Theorem. $\mathcal{P}$ is effectively denumerable.

Proof.

The bijection $\gamma : \mathcal{P} \to \omega$ is defined as follows:

$$\gamma(P) \;\;=\;\; \tau(\beta(I_1), \ldots, \beta(I_s)),$$

assuming $P = I_1, \ldots, I_s$.

The converse γ^{-1} is obvious. □

Gödel Number of Programme

The value $\gamma(P)$ is called the Gödel number of P.

$$\begin{aligned} P_n &= \text{the programme with Godel index } n \\ &= \gamma^{-1}(n) \end{aligned}$$

We shall fix this particular encoding function γ throughout.

Example

Let P be the program $T(1,3), S(4), Z(6)$.

$\beta(T(1,3)) = 18$, $\beta(S(4)) = 13$, $\beta(Z(6)) = 20$.

$\gamma(P) = 2^{18} + 2^{32} + 2^{53} - 1$.

Example

Consider P_{4127}.

$4127 = 2^5 + 2^{12} - 1$.

$\beta(I_1) = 4 + 1$, $\beta(I_2) = 4\pi(1, 0) + 2$.

So P_{4127} is $S(2); T(2, 1)$.

Kleene's Proof

Kleene demonstrated how to prove that machine computable functions are recursive functions.

Proof in Detail

The state of the computation of the program $P_e(\widetilde{x})$ can be described by a configuration and an instruction number.

A state can be coded up by the number

$$\sigma = \pi(c, j),$$

where c is the configuration that codes up the current values in the registers

$$c = 2^{r_1} 3^{r_2} \ldots = \prod_{i \geq 1} p_i^{r_i},$$

and j is the next instruction number.

Proof in Detail

To describe the changes of the states of $P_e(\widetilde{x})$, we introduce three $(n+2)$-ary functions:

$$\begin{aligned}
\mathsf{c}_n(e,\widetilde{x},t) &= \text{the configuration after } t \text{ steps of } P_e(\widetilde{x}), \\
\mathsf{j}_n(e,\widetilde{x},t) &= \text{the number of the next instruction after } t \text{ steps} \\
&\quad\ \text{of } P_e(\widetilde{x}) \text{ (it is 0 if } P_e(\widetilde{x}) \text{ stops in } t \text{ or less steps)}, \\
\sigma_n(e,\widetilde{x},t) &= \pi(\mathsf{c}_n(e,\widetilde{x},t),\mathsf{j}_n(e,\widetilde{x},t)).
\end{aligned}$$

If σ_n is primitive recursive, then $\mathsf{c}_n, \mathsf{j}_n$ are primitive recursive.

Proof in Detail

If the computation of $P_e(\widetilde{x})$ stops, it does so in

$$\mu t(\mathrm{j}_n(e, \widetilde{x}, t) = 0)$$

steps. Then the final configuration is

$$\mathrm{c}_n(e, \widetilde{x}, \mu t(\mathrm{j}_n(e, \widetilde{x}, t) = 0)).$$

We conclude that the value of the computation $P_e(\widetilde{x})$ is

$$(\mathrm{c}_n(e, \widetilde{x}, \mu t(\mathrm{j}_n(e, \widetilde{x}, t) = 0)))_1.$$

Proof in Detail

The function σ_n can be defined as follows:

$$\begin{aligned}\sigma_n(e, \widetilde{x}, 0) &= \pi(2^{x_1}3^{x_2}\ldots p_n^{x_n}, 1),\\ \sigma_n(e, \widetilde{x}, t+1) &= \pi(\mathrm{config}(e, \sigma_n(e, \widetilde{x}, t)), \mathrm{next}(e, \sigma_n(e, \widetilde{x}, t))),\end{aligned}$$

where

- $\mathrm{config}(e, \pi(c, j))$ is the configuration after $t+1$ steps;
- $\mathrm{next}(e, \pi(c, j))$ is the new number after $t+1$ steps.

Proof in Detail

$$\mathsf{ln}(e) \;=\; \text{the number of instructions in } P_e;$$

$$\mathsf{gn}(e,j) \;=\; \begin{cases} \text{the code of } I_j \text{ in } P_e, & \text{if } 1 \leq j \leq \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

Both functions are primitive recursive since

$$\begin{aligned} \mathsf{ln}(e) &= \ell(e+1), \\ \mathsf{gn}(e,j) &= \mathsf{a}(j, e+1). \end{aligned}$$

Proof in Detail

$\mathrm{u}(z) = m$ whenever $z = \beta(Z(m))$ or $z = \beta(S(m))$:

$$\mathrm{u}(z) = \mathrm{qt}(4, z) + 1.$$

$\mathrm{u}_1(z) = m_1$ and $\mathrm{u}_2(z) = m_2$ whenever $z = \beta(T(m_1, m_2))$:

$$\begin{aligned} \mathrm{u}_1(z) &= \pi_1(\mathrm{qt}(4, z)) + 1, \\ \mathrm{u}_2(z) &= \pi_2(\mathrm{qt}(4, z)) + 1. \end{aligned}$$

$\mathrm{v}_1(z) = m_1$ and $\mathrm{v}_2(z) = m_2$ and $\mathrm{v}_3(z) = q$ if $z = \beta(J(m_1, m_2, q))$:

$$\begin{aligned} \mathrm{v}_1(z) &= \pi_1(\pi_1(\mathrm{qt}(4, z))) + 1, \\ \mathrm{v}_2(z) &= \pi_2(\pi_1(\mathrm{qt}(4, z))) + 1, \\ \mathrm{v}_3(z) &= \pi_2(\mathrm{qt}(4, z)) + 1. \end{aligned}$$

Proof in Detail

The change in the configuration c effected by instruction $Z(m)$:

$$\text{zero}(c, m) = \text{qt}(p_m^{(c)_m}, c).$$

The change in the configuration c effected by instruction $S(m)$:

$$\text{succ}(c, m) = p_m c.$$

The change in the configuration c effected by instruction $T(m, n)$:

$$\text{tran}(c, m, n) = \text{qt}(p_n^{(c)_n}, p_n^{(c)_m} c).$$

Proof in Detail

The following function

$$\mathrm{ch}(c,z) \;=\; \text{the resulting configuration when the configuration } c \text{ is operated on by the instruction with code number } z.$$

is primitive recursive since

$$\mathrm{ch}(c,z) = \begin{cases} \mathrm{zero}(c,\mathrm{u}(z)), & \text{if } \mathrm{rm}(4,z)=0, \\ \mathrm{succ}(c,\mathrm{u}(z)), & \text{if } \mathrm{rm}(4,z)=1, \\ \mathrm{tran}(c,\mathrm{u}_1(z),\mathrm{u}_2(z)), & \text{if } \mathrm{rm}(4,z)=2, \\ c, & \text{if } \mathrm{rm}(4,z)=3. \end{cases}$$

Proof in Detail

The following function

$$\mathsf{v}(c,j,z) \;=\; \begin{cases} \text{the number } j' \text{ of the next instruction} & \\ \text{when the configuration } c \text{ is operated} & \text{if } j>0, \\ \text{on by the } j\text{th instruction with code } z, & \\ 0, & \text{if } j=0. \end{cases}$$

is primitive recursive since

$$\mathsf{v}(c,j,z) \;=\; \begin{cases} j+1, & \text{if } \mathsf{rm}(4,z)\neq 3, \\ j+1, & \text{if } \mathsf{rm}(4,z)=3 \;\wedge\; (c)_{\mathsf{v}_1(z)} \neq (c)_{\mathsf{v}_2(z)}, \\ \mathsf{v}_3(z), & \text{if } \mathsf{rm}(4,z)=3 \;\wedge\; (c)_{\mathsf{v}_1(z)} = (c)_{\mathsf{v}_2(z)}. \end{cases}$$

Proof in Detail

$$\mathsf{config}(e,\sigma) = \begin{cases} \mathsf{ch}(\pi_1(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \le \pi_2(\sigma) \le \mathsf{ln}(e), \\ \pi_1(\sigma), & \text{otherwise.} \end{cases}$$

$$\mathsf{next}(e,\sigma) = \begin{cases} \mathsf{v}(\pi_1(\sigma), \pi_2(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \le \pi_2(\sigma) \le \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

Proof in Detail

We conclude that the functions $\mathsf{c}_n, \mathsf{j}_n, \sigma_n$ are primitive recursive.

Further Constructions

For each $n \geq 1$, the following predicates are primitive recursive:

1. $\mathsf{S}_n(e, \widetilde{x}, y, t) \stackrel{\text{def}}{=}$ '$P_e(\widetilde{x}) \downarrow y$ in t or fewer steps'.
2. $\mathsf{H}_n(e, \widetilde{x}, t) \stackrel{\text{def}}{=}$ '$P_e(\widetilde{x}) \downarrow$ in t or fewer steps'.

They are defined by

$$\begin{aligned} \mathsf{S}_n(e, \widetilde{x}, y, t) &\stackrel{\text{def}}{=} \mathsf{j}_n(e, \widetilde{x}, t) = 0 \wedge (\mathsf{c}_n(e, \widetilde{x}, t))_1 = y, \\ \mathsf{H}_n(e, \widetilde{x}, t) &\stackrel{\text{def}}{=} \mathsf{j}_n(e, \widetilde{x}, t) = 0. \end{aligned}$$

Kleene's Normal Form Theorem

Let $\phi_e^{(n)}$ denote the n-ary function computed by P_e.

Theorem. (Kleene)

There is a primitive recursive function $\mathrm{U}(x)$ and, for each $n \geq 1$, a primitive recursive predicate $\mathrm{T}_n(e, \widetilde{x}, z)$ such that

1. $\phi_e^{(n)}(\widetilde{x})$ is defined if and only if $\exists z.\mathrm{T}_n(e, \widetilde{x}, z)$.
2. $\phi_e^{(n)}(\widetilde{x}) \simeq \mathrm{U}(\mu z T_n(e, \widetilde{x}, z))$.

Proof.

(1) $\mathrm{T}_n(e, \widetilde{x}, z) = S_n(e, \widetilde{x}, \pi_1(z), \pi_2(z))$.

(2) Let $\mathrm{U}(x) = \pi_1(x)$. Then $\phi_e^{(n)}(\widetilde{x}) \simeq \mathrm{U}(\mu z.\mathrm{T}_n(e, \widetilde{x}, z))$. □

Every computable function can be obtained from a primitive recursive function by using at most one application of the μ-operator in a standard manner.

Church-Turing Thesis

Church-Turing Thesis.

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

- Church believed that all computable functions are λ-definable.
- Kleene termed it Church Thesis.
- Gödel accepted it only after he saw Turing's equivalence proof.
- Church-Turing Thesis is now universally accepted.

Computable Function

Let $\mathcal{C}$ be the set of all computable functions.

Let $\mathcal{C}_n$ be the set of all n-ary computable functions.

Power of Church-Turing Thesis

Noone has come up with a computable function that is not in $\mathcal{C}$.

When you are convincing people of your model of computation, you are constructing an effective translation from your model to a well-known computation model.

Making Use of Church-Turing Thesis

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

We will make use of CTT in this way without explicitly defining it.

Comment on Church-Turing Thesis

CTT and Physical Implementation

- Deterministic Turing Machines are physically implementable. This is the well-known von Neumann Architecture.
- Are quantum computers physically implementable? Can a quantum computer compute more or more efficiently?

CTT, is it a Law of Nature or a Wisdom of Human?