7. The Halting Problem; The Universal Function Theorem

7.1 The Church-Turing Thesis

The ***Church-Turing Thesis*** (***CT***), formulated in terms of $\mathcal{G}$-computability, states that:

> *Any function which is computable by any algorithm whatsoever, is computable by a $\mathcal{G}$-program.*

This thesis was first formulated in the 1930's, independently by ***Church***, using the formalism of the ***$\boldsymbol{\lambda}$-calculus***, and ***Turing***, using the formalism of ***Turing machines***.

CT ***cannot be mathematically proved*** since it uses the non-mathematical notion of "***algorithm***". Its acceptance is based on three arguments:

(1) The philosophical analysis by Turing of the notion of "algorithm".

(2) Many attempted formalisms of the notion of "algorithm" have been found to be equivalent, e.g.,

- Turing machine computability,
- λ-computability,
- $\mathcal{G}$-computability,
- C-computability, etc.

(3) No counterexample to CT has been found in over 80 years.

Clearly, by CT

$$\mathcal{G}\text{-COMP} \;=\; \text{EFF}.$$

Similarly, we can formulate a ***relativised version of CT (Rel-CT)***:

$$\mathcal{G}\text{-COMP}(\vec{g}) \;=\; \text{EFF}(\vec{g}).$$

(see diagrams pp. 4-4, 4-7). The collection [Dav65] contains famous pioneering papers on computability theory, including those of Church and Turing, in which their respective versions of CT were first formulated and justified.

NOTE: Any theorem which requires CT in its proof will be marked with the superscript 'CT', and any proof which uses CT (even if not required) will also be so marked.

7.2 Decidability of sets and relations

Let B be an n-ary relation on $\mathbb{N}$. We say that B is

- ***primitive recursive*** (PR) iff its characteristic function χ_B is;
- ***$\mathcal{G}$-computable*** ($\mathcal{G}$-COMP) (or ***recursive***) iff χ_B is;
- ***decidable*** (DEC) or ***effective*** or ***algorithmic*** iff χ_B is.

(See p. 3-2 for notation.)

So B is ***decidable*** iff there is an ***algorithm*** to test for membership of B.

Similarly we can define ***relativised versions*** of these notions, i.e., $\mathrm{PR}(\vec{g})$, $\mathcal{G}\text{-COMP}(\vec{g})$, $\mathrm{DEC}(\vec{g})$.

Let B, C be n-ary relations on $\mathbb{N}$.

Theorem 7.1. *$B \cup C$, $B \cap C \in \mathrm{PR}(B, C)$, and $\overline{B} \in \mathrm{PR}(B)$.
Hence if $B, C \in \mathrm{PR}$, then so are $B \cup C$, $B \cap C$ and $\overline{B}$.*

Proof: Since $\chi_{B\cup C} = \chi_B \vee \chi_C$, $\chi_{B\cap C} = \chi_B \wedge \chi_C$, and $\chi_{\overline{B}} = \neg\chi_B$, the results follow from Theorem 5.3 (p. 5-5). $\square$

Corollary 7.2. *$B \cup C$, $B \cap C \in \mathcal{G}\text{-COMP}(B, C)$, and $\overline{B} \in \mathcal{G}\text{-COMP}(B)$.
Hence if $B, C \in \mathcal{G}\text{-COMP}$, then so are $B \cup C$, $B \cap C$ and $\overline{B}$.*

Proof: By Corollary 4.15 (p. 4-7). $\square$

NOTES:
- $B \cup C, B \cap C \in \mathrm{DEC}(B, C)$, and $\overline{B} \in \mathrm{DEC}(B)$.
- Intuitively, $B \cup C$ and $B \cap C$ are decidable in B, C, and $\overline{B}$ is dec. in B. Hence if B, C are decidable, then so are $B \cup C$, $B \cap C$ and $\overline{B}$.
- Clearly, $B \in \mathcal{G}\text{-COMP}(\vec{g}) \implies B \in \mathrm{DEC}(\vec{g})$.
  By ***Rel-CT***, the converse is also true, so that

$$B \in \mathcal{G}\text{-COMP}(\vec{g}) \iff B \in \mathrm{DEC}(\vec{g}).$$

Notation. $\mathcal{P}(\vec{x})\downarrow$ means $\psi_{\mathcal{P}}^{(n)}(\vec{x})\downarrow$ where $\vec{x} = (x_1, \ldots, x_n)$.
$\mathcal{P}(\vec{x})\downarrow y$ means $\psi_{\mathcal{P}}^{(n)}(\vec{x})\downarrow y$
$\mathcal{P}(\vec{x})\uparrow$ means $\psi_{\mathcal{P}}^{(n)}(\vec{x})\uparrow$.

7.3 The Halting Problem

The ***halting problem*** is the relation

$$\text{HP} \;=\; \{\, (\mathcal{P}, x) \mid \mathcal{P} \text{ halts on } x \,\} \;\subseteq\; \mathcal{G}\text{-PROG} \times \mathbb{N}.$$

Q. Is HP (***effectively***) ***solvable*** or ***decidable***?
We answer this using CT and the Gödel numbering of $\mathcal{G}$-PROG.

Let $\boldsymbol{Halt}(y, x)$ be the predicate $\text{HP}(\mathcal{P}_y, x)$, i.e.

$$\boldsymbol{Halt}(y, x) = \begin{cases} 1 & \text{if } \mathcal{P}_y \text{ halts on } x \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 7.3. ***Halt*** *is not $\mathcal{G}$-computable.*

Proof: Suppose it is. Then there exists a macro for it:

$$\boxed{\boldsymbol{Halt}(V, U)}\ .$$

Consider the program $\mathcal{P}$:

$$\boxed{[A]\ \mathsf{if}\ \boldsymbol{Halt}(X, X)\ \mathsf{goto}\ A}\ ,$$

Then $$\psi_{\mathcal{P}}(x) \;\simeq\; \begin{cases} \uparrow & \text{if } \boldsymbol{Halt}(x, x) \\ 0 & \text{otherwise.} \end{cases}$$

So for all x,

$$\mathcal{P} \text{ halts on } x \iff \neg \boldsymbol{Halt}(x, x). \tag{1}$$

Let $p = \#(\mathcal{P})$. Then from (1), for all x,

$$\boldsymbol{Halt}(p, x) \iff \neg \boldsymbol{Halt}(x, x).$$

Finally, putting $x = p$, we get

$$\boldsymbol{Halt}(p, p) \iff \neg \boldsymbol{Halt}(p, p),$$

a contradiction. $\square$

Note the use of ***diagonalisation*** or ***self-application*** or ***self-reference*** in this proof.

We now use CT to show the ***undecidability*** or ***"unsolvability" of HP***.

Theorem 7.4$^{\mathbf{CT}}$. *There is no algorithm which, when given a $\mathcal{G}$-program $\mathcal{P}$ and a number x, will determine if $\mathcal{P}$ halts on input x.*

Proof: Suppose there is such an algorithm. Then there is an algorithm which, given any y and x, determines if program $\mathcal{P}_y$ halts on input x. Hence by CT there is a $\mathcal{G}$-program which does the same, contradicting Thm 7.3. $\square$

EXERCISE: (Another version of the unsolvability of HP)

Show that the *diagonal* set below is *not* decidable:

$$\{\, x \mid \boldsymbol{Halt}(x, x) \,\} = \{\, x \mid \mathcal{P}_x(x) \downarrow \}.$$

7.4 The universal $\mathcal{G}$-program; UFT

Let us review what we have done so far.

- We have a method (GN) for uniquely and effectively associating $\mathcal{G}$-programs with numbers.
- In this way we can code $\mathcal{G}$-programs so as to use them essentially as inputs to other $\mathcal{G}$-programs, or even to themselves.
- We used this technique and CT to show that there is *no* algorithm by which we can determine whether a program $\mathcal{P}$ halts on an input x.

Now we use the GN to prove another important but positive result.

Let $\varphi_y^{(n)}$ be the n-ary function computed by program $\mathcal{P}_y$, i.e., $\varphi_y^{(n)} = \psi_{\mathcal{P}_y}^{(n)}$. Then

$$\varphi_0^{(n)}, \varphi_1^{(n)}, \varphi_2^{(n)}, \ldots$$

is a ***listing*** of $\mathcal{G}$-COMP$^{(n)}$, and y is the ***gn*** or ***index*** of $\varphi_y^{(n)}$.
We define the ((n+1)-ary) ***universal function*** $\Phi^{(n)}$ for $\mathcal{G}$-COMP$^{(n)}$ by:

$$\Phi^{(n)}(x_1, \ldots, x_n, y) \simeq \varphi_y^{(n)}(x_1, \ldots, x_n).$$

NOTE: We often drop the superscript '(n)' from Φ and φ when $n = 1$.

Theorem 7.5 (Universal function theorem (UFT) for $\mathcal{G}$-COMP). *$\Phi^{(n)} \in \mathcal{G}\text{-COMP}^{(n+1)}$. In fact, there is a universal program $\mathcal{U}_n$ for $\mathcal{G}\text{-COMP}^{(n)}$ which computes $\Phi^{(n)}$. That is, $\psi_{\mathcal{U}_n}^{(n+1)} = \Phi^{(n)}$.*

Proof 1 (using CT): Consider the following algorithm:

"With inputs $x_1, \ldots, x_n$, y :
construct the program $\mathcal{P}_y$;
apply it to inputs $x_1, \ldots, x_n$."

This provides an effective method for computing $\Phi^{(n)}(\vec{x}, y)$ for any $\vec{x}, y$. Hence by CT, $\Phi^{(n)}$ is $\mathcal{G}$-computable. □

Proof 2 (not using CT): We will actually *construct* $\mathcal{U}_n$, following [DW83]. First we make some general remarks on the construction of the program.

It will be necessary to code not only programs, but also *states*, by gn's.

For example, if $\boldsymbol{dom}(\sigma) = \{Y, X_1, X_2, Z_1\}$, and $\sigma(Y) = 0$, $\sigma(X_1) = 2$, $\sigma(X_2) = 3$, $\sigma(Z_1) = 1$ (say), then $\#(\sigma) = [0, 2, 1, 3] = p_1^0 \cdot p_2^2 \cdot p_3^1 \cdot p_4^3$.

For convenience we will use macros freely and ignore the rules for letters for variables and labels.

For each $n > 0$, $\mathcal{U}_n$ *simulates* the computation of the program numbered X_{n+1} on the input variables $X_1, \ldots, X_n$. Suppose

$$\mathcal{P} = (I_1, \ldots, I_m).$$

Then

$$X_{n+1} \;=\; \#(\mathcal{P}) \;=\; [\#(I_1), \ldots, \#(I_m)] \dot{-} 1.$$

The aux. variables Z, S, and K store the gn's of (resp.) the sequence of instructions, the current state, and the instruction about to be executed. So

- $Z = [\#(I_1), \ldots, \#(I_m)]$,
- S is initialised to $p_1^Y p_2^{X_1} p_3^{Z_1} p_4^{X_2} p_5^{Z_2} p_6^{X_3} \cdots \;=\; p_2^{X_1} p_4^{X_2} p_6^{X_3} \cdots,$
- K is initialised to 1.

Note that the input variables X_1, $X_2, \ldots$ have *even* places in the effective listing of program variables (p. 6-8), so the variables occupying the *odd* places take the value 0 at the beginning of the program.

Now, if at any stage

$$(Z)_K \;=\; \#(I_K) \;=\; \langle a, \langle b, c\rangle\rangle,$$

and we put $U = \boldsymbol{r}((Z)_K) = \langle b, c\rangle$, then, for the next instruction,

$$\begin{array}{rl} \boldsymbol{\ell}((Z)_K) = a & \text{is its label,} \\ \boldsymbol{\ell}(U) = b & \text{its type,} \\ \boldsymbol{r}(U) = c & \text{the variable involved } \ (\#(V) \dot{-} 1). \end{array}$$

The universal program $\mathcal{U}_n$ is as follows.

$$\begin{array}{ll}
 & Z \leftarrow X_{n+1} + 1 \\
 & S \leftarrow \prod_{i=1}^{n} (p_{2i})^{X_i} \\
 & K \leftarrow 1 \\
[C] & \textsf{if } K = \boldsymbol{Lt}(Z) + 1 \textsf{ goto } F \\
 & U \leftarrow \boldsymbol{r}((Z)_K) \\
 & P \leftarrow p_{\boldsymbol{r}(U)+1} \\
 & \textsf{if } \boldsymbol{\ell}(U) = 0 \textsf{ goto } N \\
 & \textsf{if } \boldsymbol{\ell}(U) = 1 \textsf{ goto } A \\
 & \textsf{if } \neg(P|S) \textsf{ goto } N \\
 & \textsf{if } \boldsymbol{\ell}(U) = 2 \textsf{ goto } M \\
 & K \leftarrow \boldsymbol{min}\ i_{0<i<\boldsymbol{Lt}(Z)+1} [\boldsymbol{\ell}((Z)_i) + 2 = \boldsymbol{\ell}(U)] \\
 & \textsf{goto } C \\
[M] & S \leftarrow S \ \boldsymbol{div}\ P \\
 & \textsf{goto } N \\
[A] & S \leftarrow S * P \\
[N] & K{+}{+} \\
 & \textsf{goto } C \\
[F] & Y \leftarrow (S)_1
\end{array}$$

Note that by definition of "bounded min" (Thm 5.13) if there is no i as required in line 11, then K gets the value $\boldsymbol{Lt}(Z) + 1$. $\square$

7.5 The step-counter predicate

Consider the predicate

$$\begin{aligned}\boldsymbol{stp}^{(n)}(\vec{x}, y, t) &\Leftrightarrow \mathcal{P}_y, \text{ with inputs } \vec{x}, \text{ halts in } t \text{ or fewer steps}\\ &\Leftrightarrow \exists \text{ a computation of } \mathcal{P}_y, \text{ with inputs } \vec{x}, \text{ of length} \le t+1.\end{aligned}$$

Theorem 7.6. $\boldsymbol{stp}^{(n)} \in \mathcal{G}$-COMP.

Proof 1 (using CT): Use the algorithm

"Run $\mathcal{P}_y$ with inputs $\vec{x}$ up to t steps;
if it has halted,
then $\boldsymbol{stp}^{(n)}(\vec{x}, y, t) \leftarrow 1$
else $\boldsymbol{stp}^{(n)}(\vec{x}, y, t) \leftarrow 0$." □

Proof 2 (not using CT): *Modify* the universal program to include a *step counter* Q, as follows. (Note that only two lines have been added $(*)$, and one line changed $(**)$).

$$\begin{array}{lll}
 & Z \leftarrow X_{n+1} + 1 & \\
 & S \leftarrow \prod_{i=1}^{n} (p_{2i})^{X_i} & \\
 & K \leftarrow 1 & \\
[C] & Q{+}{+} & (*) \\
 & \mathsf{if}\ Q > X_{n+2} + 1\ \mathsf{goto}\ E & (*) \\
 & \mathsf{if}\ K = \boldsymbol{Lt}(Z) + 1\ \mathsf{goto}\ F & \\
 & U \leftarrow \boldsymbol{r}((Z)_K) & \\
 & P \leftarrow p_{\boldsymbol{r}(U)+1} & \\
 & \mathsf{if}\ \boldsymbol{\ell}(U) = 0\ \mathsf{goto}\ N & \\
 & \mathsf{if}\ \boldsymbol{\ell}(U) = 1\ \mathsf{goto}\ A & \\
 & \mathsf{if}\ \neg(P|S)\ \mathsf{goto}\ N & \\
 & \mathsf{if}\ \boldsymbol{\ell}(U) = 2\ \mathsf{goto}\ M & \\
 & K \leftarrow \boldsymbol{min}\ i_{0<i<\boldsymbol{Lt}(Z)+1}[\boldsymbol{\ell}((Z)_i) + 2 = \boldsymbol{\ell}(U)] & \\
 & \mathsf{goto}\ C & \\
[M] & S \leftarrow S\ \boldsymbol{div}\ P & \\
 & \mathsf{goto}\ N & \\
[A] & S \leftarrow S * P & \\
[N] & K{+}{+} & \\
 & \mathsf{goto}\ C & \\
[F] & Y{+}{+} & (**)
\end{array}$$

□

NOTES:

1. The predicate

$$\boldsymbol{stp}_1^{(n)}(\vec{x}, y) \;\Leftrightarrow \text{“}\mathcal{P}_y, \text{ with inputs } \vec{x}, \text{ halts (at all)”}$$

   is not $\mathcal{G}$-computable, since it is (essentially) HP.

2. Similarly, the predicate

$$\boldsymbol{stp}_2^{(n)}(\vec{x}, y) = \begin{cases} t+1 & \text{if } \mathcal{P}_y \text{ halts on on } \vec{x} \text{ in } t \text{ steps} \\ 0 & \text{if } \mathcal{P}_y \text{ does not halt on input } \vec{x} \end{cases}$$

   is not $\mathcal{G}$-computable, since a $\mathcal{G}$-program for $\boldsymbol{stp}_2^{(n)}$ could easily provide a solution to HP.

3. What about $\boldsymbol{stp}_3^{(n)}(\vec{x}, y)$ — like $\boldsymbol{stp}_2^{(n)}$, but with ‘$\uparrow$’ instead of ‘0’?

4. We can prove a stronger result than Theorem 7.6:

Theorem 7.7. $\boldsymbol{stp}^{(n)} \in \mathrm{PR}$.

Proof: Let

$$\boldsymbol{K}^{(n)}(\vec{x}, y, t)$$

be the ***instruction counter*** function, giving the number of the instruction to be read by $\mathcal{P}_y$, with inputs $\vec{x}$, at time t, and let

$$\boldsymbol{S}^{(n)}(\vec{x}, y, t)$$

giving the *state*, at time t, when $\mathcal{P}_y$ has inputs $\vec{x}$.

We define $\boldsymbol{K}^{(n)}$ and $\boldsymbol{S}^{(n)}$ by ***simultaneous primitive recursion*** on t (see p. 6-3). For the *basis*, put

$$\begin{aligned} \boldsymbol{K}^{(n)}(\vec{x}, y, 0) &= 1, \\ \text{and} \quad \boldsymbol{S}^{(n)}(\vec{x}, y, 0) &= \textstyle\prod_{i=1}^{n} p_{2i}^{x_i}. \end{aligned}$$

For the *recursion step*, put

$$\begin{aligned} k &= \boldsymbol{K}^{(n)}(\vec{x}, y, t), & s &= \boldsymbol{S}^{(n)}(\vec{x}, y, t), \\ L &= \boldsymbol{Lt}(y+1), & u &= \boldsymbol{r}((y+1)_k), \\ b &= \boldsymbol{\ell}(u), & c &= \boldsymbol{r}(u), \\ p &= p_{c+1}. \end{aligned}$$

Then $\boldsymbol{K}^{(n)}(\vec{x}, y, t+1) =$

$$\begin{cases} k & \text{if } k = L+1 \\ k+1 & \text{if } (1 \le k \le L) \wedge (b \le 2 \vee p \not| s) \\ (\mu i < L+1)[i > 0 \wedge \boldsymbol{\ell}(y+1)_i = b \dot{-} 2] & \text{otherwise,} \end{cases}$$

and $\boldsymbol{S}^{(n)}(\vec{x}, y, t+1) =$

$$\begin{cases} s * p & \text{if } (1 \le k \le L) \wedge (b = 1) \\ s \ \boldsymbol{div}\ p & \text{if } (1 \le k \le L) \wedge (b = 2) \wedge p | s \\ s & \text{otherwise.} \end{cases}$$

By Theorem 6.6, $\boldsymbol{K}^{(n)}$, $\boldsymbol{S}^{(n)} \in$ PR. Finally,

$$\boldsymbol{stp}^{(n)}(\vec{x}, y, t) \iff \boldsymbol{K}^{(n)}(\vec{x}, y, t) > \boldsymbol{Lt}(y+1). \quad \square$$

We conclude this section by answering some of the questions concerning the properness of the "$\subseteq$" inclusions in the diagrams in Section 4 (pp. 4-4, 4-7.) In particular,

- $\mathcal{G}$-COMP $=$ EFF by CT, and
- $\mathcal{G}$-COMP $\subset$ FN, since $\mathcal{G}$-COMP is *countable* (φ_0, φ_1, $\varphi_2, \dots$), and FN is *uncountable* by Cantor's theorem (Theorem 1.11(a)).

Similarly for $\mathcal{G}$-COMP($\vec{g}$), etc., using Rel-CT.

NOTE: By re-proving Cantor's Theorem in the present context, we can produce a ***non-computable total function*** f as follows. Define

$$f(n) = \begin{cases} \varphi_n(n) + 1 & \text{if } \varphi_n(n) \downarrow \\ 0 & \text{if } \varphi_n(n) \uparrow . \end{cases}$$

Then $f \notin \mathcal{G}$-COMP, since (as we can easily see) for all n, $f(n) \neq \varphi_n(n)$. So f is a ***witness*** that $\mathcal{G}$-COMP $\subset$ FN.

Intuitively, f is not computable because the above definition by cases is ***not effective***, owing to the undecidability of HP.

So f is ***mathematically definable***, i.e., ***specifiable***, but ***not computable***.

Note the use of ***diagonalisation*** again here!

Hence by CT,

$$
\boxed{\begin{array}{ccccccc}
 & & \mathcal{G}\text{-COMP} & = & \text{EFF} & \subset & \text{FN} \\
 & & \cup & & \cup & & \cup \\
\text{PR} & \subseteq & \mathcal{G}\text{-TCOMP} & = & \text{TEFF} & \subset & \text{TFN}
\end{array}}
$$

and, using Rel-CT,

$$
\boxed{\begin{array}{ccccccc}
\text{PR}(\vec{g}) & \subseteq & \mathcal{G}\text{-COMP}(\vec{g}) & = & \text{EFF}(\vec{g}) & \subset & \text{FN} \\
\cup & & \cup & & \cup & & \cup \\
\text{TPR}(\vec{g}) & \subseteq & \mathcal{G}\text{-TCOMP}(\vec{g}) & = & \text{TEFF}(\vec{g}) & \subset & \text{TFN}
\end{array}}
$$