Diagonalization with Programs P in Java ~~or~~ etc.

Let $e(P)$ stand for any of:
{ the source code of P, which the text would call $\langle P \rangle$
{ compiled object code of P
the final machine code on some platform.

Define $D_{Java} = \{ \underbrace{e(P)}_{q} :$ the string $\underbrace{e(P)}_{q}$ is not accepted by $\underbrace{P}_{Q} \}$.
i.e. $P(e(P))$ does not (halt and output 1)

Theorem: There is no Java program Q such that $D_{Java} = L(Q)$.

Proof: Suppose we had such a Q. Let $q = e(Q)$. Then:

$$q \in D_{Java} \iff Q \text{ accepts } q \quad \text{by } L(Q) = D_{Java}$$
$$\iff Q \text{ does not accept } q \quad \text{by defn of } q \in D_{Java}.$$

This makes a stmt equivalent to its negation, which is never allowed (In OS terms it's a "Logic System Rollback" — and rolls back to: There is no such Q.) This contradiction shows Q cannot exist. ⊠

Corollary: The language D_{Java} is not Turing-recognizable i.e. not c.e. (synonym: not in RE)

Moreover: Any language D' that enlarges D_{Java} by adding strings not in the range of e — i.e. adding invalid codes — is also not c.e. If $e(P) = \langle P \rangle$ (source code), ditto if you add non-valid programs.

Back to Det^c Turing Machines. (Det^c = deterministic, Def^n = Definition)

$D = D_{TM} = \{\langle M \rangle : M$ does not accept $\langle M \rangle\}$ is not c.e.

$D' = \{ x \in ASCII^*$: either x is a valid TM code $\langle M \rangle$ and M does not accept $\langle M \rangle$, or x is not a valid code $\}$.

is not c.e. either.

Helpful Side Node: Define $\underline{\underline{K_{TM}}} = \{\langle M \rangle : M$ does accept $\langle M \rangle.\}$

(standard name K but not in text)

Then K_{TM} is literally the complement of D', "morally" the complement of D_{TM}

Theorem: K_{TM} and A_{TM} are Turing recognizable languages but neither one is decidable.
(text 4.11)

Proof: If K_{TM} were decidable, then its complement would be decidable too. But its complement is literally D', which is not even c.e.

Now A_{TM} is the language "$L_{A_{TM}}$" of the problem sharing the name A_{TM}

A_{TM}: INST: A Turing machine M and a string $x \in \Sigma^*$
QUES: Does M accept x?

K_{TM}: INST: A Turing machine M and the particular string $\langle M \rangle$
QUES: Does M accept $\langle M \rangle$?

K_{TM} is a restriction of A_{TM}

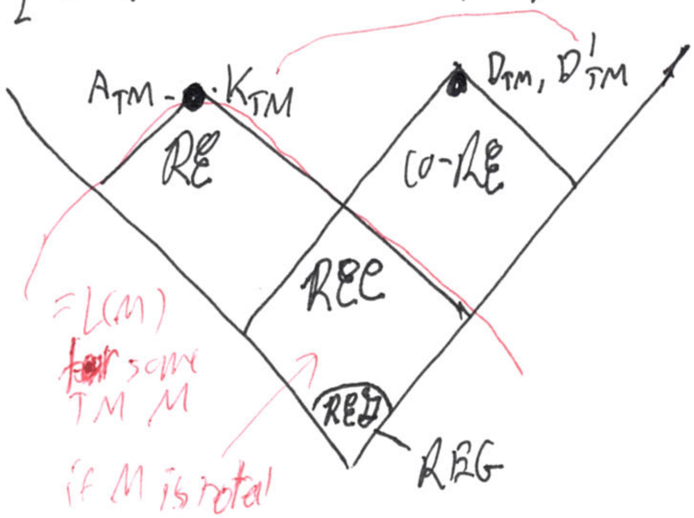
If A_{TM} were decidable, then "ipso-facto" K_{TM} would be decidable too.
But K_{TM} is undecidable, so A_{TM} is undecidable ($\equiv$ the entire text's proof)
However, A_{TM} is recognizable: $A_{TM} = L(U)$ for some/any universal TM U.
Hence the special case K_{TM} is recognizable the same way. $\boxtimes$

"Mapping Out Classes" (Idea): Use left-right reflection for complements.

$\underline{REC} = \{ L \subseteq \Sigma^* : L \text{ is decidable} \}$

$\underline{RE} = \{ L \subseteq \Sigma^* : L \text{ is recognizable} \}$

$\underline{co\text{-}RE} = \{ \text{complements of languages in } RE \}$, i.e. $\{ \tilde{L} : L \in RE \}$.



← Visually note that the complement $\tilde{A}_{TM}$ of the A_{TM} language is also over here
∴ $\tilde{A}_{TM}$ is not r.e. (Cor. 4.23)

Theorem: $RE \cap co\text{-}RE = REC$ i.e. For all languages L,
(4.22 in text)

$L \in RE \wedge L \in co\text{-}RE \iff L \in REC$.

Both L and $\tilde{L}$ are $\underbrace{\text{r.e.}}_{\text{recognizable}}$ $\iff$ L is decidable, i.e.

There exist TMs M and M' such that $L = L(M) \wedge \tilde{L} = L(M')$ $\iff$ there is a total TM T such that $L = L(T)$.

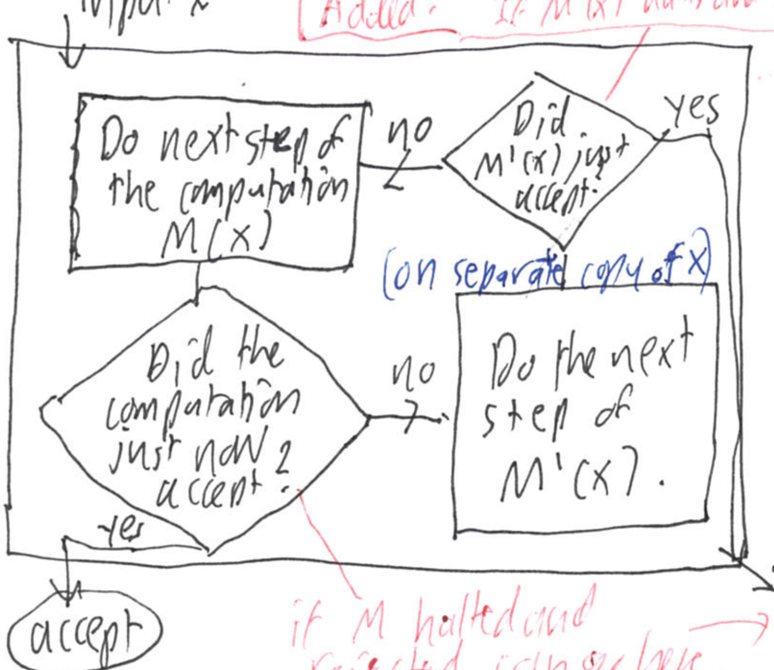
Proof: $\Leftarrow$ is easy: If we have a total TM $T = (Q, \Sigma, \Gamma, \delta, B, s, q_{acc}, q_{rej})$ then the complemented machine $T' = (Q, \Sigma, \Gamma, \delta, B, s, q_{rej}, q_{acc})$ really does accept $\sim L(T)$, i.e. $\tilde{L}$. So take $M = T$ and $M' = T'$.

$\Rightarrow$ is harder. Let M and M' be given. The goal is to build a total TM T s.t. $L(M) = L = L(T)$. Sketch as a flowchart:

[Added: If M'(x) halts and rejects you can go to accept but not needed now — it will happen later when M accepts.]

TM T:

(you can think of T being in any language / programming)



If M' accepts x then $x \in \tilde{L}$ so T should reject.

Key point: Since $L(M') = \widetilde{L(M)}$, for every $x \in \Sigma^*$, either: M accepts x (so T accepts x) or M' accepts x (so T rejects)

[But this and the other note in red are not needed — T can just "null-step" M.]

Upshot: $T(x)$ always halts and either accepts or rejects, so T is total and $L(T) = \{x : M(x) \text{ accepts}\} = L(M)$. ☒

FYI (see ps9 reading too): Which classes are closed under ~?

Define $\mathcal{CFL} = \{L : L = L(G)$ for some CFG $G\} = \{L : L = L(N)$ for some NPDA $N\}$.

DCFL = $\mathcal{DCFL} = \{L : L = L(M)$ for some DPDA $M\}$.

Writing $\mathcal{CFL}$ curly helps tell a class of languages apart from a single language — like a single CFL. But get used to non-curly CFL in both senses DCFL

$L_3 = \{a^n b^n c^n : n \geq 0\}$ is not a CFL, but its complement is

$\{a^n b^n : n \geq 0\}$ is a DCFL that is not regular

The switch F and $Q \setminus F$ or switch q_{acc} and q_{rej} trick works on DFAs and DPDAs, so DCFL is closed under "~" too.

REG is closed under ~ since every NFA converts to a DFA.

$\tilde{L}_3$ is accepted by an NPDA but not by any DPDA: it is not a DCFL. ☒