Consider Java programs that take ASCII input from System.in (choose and fix one) and accept by either printing "1" and halting. Then we can define:

\[ D_{\text{Java}} = \{ \langle P \rangle : P \text{ is a legal Java program that does not accept } \langle P \rangle \} \]

for Quixotic.

**Theorem:** There does not exist a Java program \( Q \) s.t. \( D_{\text{Java}} = L(Q) \).

**Proof:** Suppose \( Q \) exists. Then for any \( x \in \langle Q \rangle \), \( Q \) accepts \( \langle Q \rangle \iff \langle Q \rangle \in D_{\text{Java}} \). By \( \text{Def} \) of \( D_{\text{Java}} \), \( D_{\text{Java}} \) compiles and yields \( Q \) compiler. \( \iff Q \) does not accept \( \langle Q \rangle \) by def. of \( D_{\text{Java}} \).

There is no way to escape this contradiction, \( \therefore \) \( Q \) does not exist. \( \Box \)

**I.e.,** \( D_{\text{Java}} \) is not "Java-recognizable." By interconvertibility of Java and Turing machines, \( D_{\text{Java}} \) is not Turing-recognizable either. Similarly, \( D_{\text{TM}} \) is not "Java-recognizable" either. '{"Turing acceptable\} -> computable encodable in R.E. 
Hence neither \( D_{\text{TM}} \) nor \( D_{\text{Java}} \) is
However, \( \neg D_{Java} = \exists x: x \text{ does not compile in Java, or} \)

\( \neg K_{Java} \Rightarrow \exists x: x \text{ compiles to } P \text{ such that } P \text{ does accept } x \).

\( K_{TM} = \{ x: x \text{ is a legal Turing Machine that accepts } x \} \).

\( O: K \text{ is r.e. but not decidable. } D \text{ is not even r.e./c.e.} \)

One other standard notation uses a fixed subscript "TM" or "Java".

"Enumeration = M_0, M_1, M_2, M_3, \ldots" of Turing Machines, or a "Gödel Numbering" P_0, P_1, P_2, P_3, \ldots of Java programs. Then:

\( D_{Java} = \{ i: P_i \text{ does not accept } i \} \text{ if number or string.} \)

\( O_{TM} = \{ e: M_e \text{ does not accept } e \} \text{ if number or string.} \)

\( K_{TM} = \{ e: M_e \text{ does accept } e \} \text{ if number or string.} \)

\( A_{TM} = \{ (M, x): \text{ The TM } M \text{ accepts } x \} \)

Theorem: \( \neg D_{TM} \) is r.e. but not decidable.

Proof: First, \( \neg D_{TM} \) does equal \( L("Turing\ Chip") \) if a Java program

But if it were decidable by a program \( R \), then

\( e \in K \iff R \text{ accepts } (e, e) \) so that

\( K_{TM} \) would be decidable too. But we showed that false. \( \Box \)

Intuitively, \( K_{TM} \) is a "special case restriction of the A_{TM} \) problem.
**Definition**: Define \( A \) and \( B \) to be \( \text{para-complement} \) if \( A \cap B = \emptyset \) and \( A \cup B \) is decidable. Call it a regular para-complement if \( (A \cup B) \) is decidable and \( A \cup B \) is regular.

**Examples**: DJam and KJam are literally para-complements \( U = \{ x \mid x \text{ compiles in Java} \} \) and literal complements are \( \text{"para" with } U = \Sigma^* \text{, which is regular.} \)

- \( \{ a^m b^n c^n \mid m \leq n \text{ or } n \geq m \} \) and \( \{ a^m b^n c^k \mid m = n \text{ or } k \neq 0 \} \) are regular para-complements with \( U = a^*b^*c^* \) as an HW.

**Theorem**: If \( A \) and \( B \) are para-complements (for a decidable \( U \))

- \( A \) is decidable \( \iff \) \( B \) is decidable.
- \( \text{if } A \text{ and } B \text{ are r.e., then both are decidable.} \)

**Proof**: Take a total TM \( M_u \) st. \( L(M_u) = U \), i.e. \( M_u \) halts for all inputs.

- Suppose \( A \) is decidable. Then we have a total TM \( M_a \) st. \( L(M_a) = A \).

**Flowchart convention**: Total routines go in solid boxes, \( \square \) in boxes with inputs, \( \diamond \) in boxes with outputs.

**Flowchart**

- \( \text{Input } x \in \Sigma^* \)

  - **Does \( u \) accept \( x \)?**
    - no: reject \( x \)
    - yes: \( M_u \) accepts \( x \)

  - **Does \( M_u \) accept \( x \)?**
    - no: reject \( x \)
    - yes: accept \( x \)

**Part (b)**: We can take TMs \( M_a \) and \( M_b \) st. \( L(M_a) = A \) and \( L(M_b) = B \).

- They need \\( \text{not } \) be total. However, the act of stepping each one step ahead in a computation does terminate, so can go in solid box.
When \( U = \Sigma' \) this says: \( A \text{ language } A \text{ is decidable} \Rightarrow \overline{A} \text{ is decidable} \)

\[ \text{REC is closed under complements.} \]

\[ \text{If } A \text{ and } \overline{A} \text{ are r.e., then both are decidable.} \]

\[ \Rightarrow \text{RE n coRE = REC (REC and DEC)} \]

The Halting Problem:

**INST:** A TM \( M \) and an input \( x \) to \( M \).

**Ques:** Does \( M(x) \downarrow \)? (\( \downarrow \equiv \text{halts}, \ T \equiv \text{does not halt} \))

The language of this problem can be called \( HP_m \).

**Thm:** \( HP_m \) is r.e. but undecidable.

**Proof:** \( HP_m \) would be \( L("\text{Turing Kit}\)" If Turing Kit accepted

If \( M \) had a decider \( R \) for \( HP_m \),

then we would get one for \( A_{TM} \) by modifying a given \( M \) to \( M' \) that has a loop \( \overline{R}(1,0,5) \) at its rejecting state.

\[ M'(x) \downarrow \Rightarrow M(x) \text{ accept, so } R \text{ would decide } A_{TM} \text{ too.} \]

**Added:** What we have actually done is reduce \( A_{TM} \) to \( HP_m \).

We can reduce the other way if we make the new TM accept if and when the original TM halts (by redirecting arcs to \( \delta \) to go to state again.) Reductions are the topic of Thursday's lecture.