XXII Twenty-Second Lecture of Automata & Formal Languages

24 NOVEMBER 2022

RECAP Lecture XXI:

Computably enumerable $\iff \Sigma_1$

Computable $\iff \Delta_1$

ZIGZAG Technique

IMPORTANT

If you do not know what the operations

$u, v \longmapsto u * v$

$u \longmapsto u_{(0)}$

$u \longmapsto u_{(1)}$

are, re-visit the section on merging & splitting! That's the final part of §4.5!

IF f and g are partial computable, you CANNOT say: "first compute f, then compute g" unless you're happy with never getting to g...

From Lecture $\overline{XXI}$:

CLOSURE PROPERTIES

get

$$\chi_{A\cap B}(\vec{w}) = \begin{cases} a & \text{if } \chi_A(\vec{w}) = a = \chi_B(\vec{w}) \\ \varepsilon & \text{o/w} \end{cases}$$

$$\chi_{A\cup B}(\vec{w}) = \begin{cases} \varepsilon & \text{if } \chi_A(\vec{w}) = \varepsilon = \chi_B(\vec{w}) \\ a & \text{o/w} \end{cases}$$

$$\chi_{\mathbb{W}\setminus A}(\vec{w}) = \begin{cases} a & \text{if } \chi_A(\vec{w}) = \varepsilon \\ \varepsilon & \text{o/w} \end{cases}$$

All obviously computable.
[This is essentially the idea of a product machine.]

Homework Try to make this idea of a "product machine" precise: wh… properties does a product operation need to have t… allow the con… that the cl… is closed under all Boole… operations?

Concatenation AB is computable. $[A, B \subseteq \mathbb{W}]$

Given w, check all possible $w = vu$ for v initial segment of w.
How many? $|w|$ many initial segments.
For each such, check $\chi_A(v) = a = \chi_B(u)$.
If so, output a;
if none of them work, output ε.

q.e.d.

Proposition 4.39 Closure properties of c.e. sets

The c.e. sets are closed under union, intersection, concatenation, but not complementation & difference.

Proof. COMPLEMENTATION DIFFERENCE are just the fact that $\mathbb{W}\setminus\mathbb{K}$ is Π_1, not Σ_1.

INTERSECTION The same construction as for computable sets with χ_A, χ_B works for c.e. sets with $\psi_A; \psi_B$.

UNION This does not work as "serial computation", so we need to use the zigzag technique.

$\longrightarrow$ ES#4.

CONCATENATION An application of zigzag to the proof idea in the computable case:

$(w,v,u) \in Z :\Longleftrightarrow$ v is initial segment of w with $w = vv'$ and after $\#u$ steps, we have
$$\psi_A(v) = a$$
$$\psi_B(v') = a$$

$$Y = \{(w,u) \,;\, (w, u_{(0)}, u_{(1)}) \in Z\}$$

$$w \in AB \Longleftrightarrow \exists v \; (w,v) \in Y.$$

q.e.d.

<u>Proposition</u> X c.e. $\iff$ there is a partial computable function f s.t. $X = \mathrm{ran}(f)$

REMARK: The name "c.e." derives from the idea that a RM can list (= enumerate) all elements of X [Cf. ES#4!]

<u>Proof.</u> "$\Longrightarrow$" If ψ_X computable, then so is

$$f : w \longmapsto \begin{cases} w & \text{if } \psi_X(w)\downarrow \\ \uparrow & \text{o/w} \end{cases}$$

Clearly $\mathrm{ran}(f) = X$.

"$\Longleftarrow$" Suppose $f : \mathbb{W} \dashrightarrow \mathbb{W}$ with $X = \mathrm{ran}(f)$.

[Naive idea: $w \in \mathrm{ran}(f) \iff \exists v\ f(v) = w$]

Suppose $f = f_{C,1}$. Use ~~zigzag~~ by

$$Z = \{(w, v, u) ;\ t_{C,1}(v, u) = a \ \&\ f_{C,1}(v) = w\}$$

$$Y := \{(w, v) ;\ (w, v_{(0)}, v_{(1)}) \in Z\}$$

$\mathrm{ran}(f) = p(Y)$.

q.e.d.

§ 4.10 Church's Thesis

Alan TURING

Different approaches but SAME NOTION OF COMPUTABILITY!

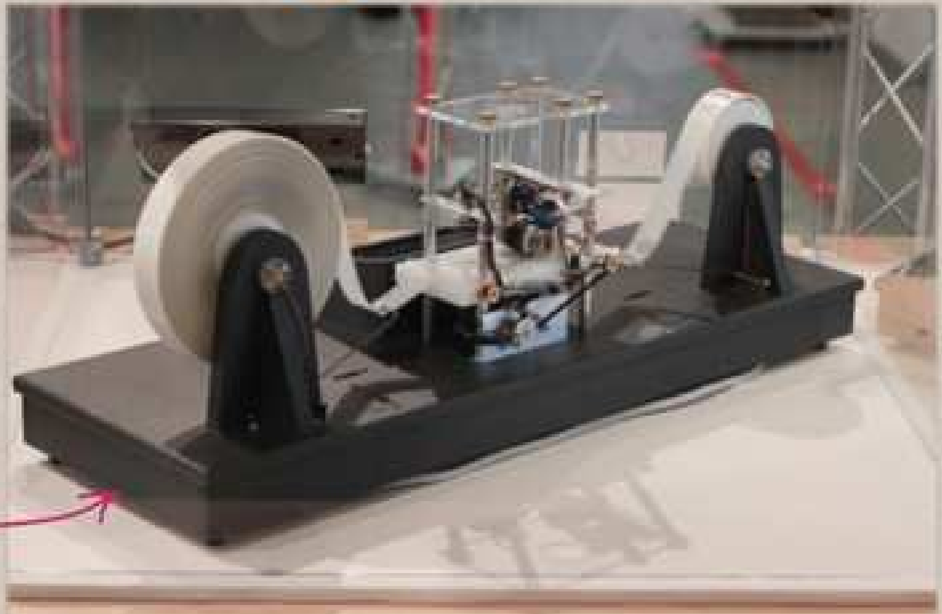
Alonzo CHURCH

Turing did not have RM.

Instead: TURING machine.

A tape with cells indexed by $\mathbb{N}$.

A read/write head that moves on the tape: reads & writes



Note that physical Turing machines like this one have only a finite tape.

Turing Machines

START STATE

HALT STATE

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

(i) An alphabet Σ with $\Sigma' := \Sigma \cup \{\Box\}$.

(ii) A finite set of states Q, disjoint from Σ'; $q_S, q_H \in Q$.

(iii) Write $\Omega := \Sigma' \cup Q$.

(iv) ***Turing instructions*** $\text{Instr} := \{\mathbf{L}, \mathbf{R}, \circ\} \times \Sigma' \times Q$.

$(\mathbf{L}, a, q)$ interpreted as "move head left, write a, go to state q".
$(\mathbf{R}, a, q)$ interpreted as "move head right, write a, go to state q".
$(\circ, a, q)$ interpreted as "don't move head, write a, go to state q".

(v) ***Turing programs*** $P : Q \times \Sigma' \to \text{Instr}$.

(vi) ***Turing configurations*** $C \in \Omega^*$ with precisely one state in the string.

$\alpha q \beta \quad \alpha, \beta \in (\Sigma')^*$

Interpretation: q is the state of machine and indicates the position of the R/W head.

(vii) ***Turing machine transforms C to C'***:

(L, c, q') : $\quad aqb \longrightarrow q'ac$

(R, c, q') : $\quad aqb \longrightarrow acq'$

$(\circ, c, q')$: $\quad aqb \longrightarrow aq'c$

(viii) ***Start configuration with input*** $\vec{w} = (w_0, ..., w_{k-1}) \in \mathbb{W}^k$:

$$q_S \Box w_0 \Box w_1 \Box \dots \Box w_{k-1} \Box =: C_S(\vec{w})$$

(ix) ***Turing computation with input*** $\vec{w} \in \mathbb{W}^k$:

$$C(0, M, \vec{w}) := C_S(\vec{w})$$

$$C(k+1, M, \vec{w}) := C'$$

if M transforms $C(k, M, \vec{w})$ to C'.

M halts if the state is q_H

$f_{M,k}(\vec{w}) := v$ if the M-computation with input $\vec{w}$ halts and v is the word between the 1st & 2nd $\Box$.

f Turing computable $:\Longleftrightarrow \exists M$ $f = f_{M,k}$.

While programs

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

(i) An alphabet Σ.

(ii) A number $n > 0$ of registers.

(iii) A finite program P, build by recursion from basic instructions:

   (a) `add`(i, a) is a while program
   ("add a to the ith register");

   (b) `remove`(i) is a while program
   ("remove the last letter from the ith register");

   (c) if P and Q are programs, then so is PQ;

   (d) if P is a program, then so is `while` i `not empty do` (P).

(iv) ***While configurations***: n-tuple of words together with a marker that tell us where in the program we are.

(v) ***While program transforms C to C'***: perform the instruction behind the marker and move the marker to the end of the instruction, back to the start of the while loop, or to the end of the while loop.

(vi) ***Start configuration with input $\vec{w} = (w_0, \ldots, w_{k-1}) \in \mathbb{W}^k$*** is $\vec{w}$ with marker at the start of the program.

(vii) ***While computation with input $\vec{w} \in \mathbb{W}^k$***: starts at the start configuration and performs transformations. ***Halts*** if there is no next instruction after the marker.

Exactly as before: Output at halting time in register 0; define partial fn $f_{M,k}$

f is WHILE computable if there is M s.t. $f = f_{M,k}$

__Theorem__ (w/o proof)

Let $f: \mathbb{N}^k \dashrightarrow \mathbb{N}$ be a function.
Then TFAE:

(i) f is computable
(ii) f is partial recursive
(iii) f is Turing computable
(iv) f is while computable.

The confluence of so many distinct and not at all obviously equivalent notions means that we have identified a STABLE CONCEPT

The CTT is not a mathematical statement. It is an interpretation of an informal pre-mathematical notion!

The Church-Turing Thesis. The mentioned equivalent formal concepts of computability describe the informal notion of computability successfully: any reasonable attempt to describe the informal notion of computability will lead to a formal notion that is equivalent to the ones we have described.

RETURN TO

X is a type 0 language

$$\Longleftrightarrow$$

X is r.e.

[Recap. Already proved $\Longrightarrow$ in Lecture XXI.]

Proof SKETCH of $\Longleftarrow$:

Suppose M is a TM computing ψ_X:

$$q_S \square w \square \xrightarrow[\text{computation}]{M} \square a \square$$

+ the halt state

$$q_H \square a \square$$

This is a rewrite system with the rule described earlier transforming

$$q_S \square w \square \text{ to } q_H \square a \square$$

Grammar starts from S, produces $q_H \square a \square$, does all Turing instructions backwards.

When q_S is seen, it deletes everything but w.

q.e.d.

Solving decision problems

Using Church's thesis to define the informal notion of algorithm, we can give precise statements for our decision problems.

First encode grammars by a function code in such a way that for each $w \in \mathbb{N}$ there is a G s.t. $code(G) = w$. Let's write G_w for this.

WORD PROBLEM $\{(v,w) ; w \in \mathcal{L}(G_v)\}$

EMPTINESS PROBLEM $\{ w ; \mathcal{L}(G_w) = \emptyset \}$

EQUIVALENCE PROBLEM $\{(w,v) ; \mathcal{L}(G_w) = \mathcal{L}(G_v)\}$

We say such a problem is **SOLVABLE** if the set is computable.

Theorem Word problem for type 0 grammars is unsolvable.

Proof. $W = \{(w,v);\ w \in \mathcal{L}(G_v)\}$

Need to show that this is not computable.

[Note that W looks very much like $\mathbb{K}_0$.]

Suppose it is.

Define
$$f(w) := \begin{cases} \uparrow & \text{if } w \in \mathcal{L}(G_w) \\ a & \text{if } w \notin \mathcal{L}(G_w). \end{cases}$$

f is computable, thus $\mathrm{dom}(f)$ is c.e.

So find grammar G s.t.

$$\mathcal{L}(G) = \mathrm{dom}(f)$$

Let d be such that $G = G_d$.

$$d \in \mathcal{L}(G_d) \iff d \in \mathrm{dom}(f) \iff d \notin \mathcal{L}(G_d)$$

q.e.d.