RECURSION THEORY
Lecture X

12 Jan 2022

14 Januar = Tarski's birthday
= Gödel's date of death

UNESCO World Logic Day

Website DENIZ SARIKAYA

http://wld.cipsh.international
→ WORLD LOGIC DAY 2022

2019 General Conference UNESCO
instituted WLD

CIPSH
Conseil International de Philosophie
et des Sciences Humaines

Internationaler Rat der Geisteswissenschaften

Monday Guardian Puzzle Column
→ Alex Bellos

Monday 10 January 2022:
Gödel's Incompleteness Theorem

starting tomorrow

Two remaining topics before we get to Gödel's Incompleteness Theorem in Lecture XI :

1. Fixed pt thm
2. Growth functions

#1. is going to give us the answer to the question
"Are $\mathbb{K}$ & $\mathbb{K}_0$ index sets ?" [NO]

boldface Halting problem

Recursion Theorem / Kleene Fixed Point Theorem

If g is a total computable function, there there is a P s.t.

$$f_P = f_{g(P)}$$

Proof.

$$w, v \longmapsto \begin{cases} f_{f_w(w)}(v) & \text{if } w \in K \\ \uparrow & \text{otherwise} \end{cases}$$

This operation is computable by the usual argument: given w, check whether this means that $f_w(w)\downarrow$ $w \in K$; if you fail, produce $\uparrow$, otherwise use $f_w(w)\downarrow$ as code for a function and apply that to v.

By s-m-n, we find total computable h s.t.

$$f_{h(w)}(v) = \begin{cases} f_{f_w(w)}(v) & \text{if } w \in K \\ \uparrow & \text{o/w.} \end{cases}$$

Since both g and h are total computable, so is $g \circ h$. Thus there is some p s.t. $f_p(v) = g(h(v))$.

Claim

$h(P)$ is a fixed point:

$$\boxed{f_{h(w)}(v) = \begin{cases} f_{f_w(w)}(v) & w \in K \\ \uparrow & o/w \end{cases}}$$

$$\underline{f_P}(v) = g(h(v)) \quad (**)$$

i.e., $f_{h(P)} = f_{g(h(P))}$.

To make notation easier, let's write
$$\boxed{f_{h(w)}(v) = f_{f_w(w)}(v)} \quad (*)$$
and understand this as "undefined" if $f_w(w)\uparrow$.

Check:

$$f_{h(P)} \overset{(*)}{=} f_{f_P(P)}(v) \overset{(**)}{=} f_{(g \circ h)(P)}(v)$$
$$= f_{g(h(P))}(v).$$

That's precisely what we claimed!

q.e.d.

Applications

1. If g is a constant function
$$g(w) = c$$
for some c. By the fixed point thm, we find P s.t.
$$f_P = f_{g(P)} = f_c.$$
[That's not terribly interesting.]

2. Consider
$$g(w,v) := \begin{cases} \varepsilon & \text{if } v = w \\ \uparrow & \text{o/w} \end{cases}.$$
Clearly a computable function. So by s-m-n theorem, we find a total computable h s.t.
$$f_{h(w)}(v) = g(w,v).$$
Apply the fixed point theorem to h and obtain P s.t. $\boxed{f_{h(P)} = f_P.}$ (*)

$$f_{h(P)}(v)\downarrow \iff P = v$$

So $W_{h(P)} = \mathrm{dom}(f_{h(P)}) = \{P\}$.

$\quad\ \ \ \|$

$\quad\ W_P$

So, we obtained some P s.t.

$$W_P = \{P\}.$$

<u>Corollary</u> $\mathbb{K}$ is not an index set.

Note that if X is an index set and $P \in X$, then if $W_P = W_{P'}$, then $P' \in X$.

Consider P as in Application 2.

Clearly $f_P(P)\downarrow$, and thus $P \in \mathbb{K}$!

So if $\mathbb{K}$ is an index set, then every other P' s.t. $W_{P'} = W_P$ must also be in $\mathbb{K}$.

By the Padding Lemma, there are infinitely many P' s.t. $W_{P'} = W_P$.

Take P' s.t. $P' \neq P$ and $W_{P'} = W_P = \{P\}$.

Then $f_{P'}(P')\uparrow$, so $P' \notin \mathbb{K}$.

Similarly, $\mathbb{K}_0 = \{ w * v \,;\, f_w(v)\downarrow \}$ [the boldface Halting problem] is not an index set.

This explains why we called $\mathbb{K}_1 = \text{Nonemp} = \{ w \,;\, \exists v\ f_w(v)\downarrow \}$ which is many-one-equivalent to $\mathbb{K}$ and $\mathbb{K}_0$ the index set version of the halting problem.

GROWTH FUNCTIONS

Def. If $f : \Sigma^* \longrightarrow \Sigma^*$ is any total function, we can define its growth function

$$\gamma_f : \mathbb{N} \longrightarrow \mathbb{N}$$

by $\gamma_f(n) := \max\{ |f(w)| \,;\, \underbrace{|w| \leq n}_{} \}$

This is welldefined since Σ is finite.

since Σ is finite: only finitely many of these

If $g, h : \mathbb{N} \longrightarrow \mathbb{N}$, we say that

g dominates h	$g \geq h$	iff f.a. n $g(n) \geq h(n)$
g strictly dominates h	$g > h$	iff f.a. n $g(n) > h(n)$
g eventually dominates h	$g \geq^* h$	there is N s.t. f.a. $n > N$ $g(n) \geq h(n)$
g eventually strictly dominates h	$g >^* h$	there is N s.t. f.a. $n > N$ $g(n) > h(n)$

We write $\boxed{h \leq^* g}$.

<u>Definition</u> A function $g : \mathbb{N} \longrightarrow \mathbb{N}$ is called a <u>computable growth fn</u> if there is a computable total function $f : \Sigma^* \longrightarrow \Sigma^*$ s.t. $g = \gamma_f$.

[<u>Remark</u>. Note that this is <u>not</u> the same as "computable" for functions from $\mathbb{N} \longrightarrow \mathbb{N}$. Imagine that $\Sigma = \{\bar{0}, \bar{1}\}$ and that we interpret words in Σ^* as binary numbers. Then functions from Σ^* to Σ^* are in canonical bijection with functions from $\mathbb{N}$ to $\mathbb{N}$. So the obvious definition of <u>computable</u>

for functions from $\mathbb{N}$ to $\mathbb{N}$ would be:
the canonical representation on Σ^*
is computable.
By our theory, we know that the characteristic function of the halting problem is not computable.

$$f: w \longmapsto \begin{cases} \overline{0} & \text{if } w \notin \mathbb{K} \\ \overline{1} & \text{if } w \in \mathbb{K}. \end{cases}$$

But γ_f is the constant function 1.
So by moving from the function to its growth behaviour, we are losing the information coded in f.

<u>Definition</u> Consider the following function:

$$S : \mathbb{N} \longrightarrow \mathbb{N}$$

$$S(n) := \max\{ i ;\ \exists P\ |P| \leq n \text{ and } f_P(\varepsilon)\downarrow \text{ after exactly } i \text{ steps} \}$$

This is the length of the longest computation (with empty input) of a program of length at most n that still halts.

Since Σ is finite, there are only finitely many programs. Some of these halt, others don't. The collection of halting moments is a finite subset of $\mathbb{N}$ and thus has a maximal element.

We'll see that S is not a computable growth function. Moreover, no function that dominates S can be a computable growth function.

In other words:

$\boxed{\text{$S'$ grows faster than any function that can be computed.}}$

Historical Remark

There is a class of functions smaller than the computable functions called primitive recursive functions [this is the notion of computability that Gödel used in his proof of the incompleteness theorem].

All primitive recursive fn. are total, so it's clear that there are computable partial fns that are not prim. rec.

Are there also total computable fns that are not prim. rec.

→ ACKERMANN function.

Remember We defined addition by recursion:

$n+m$

$\mathrm{add}_n(0) := n$

$\mathrm{add}_n(m+1) := \mathrm{add}_n(m) + 1.$

Multiplication $\mathrm{mult}_n(0) := 0$

$\mathrm{mult}_n(m+1) := \mathrm{mult}_n(m) + n.$

$\exp_n(m+1) = \exp_n(m) \cdot n$

We can generalise to

$$\boxed{Ack(k+1, n, m+1) := Ack(k, Ack(k+1, n, m), n)}$$

$k=0$ successor
$k=1$ addition
$k=2$ multiplication
$k=3$ exponentiation
$k=4$ hyperexponentiation

⋮

Ackermann's theorem: each prim. rec. fn is bounded by some

$$(n, m) \longmapsto Ack(k, n, m)$$

Therefore, the Ackermann function itself cannot be prim. rec.

Theorem If g dominates S, then g cannot be a computable growth function.

Proof. General idea

Assume that g is a computable growth function and provide an algorithm to determine whether $w \in K$.

Let g be a function as required and let $f : \Sigma^* \longrightarrow \Sigma^*$ s.t.

f computable

$g = \gamma f$.

$\boxed{\text{fix program } P \text{ s.t. } f_P = f}$

This means that

$$g(n) = \max\{|f(w)| ; |w| \leq n\}.$$

Also g dominates S, so for all n

$$g(n) \geq S(n)$$

$$S(n) = \max\{i ; \exists Q\ |Q| \leq n \text{ and } f_Q(\varepsilon)\downarrow \text{ after exactly } i \text{ steps}\}.$$

Consider a fixed program P and a fixed word w. It's easy to construct a program P_w s.t. P_w do the following: it writes w in the input band then runs P.

$$f_{P_w}(\varepsilon) = f_P(w)$$

[Essentially the s-m-n theorem: curry the two variables w and P].

In particular $w \longmapsto P_w$ is computable.

We now give an algorithm to determine whether $w \in K$.

Given w, compute P_w.

Under the assumption that g is computable growth fn and $g = \gamma_F$, check all computations $f_Q(\varepsilon)$ for $|Q| \leq |P_w|$.

We only need to check each of the computations up to time $S(|P_w|) \leq g(|P_w|)$ but $g(|P_w|)$ is computable by f.

So after a finite amount of time I have checked all computations

$$f_Q(\varepsilon)$$

for all Q with $|Q| \leq |P_w|$ up to time at least $S(|P_w|)$. So a computation that hasn't halted by that time will not halt.

So, we know for each Q s.t. $|Q| \leq |P_w|$ whether $f_Q(\varepsilon)$ halts.

In particular, we have checked whether

$$f_{P_w}(\varepsilon) \text{ halts,}$$

and that means whether $f_P(w)$ halts.

We only need this for the program $f_w(w)$.
So if $w \in \mathbb{K}$, then this algorithm has determined that it halts.
But if $w \notin \mathbb{K}$, then the answer the algorithm gives is correct since no computation halts after our bound.

q.e.d.

NEXT LECTURE: tomorrow 16^{15}.