IX

COMPUTABILITY
DECIDABILITY
INCOMPLETENESS

CD1

Ninth Lecture
13 JUNE 2023

Lecture VIII :

DECIDABILITY = COMPUTABILITY

$A \subseteq W^{\mathbb{N}}$ is a decidable problem
(can be solved by an algorithm)

$\Longleftrightarrow$

A is a computable set

[Church-Turing thesis]

3. $\{(w,v);\ w \in W_v\} = \{(w,v);\ w \in \mathrm{dom}(\{v\}_1)\}$
$= \{(w,v);\ \{v\}_1(w)\downarrow\}$
$= \mathbb{K}_0$.

So this is not computable, so not decidable

In words : The word problem for c.e. sets is not decidable.

4. $\{w;\ W_w = \emptyset\} = \mathrm{Emp}$

This is not computable by Rice's Theorem.

In words: The emptiness problem for c.e. sets is not decidable.

5. Equivalence problem

$\{(w,v);\ W_w = W_v\}. = \mathrm{Eq}$

We show that

$\mathrm{Emp} \leq_m \mathrm{Eq}$.

Let e be s.t. $W_e = \emptyset$.

The function $h: w \longmapsto (w,e)$ can be performed by a register machine.

So if Eq is computable, i.e., χ_{Eq} is a computable function, then so is

$$\chi_{\mathrm{Eq}} \circ h = \chi_{\mathrm{Emp}},$$

so Emp is computable. Contradiction!

Therefore: The equivalence problem for c.e. sets is not decidable.

Note These problem require a collection of sets for which we solve the problem.

Note If you restrict the problems to a subset of the set of c.e. sets, there is no reason to assume that the problems remain undecidable.

SILLY EXAMPLE:

Decide whether $W_w = \emptyset$ for any $w \in Emp$.

is trivially decidable, since the answer is always yes.

Some less silly examples

[The corresponding models of computation can all be found in typical undergrad textbooks of theoretical computer science.]

(1) Finite automata

A finite automaton is a graph with a transition relation whose a token moves on the graph as it reads the word symbol-by-symbol.

At the end, after $|w|$ many steps, it accepts or rejects the word based on where token ended up.

Clearly, given an automaton A and a word w, we can determine in $|w|$ steps, whether w is accepted by A.

So: the word problem for finite automata is decidable.

For emptiness, the emptiness problem for finite automata is essentially the problem of REACHABILITY (of accepting vertices in the graph) which is known to be algorithmically solvable.

For equivalence, this is a nontrivial prof:

1. We show that for finite automata, there is (up to isomorphism) a unique minimal automaton.

2. We show that there is an algorithm to calculate from a given automaton A its unique minimal automaton $\hat{A}$. ["table filling algorithm"]

3. Therefore A and B produce the same set iff $\hat{A} = \hat{B}$. [which can be checked algorithmically by 2.]

---

(2) Rewrite systems

GENERATIVE GRAMMARS

Fix a set of rewrite rules and say a word w is accepted if there is a finite path following the rewrite rules that produces w.

Note. With no restrictions, rewrite systems just generate the c.e. sets.

If you demand the rules are, say, CONTEXT-FREE (i.e., the application of the rule does not depend on the context) you get a smaller collection of sets [yet bigger than those we get by finite automata].

Here:

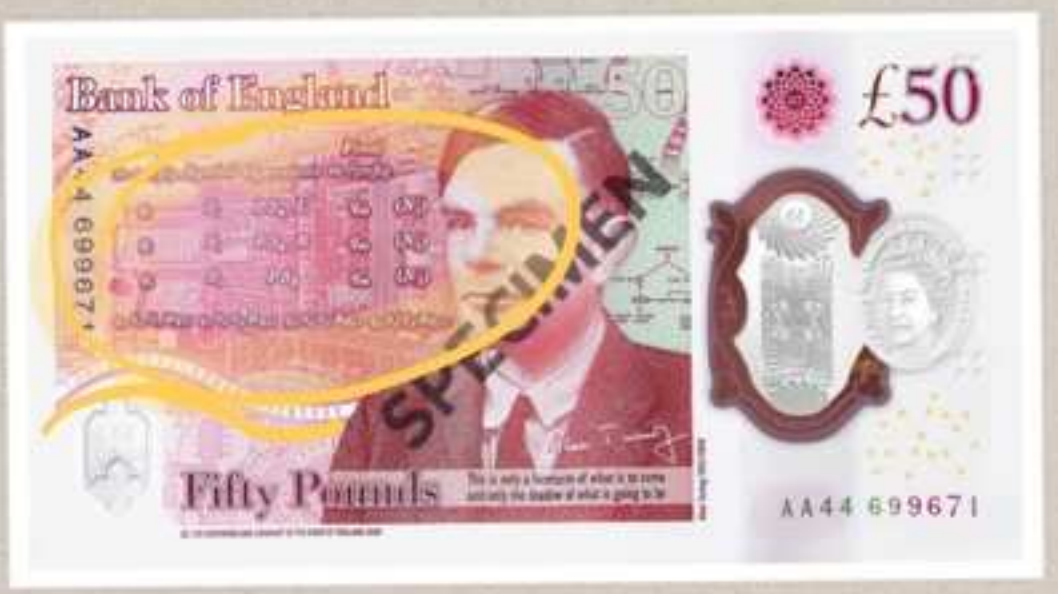
(a) the word problem for context-free grammars is decidable;

(b) the emptiness problem for context-free grammars is decidable;

(c) the equivalence problem for context-free grammars is not decidable.

Reference: The lecture notes of the course AUTOMATA & FORMAL LANGUAGES on moodle.

FROM NOW ON we are going to use some Mathematical concepts of the course Logic and Set Theory:



THE ENTSCHEIDUNGS-PROBLEM

For 1., we need a representation of the formulas $\varphi \in \mathcal{L}$ by words $w \in W$.
If we have this, we can w.l.o.g. assume $\mathcal{L} \subseteq W$.
Then $\{\varphi \mid \vdash \varphi\} \subseteq W$,
so it's a decision problem.

More about the ENTSCHEIDUNGS-PROBLEM in lecture IX.

Reminder

Gödel's Completeness Theorem:

$$\vdash \varphi \iff \models \varphi$$

$\vdash \varphi$: φ is provable — there is a finite seq. of formulas $\varphi_0 \dots \varphi_n$ s.t. all are either axioms of logic or follow from previous ones, and $\varphi_n = \varphi$

$\models \varphi$: φ is semantically valid — for all models M, $M \models \varphi$.

In order for

$$\{\varphi;\ \vdash\varphi\} \subseteq W,$$

we need to make sure that all symbols in φ occur as symbols in Σ.

So if the logical and non-logical symbols of $\mathcal{L}$ are all in Σ, formulas are words in W. [Not just represented by.]

Problem Logical languages have infinitely many variables and no bound whatsoever on the number of nonlogical symbols.

Solution 1 If we show that the Entscheidungsproblem is unsolvable for any finite restriction of $\mathcal{L}$, then any bigger problem cannot be decidable either.

[This would only work if the proof works with some finite fragment.]

Note: Our proof will work with finitely many non-logical symbols, but might use unboundedly many variables.

Solution 2

[Remember the similar problem when we encoded machines as words and we needed infinitely many symbols for states.]

Use 0, 1 and encode variable in binary, i.e.

V0 ⟶ stands for v_0

V1 ⟶ v_1

V10 ⟶ v_2

V11 ⟶ v_3

etc.

Summary We shall make Σ big enough to include 0, 1, all logical connectives $(\exists, \forall, \wedge, \vee, \neg, (,))$, and a finite number of non-logical symbols to be determined in the proof.

Proof idea

We reduce the undecidability of $\{\varphi_i \vdash \varphi\}$ to the undecidability of $\mathbb{K}$ by providing a reduction function witnessing

$$\mathbb{K} \leq_m \{\varphi_i \vdash \varphi\}$$

[i.e., $h : \mathbb{W} \longrightarrow \mathbb{W}$
$x \in \mathbb{K} \longleftrightarrow h(x) \in \{\varphi_i \vdash \varphi\}$,

i.e. $w \longmapsto \varphi_w$ s.t.

$$\varphi_{w,1}(w)\downarrow \iff \vdash \varphi_w.$$

Idea:

Formula φ_w expressed "The machine coded by w halts on input w."

Note The work done in Lecture II is a formalisation of

"the machine coded by w halts at input w".

With some (not much) extra work, we could have formula σ_w s.t.

$$\varphi_{w,1}(w)\downarrow \iff \sigma_w \text{ is true}$$

Note that this immediately gives us

$$K \leq_m \{\varphi : \varphi \text{ is true}\}$$

by the reduction function

$$w \longmapsto \sigma_w.$$

Since Gödel's incompleteness theorem will show that

$$\{\varphi : \varphi \text{ is true}\} \neq \{\varphi : \vdash \varphi\},$$

this is not an answer to the Entscheidungsproblem.

Need to work a little harder!

Lecture II

PART C : COMPUTABILITY

§C.1 REGISTER MACHINES

FROM LECTURE #1

START
HALT

INSTRUCTION
STATES
SYMBOLS

natural numbers

UPPER REGISTER INDEX:
largest k occurring in a RM.

4.1 Register machines

Let Σ be an alphabet and Q a non-empty finite set whose elements we shall call states. A triple of the form

$$(k, a, q') \quad (k, a, q', q'') \quad (k, q', q'')$$

is called a Σ, Q instruction. For suitable Σ, Q and k these are written

$k(a, q')$	['add']
$k(a, q', q'')$	['check']
$k(-, q', q'')$	['check']
$k(q', q'')$	['remove']

Instruction	Interpretation
$k(a, q')$	'Add the letter a to the contents of register k and go to state q''
$k(a, q', q'')$	'Check whether the last letter in register k is a; if so, go to state q'; otherwise, go to state q'''
$k(-, q', q'')$	'Check whether register k is empty; if so, go to state q'; otherwise, go to state q'''
$k(q', q'')$	'Check whether register k is empty; if so, go to state q'; otherwise, remove the final letter of its contents and go to state q'''

Definition A Σ-register machine is a triple (Σ, Q, P) where Σ is an alphabet, Q is a non-empty finite set of states, P is a function with domain Q s.t. for all $q \in Q$, $P(q)$ is an instruction.

Also assume that $q_y \neq q_n \in Q$.
YEAH STATE NOPE STATE

If C is a configuration and $M = (\Sigma, Q, P)$ is a RM, we say that **M transforms C to C'** if

Case 1. If $P(q) = +(k, a, q')$ and $C'' = (q', w_0, \ldots, w_{k-1}, w_k a, w_{k+1}, \ldots, w_m)$.

Case 2. If $P(q) = ?(k, a, q', q'')$,

Subcase 2a. $w_k = wa$ for some w and $C'' = (q', w_0, \ldots, w_m)$ or

Subcase 2b. $w_k \neq wa$ for any w and $C'' = (q'', w_0, \ldots, w_m)$.

Case 3. If $P(q) = ?(k, \varepsilon, q', q'')$,

Subcase 3a. $w_k = \varepsilon$ and $C'' = (q', w_0, \ldots, w_m)$ or

Subcase 3b. $w_k \neq \varepsilon$ and $C'' = (q'', w_0, \ldots, w_m)$.

Case 4. If $P(q) = -(k, q', q'')$,

Subcase 4a. $w_k = \varepsilon$ and $C'' = (q', w_0, \ldots, w_m)$ or

Subcase 4b. $w_k = wa$ for some a and $C'' = (q'', w_0, \ldots, w_{k-1}, w, w_{k+1}, \ldots, w_m)$.

$C = (q, w_0, \cdots, w_m)$.

COMPUTATION SEQUENCE

recursive definition based on the instruction

adding a symbol

removing a symbol

Observe
Transforming C to C' is a function on the set of configurations.

"HALTING"

Our logical language:

		INTERPRETED AS
NAT	unary predicate	$\mathbb{N}$
S	unary function	$n \longmapsto n+1$
Z	constant	0

1. NAT(Z)
2. $\forall x$ (NAT(x) $\longrightarrow$ NAT(S(x)))

SYMBOL	unary predicate	Σ	3. $\forall x \forall y$ SEQWORDS(x) $\wedge$ NAT(y) $\longrightarrow$ WORD(PROJ(x,y))
WORD	unary predicate	W	
SEQWORDS	unary predicate	$(W)^*$	
LENGTH	unary function	$\lvert\cdot\rvert$	4. $\forall x$. WORD(x) $\longrightarrow$ NAT(LENGTH(x))
EMPTY	constant	ε	5. WORD(EMPTY) $\wedge \neg$ SYMBOL(EMPTY)
LAST	unary function	last symbol in w	6. $\forall x$ WORD(x) $\longrightarrow$ SYMBOL(LAST(x)) $\vee$ LAST(x) = EMPTY
ADD	binary function	$w \longmapsto wa$	
REMOVE	unary function	$wa \longmapsto w$	
PROJ	binary function	$(w_1, w_2) \longmapsto w_e$	7. $\forall x \forall y$ WORD(x) $\wedge$ SYMBOL(y) $\longrightarrow$ WORD(ADD(x,y))

8. $\forall x$ WORD(x) $\longrightarrow$ WORD(REMOVE(x))

9. $\forall x \forall y$ WORD(x) $\wedge$ SYMBOL(y) $\longrightarrow$ LAST(ADD(x,y)) = y

10. $\forall x \forall y$ WORD(x) $\wedge$ SYMBOL(y) LAST(x) = y $\longrightarrow$ ADD(REMOVE(x),y) = x .

STATE
INST
CONFIG
} unary relations

START
HALT
} constants

+
?1
?2
−
} unary relations

IREG
ISYM
IST1
IST2
} unary functions

CSTATE
CCONTENT
} unary functions

states
instructions
configurations
} types

11. $\forall x$ INST(x) $\leftrightarrow$ +(x) $\vee$?1(x) $\vee$?2(x) $\vee$ −(x)

qS
qH

12.-15. $\forall x$ +(x) $\rightarrow$ $\neg$?1(x) $\wedge \neg$?2(x) $\wedge \neg -(x)$
[and similar for ?1, ?2, −]

type of instruction

information given in instruction

16. $\forall x$ INST(x) $\rightarrow$ NAT(IREG(x)) $\wedge$ SYMBOL(ISYM(x)) $\wedge$ STATE(IST1(x)) $\wedge$ STATE(IST2(x))

state/content of config.

17. $\forall x$ CONFIG(x) $\rightarrow$ STATE(CSTATE(x))
18. $\forall x$ CONFIG(x) $\rightarrow$ SEQWORDS(CCONTENT(x))
19. STATE(START) $\wedge$ STATE(HALT)

MSTATES
MPROG
IN
ASS

unary function
unary function
binary relation
binary function

Q
P
$\in$
function appl.

20. $\forall x \forall y$ IN(y, MSTATES(x)) $\rightarrow$ STATE(y)

21. $\forall x \forall y \forall z$ y = MPROG(x) $\wedge$ IN(z, MSTATES(x)) $\rightarrow$ INST(ASS(y, z))