Theorem: For every program $P$ that can be written in any high-level language that has ever been devised, we can find a Turing machine $M$ that is equivalent to $P$ in all of several senses:

1. If we consider the language $L(P)$ of your $P$ to be $\exists x \in \text{As} : \text{As}(x) \iff P(x)$ exists with error states, then $L(M) = L(P)$.

2. If $P$ computes a numerical function, then $M$ can be a TM transducer that computes the same fn.

3. $M$ emulates every "step" of $P$ by step(s) of its own.

Consequence for us:
- TMs are a generally relevant hard-and-fast model for assessing real programs.
- It is henceforth OK to describe TMs in pseudocode.

Theorem: For every NTM $N$ we can build a TM $M$ s.t. $L(M) = L(N)$.

Proof:
- Write in Java a program $P$ that tries all possibilities for $N$ on a given $X$ and accepts $P$ when an accepting config of $N$ on $X$ is found.
- Convert $P$ to TM $M$. 

---

Theoröm: For every program $P$ that can be written in any high-level language that has ever been devised, we can find a Turing machine $M$ that is equivalent to $P$ in all of several senses:

1. If we consider the language $L(P)$ of your $P$ to be $\exists x \in \text{As} : \text{As}(x) \iff P(x)$ exists with error states, then $L(M) = L(P)$.

2. If $P$ computes a numerical function, then $M$ can be a TM transducer that computes the same fn.

3. $M$ emulates every "step" of $P$ by step(s) of its own.

Consequence for us:
- TMs are a generally relevant hard-and-fast model for assessing real programs.
- It is henceforth OK to describe TMs in pseudocode.

Theorem: For every NTM $N$ we can build a TM $M$ s.t. $L(M) = L(N)$.

Proof:
- Write in Java a program $P$ that tries all possibilities for $N$ on a given $X$ and accepts $P$ when an accepting config of $N$ on $X$ is found.
- Convert $P$ to TM $M$. 

---
The classes \( \mathcal{RE} = \{ L : L = L(M) \text{ for some DTM } M \} \) and \( \mathcal{REC} = \{ L : L \leq L(M) \text{ for some total DTM } M \} \) are the same with NIMs or Java or any other known program concept.

Hence the notions of recognizable and decidable apply generally to (i.e., languages) decision problems (and also functions).

Some examples in "INSTANCE - QUESTION" format:

**Acceptance Problem for DFAs (A DFA)**

**INST:** A DFA \( M \), an argument \( x \) to \( M \).

**Gives:** Does \( M \) accept \( x \)?

Easily Decidable — Just run \( M(x) \).

The language \( L = L_{A DFA} \) of this problem is \( \{ \langle M, x \rangle : M \text{ accepts } x \} \).

Note: \( M = (Q, \Sigma, S, s, F) \) say \( \Sigma = \{ a, b \} \) a single string \( w \) over ASCII.

\( L(M) \) itself is over \( \Sigma \).
But \( L_{A DFA} \) is over ASCII since it handles \( M \)'s for any \( \Sigma \).

(Ultimately, theoretically, we blur all distinctions and consider all languages to be over the alphabet \( \Sigma = \{ 0, 1 \} \).

2. (Non-)Emptiness Problem for DFAs. (NE DFA)

**INST:** Just a DFA \( M = (Q, \Sigma, S, s, F) \), \( \Sigma = \{ a, b \} \) a single string \( w \) over ASCII.

**Gives:** Is there any \( x \in \Sigma^* \) s.t. \( M \) accepts \( x \). \( L(M) \neq \emptyset \)?
7. EMPTINESS FOR DFAs (\texttt{NE\textsc{DFA}}) These problems are \textit{equally decidable} since a \textit{yes} answer for one \(\equiv\) a \textit{no} answer to the other.

Thus: Both of these problems are decidable by Breadth First Search.

\texttt{NE\textsc{DFA}}: Given \(M, L(M) \neq \emptyset\) there exists a path from \(s\) to some final state \(f\).

We can code BFS in a program \(P\). \(P\) is total since BFS always terminates.

- Can convert \(P\) to a total \(\textsc{PSM}\) \(M_T\):
  \[ L(\texttt{NE\textsc{DFA}}) = L(M_T) \text{ and } M_T \text{ is total}, \text{ so } \texttt{NE\textsc{DFA}} \text{ is also called decidable}. \]

\[ M_T = (\Gamma, \Sigma, \Gamma, \delta, s, \{\text{acc, rej}\}) \]

Because \(M_T\) is total,
\[ M_T' = (\Gamma, \Sigma, \Gamma, \delta, s, \{\text{rej, acc}\}) \approx \text{ is likewise total, and } L(M_T') = L(\texttt{EDFA}). \]

Theorem: The class \(\text{REC}\) is closed under complements.

For every language \(L\) that is decidable, \(\overline{L}\) is also decidable.

For every total TM \(M_T\) we can build a total TM \(M_T'\) s.t. \(L(M_T') = L(M_T)\).

We can build \(M_T'\) such that for all \(x \in \Sigma^*\), \(M_T'\) accepts \(x \iff M_T\) does not accept \(x\).

Proof: Given \(M_T = (\Gamma, \Sigma, \Gamma, \delta, s, \{\text{acc, rej}\})\) build \(M_T'\) as above. \(M_T'\) is the same box with \texttt{acc} and \texttt{rej} swapped.
4. **Acceptance for TMs (Atm)**

**Inst:** A DTM $M$, an input $x \in \Sigma^*$ to $M$.

**Query:** Does $M$ accept $x$?

⚠️ Unlike as we did for DFAs, we can't decide this so easily by "Just run $M(x)$" because $M(x)$ might never halt.

Q: Is there a clever way like with DFAs for DFAs?

**An "Easier" Problem:** Self Acceptance for TMs ($K_{tm}$)

**Inst:** A DTM $M$.

**Query:** Does $M$ accept $\langle M \rangle$?

The complement of the problem is $\overline{K_{tm}}$.

**Inst:** A DTM $M$.

**Query:** Does $M$ not accept $\langle M \rangle$?

$L_{\overline{K_{tm}}}$, i.e., not the language $L_{\overline{K_{tm}}} = \{ \langle M \rangle : M$ does not accept $\langle M \rangle \}$.

This and $K_{tm}$ are "equally decidable", but neither is decidable.

In fact, $L_{\overline{K_{tm}}}$ is not even recognizable. **Proof:**

Suppose $L_{\overline{K_{tm}}}$ were recognizable. Then there would be a TM $Q$ such that $L(Q) = L_{\overline{K_{tm}}}$, i.e., $L(Q) = \{ \langle M \rangle : Q$ accepts $\langle M \rangle \} \cap L_{tm}$.

In particular, $Q$ accepts $\langle Q \rangle \iff \langle Q \rangle \in L_{\overline{K_{tm}}} \iff Q$ does not accept $\langle Q \rangle$.

In logic, a statement $S$ may never $\iff$ its negation $\neg S$. i.e., $Q$ cannot exist.

How text would name it, as a language without writing $L_{\overline{K_{tm}}}$?

Not to confuse with a DTM.