XIX

Nineteenth Lecture
AUTOMATA & FORMAL LANGUAGES
Friday, 17 November 2023

We are in the proof of

Theorem Every recursive function is computable.
[The converse is also true; discussed later.]

Lecture $\overline{\text{XVIII}}$, page 8.

By induction: show closure of computable functions under RECURSION & MINIMISATION.

Recursion
Lecture $\overline{\text{XVIII}}$
page 9

Recursion Fix f, g computable

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

Show h is computable.
Fix $\vec{w}, v$ & describe a RM that computes $h(\vec{w}, v)$.

Two scratch registers k & ℓ.
Compute $f(\vec{w})$ and write into register ℓ.
Check if $v = \varepsilon$. $\xrightarrow{\text{YES}}$ Output register ℓ.
↓ NO
REPEAT SUBROUTINE
UNTIL v is equal to register k.

SUBROUTINE
- Compute $g(\vec{w}, v_0, v_1)$
  where v_0 is the content of reg. k
  & v_1 is the content of reg. ℓ
- Write result into register ℓ.
- Apply s to register k.

Output register ℓ.

Minimisation

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

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

is called the *minimisation result of* f.

Proof that if f is computable, then so is h.
Input $\vec{w}$. Describe the RM that performs $h(\vec{w})$.

Use scratch registers k, l, m, all initially empty.

Repeat SUBROUTINE
until register l is empty.
Write reg. k as output.

SUBROUTINE
- Replace content of reg. k with content of reg. m.
- Apply s to reg. k and write the result in reg. m.
- Compute $f(\vec{w}, v)$ where v is reg. k
- Write result in reg. l.

q.e.d.

Application

Already saw that $+, \cdot$ are recursive.
Now we know they are computable.

We mean $(v, w) \longmapsto u$ iff $\#u = \#v + \#w$.

In particular consider

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

the CANTOR ZIGZAG FUNCTION.

This gives us the computability of the function

$(v, w) \longmapsto u$ iff
$$\#u = z(\#v, \#w)$$

which is a bijection between $\mathbb{N}^2$ and $\mathbb{N}$.
This allows us to computably merge and split words.

$$(v, w) \longmapsto u$$

$$\text{iff } \#u = z(\#v, \#w)$$

Write $u =: v * w$ MERGING

Similarly, the maps

$$v \longmapsto v_{(0)}$$

$$v \longmapsto v_{(1)}$$

with property that $v_{(0)} * v_{(1)} = v$ are computable.

SPLITTING

$\longrightarrow$ ES #3

Example (40)

4.6 Remark on the choice of alphabet

We defined computability for partial functions $f : \mathbb{W}^k \dashrightarrow \mathbb{W}$ in terms of Σ-register machines: the instructions and behaviour of register machines are closely tied to their alphabet and register machines can only compute partial functions that use the letters that the machines are built for. Clearly, if $\Sigma \subseteq \Sigma'$ and $f : \mathbb{W}^k \dashrightarrow \mathbb{W}$ is computable by a Σ-register machine, then it is computable by a Σ'-register machine. But could it be that the notion of computability

[Not lectured.]

$$A \subseteq \mathbb{W} = \Sigma^*$$

COMPUTABLE $\iff \exists M$ s.t. $\chi_A = f_{M,1}$.

$$M = (\Sigma, Q, P)$$

Question: Is it clear that this notion of compubility (Σ-computability) is equivalent to Σ'-computability for $\Sigma' \supseteq \Sigma$.

Answer: Yes, so don't have to worry about adding extra symbols.

§ 4.7 UNIVERSALITY

Machines used to be built for <u>one</u> purpose; modern computers can change their program [the difference between HARDWARE & SOFTWARE].

This is a fundamental cultural change.

<u>Coding of machines</u>

Using § 4.6, we add extra symbols to make things easy for us.

$\mathbf{0} \quad \mathbf{1} \quad \mathbf{(} \quad \mathbf{)} \quad \mathbf{,} \quad \boxplus \quad \boxminus \quad \mathbf{?} \quad \mapsto \quad \boxminus \quad \mathcal{E}$

boldface parenthesis

boldface comma

<u>Natural numbers</u>

Encode them in binary:

$\mathbf{0}$	0
$\mathbf{1}$	1
$\mathbf{10}$	2
$\mathbf{11}$	3

etc.

code(k) for the code of k

States Since Q does not matter as a set, we can assume that

$$Q = \{q_0, \dots, q_k\}$$ for some fixed listing of states

$$\text{code}(q_i) := \text{code}(i).$$

Instructions Example: $I = +(k, a, q')$

$\text{code}(I) :=$

$$\#(\!(\text{code}(k) \triangledown a \triangledown \text{code}(q')\!)$$

Similarly for the other instructions.

Program lines $q \longmapsto P(q)$ where $P(q)$ is an instruction

$$\text{code}(q \mapsto P(q)) := \text{code}(q) \Rightarrow \text{code}(P(q)).$$

Σ-RM $(\Sigma, Q, P) = M$ where $Q = \{q_0, \dots, q_k\}$

$$\text{code}(M) := \text{code}(q_0 \mapsto P(q_0)) \triangledown \cdots \triangledown \text{code}(q_k \mapsto P(q_k))$$

Seq. of words in $W = \Sigma^*$ $\vec{w} = (w_0, \dots, w_n)$ $\text{code}(\vec{w}) :=$

$$e\, w_0 \,\square \cdots \square\, w_n.$$

Configurations $C = (q, \vec{w})$ $\text{code}(C) :=$

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

Observation ① Whether something is a code for
- state
- instruction
- program like
- RM
- seq. of word
- config.

can be checked by a RM.

② If C is a configuration and M a reg. machine with
$w = \text{code}(C)$
$v = \text{code}(M)$, then

$$(w,v) \longmapsto u$$

where $u = \text{code}(C')$ and M transforms C to C'
is performed by a RM.

Proposition The operation
$$w, v, u \longmapsto \text{code}(C(M, \vec{w}, \#u))$$
where $\text{code}(M) = w$, $\text{code}(\vec{w}) = v$
can be performed by a RM.

Proof. $C(M, \vec{w}, u)$ was defined in terms of the TRANSFORM operation by recursion. Using Obs. ② & the fact that comp. fns are closed under recursion yield the desired result. q.e.d.

Corollary 4.25. The total operation "check whether M has halted with input $\vec{w}$ after at most $\#v$ steps" can be performed by a register machine. We call the corresponding total (characteristic) function

$$t_{M,k}(\vec{w}, v) := \begin{cases} a & M \text{ has halted with input } \vec{w} \text{ after at most } \#v \text{ steps and} \\ \varepsilon & \text{otherwise} \end{cases}$$

the *truncated computation of M*. Note that this is a characteristic function.

Proof.

We construct RM with scratch registers k, l.

Repeat SUBROUTINE until register k = v.
Halt and output ε.

Note that we can only call the subroutine #v many times. Since $(\mathbb{W}, <) \cong (\mathbb{N}, <)$, this terminates in finite time.

SUBROUTINE:
- Replace content of reg. k by content of reg. l
- Compute $C(M, \vec{w}, u) = (q, \vec{v})$ where u is the content of k
- Check if $q = q_H$
  - YES: Halt & output a.
  - NO: Apply s to reg. k and write the result in reg. l.

q.e.d.

Theorem 4.26 (The Software Principle)**.** There is a register machine U, called a *universal register machine* such that for every register machine M and sequence of words $\vec{w}$, we have that

$$f_{U,2}(v,u) = \begin{cases} f_{M,k}(\vec{w}) & \text{if } v = \text{code}(M) \text{ for a register machine } M \\ & \quad \text{and } u = \text{code}(\vec{w}) \text{ for a sequence of words of length } k, \\ \uparrow & \text{otherwise.} \end{cases}$$

Proof in Lecture XX

The Software Principle changes the paradigm of "machine".

Traditionally, we had one machine per task, now we have one machine U [for UNIVERSAL REGISTER MACHINE] that can do the job of any machine with the right input !!!

Remark. U has a fixed upper register index and this means that a machine with limited memory can do ANY task, assuming that you feed it the right software [essentially the code of the machine performing the task].