Diagonalization with Programs $P$ in Java and etc.

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

Define $D_{Java} = \{ e(P) : \text{the string } e(P) \text{ is not accepted by } P \}$

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

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

$\neg y \in D_{Java} \iff Q \text{ accepts } y$ by $L(Q) = D_{Java}$

$\iff Q \text{ does not accept } y$ by def of $y \in D_{Java}$.

This makes a start 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 — it also is not c.e. If $e(P) \neq L(P)$ (source code), ditto if you add non-valid programs.
Back to Def Turing Machines (Def = deterministic) Def \[=\] Definition

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

\[D' = \{x \in \text{AS(\text{11})}^*: \text{either } x \text{ is a valid TM code } \langle M \rangle \text{ and } M \text{ does not accept } \langle M \rangle \text{ or } x \text{ is not a valid code}\} \]

Helpful Side Note: Define \[K_{TM} = \{\langle M \rangle: M \text{ does accept } \langle M \rangle \}\]

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 are decidable.

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 \[L_{TM}\] of the problem shapely the name \[A_{TM}\]

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

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

\[K_{TM}\] is a restriction of \[A_{TM}\]

If \[A_{TM}\] were decidable, then "easy-fact" \[K_{TM}\] would be decidable too. But \[K_{TM}\] is undecidable, so \[A_{TM}\] is undecidable (\(\equiv\) the entire text is proof).

However, \[A_{TM}\] is recognizable: \[A_{TM} = L(U)\] for some universal TM \(U\).

Hence the special case \[K_{TM}\] is recognizable the same way. \(\square\)
Mapping out classes: (Idea): Use left-right reflection for complement.

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

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

$\text{co-RE} = \{ \text{complement of languages in RE} \} = \{ \overline{\text{L}} : \text{L} \in \text{RE} \}$

Visually note that the complement ATM of the ATM language is also over here.

$\vdash \text{ATM is not c.e. (Cor. 4.23)}$

Theorem: $\text{RE} \cap \text{co-RE} = \text{REC}$, i.e. for all languages $L$,

$L \in \text{RE} \land \overline{L} \in \text{co-RE} \iff L \subseteq \text{REC}$.

Both $L$ and $\overline{L}$ are RE $\iff L$ is decidable, i.e. recognizable.

There exist TMs $M$ and $M'$ such that $L = L(M) \land \overline{L} = L(M')$.

Proof: $\Leftarrow$ is easy: If we have a total TM $T = (Q, \Sigma, \Gamma, \delta, S, \epsilon, q_0, F)$
then the complemented machine $T' = (Q, \Sigma, \Gamma, \delta, S, \epsilon, q_0, \overline{F})$ really does accept $\overline{L(T)}$, i.e. $\overline{L}$. 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.
TM T:
(you can think of T being in any language programming)

If M halted and rejected, set y up lone
But this and the other note in red are not needed - T can just "null step" M

If M halted and accept, set y to blank

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

Exercise (see PS 9 reading too): Which classes are closed under \( \sim \)?

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

Define \( \text{DCFL} = \text{CFL} = \{ L : L = L(M) \text{ for some DPDA M} \}. \)

Writing 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.

The switch F and \( \Box \) F or switch race and tre 

trick works on DFAs and

DFA s, so DCFL is closed under \( \sim \) too.

REG is closed under \( \sim \) since every NFA can reach to a DFA.

\( L_7 \) is accepted by an NPODA but not by any DFA, it is not a DCFL.