XIX Automata & Formal Languages

NINETEENTH LECTURE 17 November 2022

RECAP: §4.5 Gödel's Recursive Functions

The following operations on partial functions were considered by Alonzo Church (1903–1995):
The functions

$$\pi_{k,i} : \mathbb{W}^k \to \mathbb{W} : \vec{w} \to w_i \text{ (\textit{projection functions})}$$
$$c_{k,\varepsilon} : \mathbb{W}^k \to \mathbb{W} : \vec{w} \to \varepsilon \text{ (\textit{constant functions})}$$
$$s : \mathbb{W} \to \mathbb{W} : w \mapsto v \quad \text{(where } \#(v) = \#(w) + 1\text{; the \textit{successor function}).}$$

BASIC FUNCTIONS

are called *basic functions*. We have already proved that all basic functions are computable.

Suppose $f : \mathbb{W}^m \dashrightarrow \mathbb{W}$ and $g_1, ..., g_m : \mathbb{W}^k \dashrightarrow \mathbb{W}$ are partial functions, then the partial function h defined by

$$h(\vec{w}) := f(g_1(\vec{w}), ..., g_m(\vec{w}))$$

is called the *composition of f with $(g_1, ..., g_m)$*. The notational convention used for operations applies here as well: if any term on the right hand side is undefined, then so is the left hand side.

COMPOSITION

Suppose $f : \mathbb{W}^k \dashrightarrow \mathbb{W}$ and $g : \mathbb{W}^{k+2} \dashrightarrow \mathbb{W}$ are partial functions, then the function h defined by the recursion equations

$$h(\vec{w}, \varepsilon) = f(\vec{w}) \text{ and}$$
$$h(\vec{w}, s(v)) = g(\vec{w}, v, h(\vec{w}, v))$$

is called the *recursion result of f and g*.

RECURSION

Suppose $f : \mathbb{W}^{k+1} \dashrightarrow \mathbb{W}$ is a partial function, then the partial function h defined by

$$h(\vec{w}) := \begin{cases} v & \text{if for all } u \leq v, \text{ we have that } f(u)\downarrow \text{ and} \\ & \quad v \text{ is } <\text{-minimal such that } f(\vec{w}, v) = \varepsilon \text{ or} \\ \uparrow & \text{if for all } v,\ f(\vec{w}, v) \neq \varepsilon \end{cases}$$

is called the *minimisation result of f*.

MINIMISATION

We say that a class $\mathcal{C}$ of partial functions is closed under composition, recursion, or minimisation if, whenever $f, g, g_1, ..., g_m$ are in $\mathcal{C}$, then the composition of f with $(g_1, ..., g_m)$, the recursion result of f and g, or the minimisation result of f, respectively, are in $\mathcal{C}$.

RECURSIVE FNS : smallest class containing basic fns & closed under ∇ COMP, REC, MIN.

Theorem 4.24 Every partial recursive function is computable.

PROOF

1. We already saw that the basic functions are computable.
2. Computable fns are closed under composition.
3. So it remains to show that the computable fns are closed under

RECURSION
&
MINIMISATION.

RECURSION: f, g computable

$$h(\vec{w}, \varepsilon) := f(\vec{w})$$
$$h(\vec{w}, s(v)) := g(\vec{w}, v, h(\vec{w}, v))$$

Show that h is computable.

Assume $\vec{w}$ & v are given and compute

$$h(\vec{w}, v).$$

We describe a register machine.

Take two empty unused reg. k, l.

Compute $f(\vec{w})$, write it to reg. l.

[If $f(\vec{w})\uparrow$, then produces the desired result.]

If $v = \varepsilon$, then just output the content of reg l.

O/w, apply s to reg. k.

SUBROUTINE Compute $g(\vec{w}, v, u)$

where u is the content of reg l.
We write this into register l.

Check whether v = reg. content of k.
If so, output reg. l & halt.
O/w apply s to reg. k and
restart subroutine.

(Recursion)
q.e.d.

MINIMISATION Assume that f is computable.

Take reg. k unused & empty.

SUBROUTINE Compute $f(\vec{w}, u)$ where u is the current content of reg. k.

If ↑, then this is the desired result.

If ↓, check whether this is ε.

If yes, output the reg. content of k & halt.

If not, apply s to reg. k and restart the subroutine.

(Minimisation)
q.e.d.

Corrected after the lecture:
YES & NO were the wrong way round during the lecture.

Remark 1. Since $+, \cdot$, exp can be done with rec. fns, they can be done w/ RM.

Remark 2. More than that: we are allowed to use RECURSION & MINIMISATION as construction principles for computable fns / RM.

Remark 3. Remember that there is a bijection

$$z : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

Cantor ZIGZAG function

$$z : (i,j) \longmapsto \frac{(i+j)(i+j+1)}{2} + j$$

Since $+, \cdot$, and "find the unique v s.t. $2 \cdot \#v = \#v$" are computable, this gives us a computable bijection

$$\mathbb{N}^2 \longrightarrow \mathbb{N}$$

$$(v,w) \longmapsto v \text{ s.t. } \#v = z(\#v, \#w).$$

MERGING

$$v, w \longmapsto v * w$$

where $v*w$ is the unique v s.t.

$$\#v = z(\#v, \#w)$$

SPLITTING

$$w \longmapsto (v, v) \text{ s.t.}$$

$$\#w = z(\#v, \#v) \quad [\text{inverse of } z]$$

Write $w_{(0)} := v$ $\quad w_{(1)} := v$.

§4.6 Remark on the choice of the alphabet

Σ alphabet $\quad X \subseteq \Sigma^* = \mathbb{W}$

NOTION OF COMPUTABILITY LIVES ON $\mathbb{W} = \Sigma^*$.

Clearly, if $\Sigma \subseteq \Sigma'$, then every Σ-RM is a Σ'-RM.

Is it conceivable that if $\Sigma' \neq \Sigma$, that the notion of Σ'-computability is strictly stronger than the notion of Σ-computability.

Answer: NO!

Proof: Encode everything in binary strings & prove equivalence.

[In the typed notes: Details.]

§ 4.7 Universality

Goal: Show that there is a universal RM that can mimic every RM.

→ The SOFTWARE principle.

Encoding Register machines

Fix alphabet Σ and add extra symbols

boldface zero, boldface one, boldface plus, boldface minus, boldface q-mark, boldface parenthesis, boldface comma, boldface mapsto, BOX

$$\mathbb{0}\ \mathbb{1}\ \#\ \square\ ?_{\circ}\ (\ )\ ,\ \mapsto\ \square$$

Call the whole thing Σ'.

① Encode $k \in \mathbb{N}$ as binary using $\mathbb{0}$ & $\mathbb{1}$:

e.g. $\mathbb{1}\mathbb{0}\mathbb{0}\mathbb{1}\mathbb{1} = \text{code}(19)$

② Assume $Q = \{q_0, \dots, q_u\}$. We write Σ, not Σ'

$\text{code}(q_k) := \text{code}(k)$

③ Encode instructions $I \in \text{Instr}(\Sigma, Q)$ using $\#, \square, ?_{\circ}, (,), ,$

E.g. $\#(\text{code}(k)_{,}\, a_{,}\, \text{code}(l))$ is $\text{code}(+(k,a,l))$

④ Encode program line

$$q \longmapsto I$$

as $\quad code(q) \mapsto code(I)$

$$=: code(q \longmapsto I)$$

⑤ Encode RM with program P as

$$code(q_0 \longmapsto P(q_0)) \# \cdots \# code(q_n \longmapsto P(q_n))$$

⑥ If $\vec{w}$ is a sequence of words

$$code(\vec{w}) := \square \, w_0 \, \square \cdots \square \, w_n \, \square$$

if $\vec{w} = (w_0, \ldots, w_n)$

⑦ Encode configuration $(q, \vec{w})$ by

$$code(q) \, code(\vec{w})$$

Operations performed & questions answered by register machines

1. Is w a code for a ...
   1. number / state?
   2. instruction?
   3. program line?
   4. register machine?
   5. sequence of words?
   6. configuration?
2. If w is a code for a register machine, which instruction does it have for state q?
3. If w is a code for a sequence of words, how long is it?
4. If w is a code for a configuration, which state is it in?
5. Given $\mathrm{code}(C)$ and $\mathrm{code}(I)$, apply I to C and output the code of the resulting configuration.
6. Given $\mathrm{code}(C)$ and $\mathrm{code}(M)$, output the code of the configuration that is the transformation of C by M.

Lemma 4.23 The function

$$h: w, u, v \longmapsto \begin{cases} \mathrm{code}(C(M, \vec{w}, \#v)) \\ \quad \text{if there are } M, \vec{w} \text{ s.t.} \\ \quad w = \mathrm{code}(M) \\ \quad u = \mathrm{code}(\vec{w}) \\ \uparrow \quad \text{o/w} \end{cases}$$

is computable.

Proof. Apply recursion with the operation given in ❻.

$h(\text{code}(M), \text{code}(\vec{w}), \varepsilon) := \text{code}(q_S)\,\text{code}(\vec{w})$

$h(\text{code}(M), \text{code}(\vec{w}), s(v)) := \text{code}(C')$

where C' is the result of transforming $h(\text{code}(M), \text{code}(\vec{w}), v)$ via the machine M.

q.e.d.

Corollary 4.24 The function

$$t_{M,k}(\vec{w}, v) := \begin{cases} q & M \text{ has halted before time } \#v \text{ on input } \vec{w} \\ \varepsilon & \text{o/w} \end{cases}$$

called the truncated computation is computable.

Proof Using recursion on the function h from Lemma 4.23, check all values of h for words v s.t. $\#v < \#v$. If one of the values is in state $q_{\#}$ output a ; o/w output ε. q.e.d.

Theorem 4.25 THE SOFTWARE PRINCIPLE

The function

$$g(v,w) := \begin{cases} f_{M,k}(\vec{w}) & \text{if } v = \text{code}(M),\ w = \text{code}(\vec{w}) \text{ with length } k \\ \uparrow & \text{o/w} \end{cases}$$

is computable.

This means that there is a UNIVERSAL REGISTER MACHINE U s.t.

$$f_{U,2}(v,w) = g(v,w),$$

i.e, a RM that gets $v = \text{code}(M)$ as input, regarding it as SOFTWARE and performs the computation of M.

$\longrightarrow$ Proof in Lecture XX.