Computational Theory

Decidability

Curtis Larsen

Utah Tech University—Computing

Fall 2023

Adapted from notes by Russ Ross

Decidable Languages

Reading: Sipser §4.1.

Recognizable and Decidable

- $L(M) = \{w : M \text{ accepts } w\}$.
- L is **Turing-recognizable** if $L = L(M)$ for some TM M, i.e.
  - $w \in L \Rightarrow M$ halts on w in state q_{accept}.
  - $w \notin L \Rightarrow M$ halts on w in state q_{reject} OR M never halts (it “loops”).
- L is **decidable** if $L = L(M)$ for some TM M, i.e.
  - $w \in L \Rightarrow M$ halts on w in state q_{accept}.
  - $w \notin L \Rightarrow M$ halts on w in state q_{reject}.

Problems as Turing Machine Languages

- Encoding arbitrary objects: $\langle O \rangle$ is a suitable string representation of O.
- Problems can be encoded as languages:

  Let G be an undirected graph. We want to know if G is connected. We can state this as the language:

$$A = \{\langle G \rangle | G \text{ is a connected undirected graph } \}$$

- The Turing machine that processes input has a (usually) hidden step that decodes and verifies the encoding, rejecting if the verification fails.

A_{DFA}

- **Language**:
  $A_{\text{DFA}} = \{\langle B, w\rangle | B$ is a DFA that accepts input string $w\}$
- **Theorem 4.1**: A_{DFA} is a decidable language.

A_{DFA}

- **Language**:
  $A_{\text{DFA}} = \{\langle B, w\rangle | B \text{ is a DFA that accepts input string } w\}$
- **Theorem 4.1**: A_{DFA} is a decidable language.
- **Machine**: Let $M =$ "On input $\langle B, w\rangle$, where B is a DFA and w is a string:
  1. Simulate B on input w.
  2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."

A_{DFA}

- **Language**:
  $A_{\text{DFA}} = \{\langle B, w\rangle | B$ is a DFA that accepts input string $w\}$
- **Theorem 4.1**: A_{DFA} is a decidable language.
- **Machine**: Let $M =$ "On input $\langle B, w\rangle$, where B is a DFA and w is a string:
  1. Simulate B on input w.
  2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."
- **Proof Sketch**: The input is finite. Every step of the simulation consumes one input symbol. The simulation will terminate. M only needs to track the current input position, and the current state. When finished, final state's classification is all that is needed.

A_{NFA}

- **Language**:
  $A_{\text{NFA}} = \{\langle B, w\rangle | B$ is an NFA that accepts input string $w\}$
- **Theorem 4.2**: A_{NFA} is a decidable language.

A_{NFA}

- **Language**:
  $A_{\text{NFA}} = \{\langle B, w\rangle | B \text{ is an NFA that accepts input string } w\}$
- **Theorem 4.2**: A_{NFA} is a decidable language.
- **Machine**: Let $N =$ "On input $\langle B, w\rangle$, where B is an NFA and w is a string:
  1. Convert B to an equivalent DFA C, using the procedure from Theorem 1.39.
  2. Run M from Theorem 4.1 on input $\langle C, w\rangle$.
  3. If M accepts, *accept*; otherwise, *reject*."

A_{NFA}

- **Language**:
  $A_{\text{NFA}} = \{\langle B, w\rangle | B \text{ is an NFA that accepts input string } w\}$
- **Theorem 4.2**: A_{NFA} is a decidable language.
- **Machine**: Let $N =$ "On input $\langle B, w\rangle$, where B is an NFA and w is a string:
  1. Convert B to an equivalent DFA C, using the procedure from Theorem 1.39.
  2. Run M from Theorem 4.1 on input $\langle C, w\rangle$.
  3. If M accepts, *accept*; otherwise, *reject*."
- **Proof Sketch**: The procedure from Theorem 1.39 is finite. M is a decider. N must also be a decider.

A_{REX}

- **Language**:
  $A_{\text{REX}} = \{\langle R, w\rangle | R$ is a regular expression that generates string $w\}$
- **Theorem 4.3**: A_{REX} is a decidable language.

A_{REX}

- **Language**:
  $A_{\text{REX}} = \{\langle R, w\rangle | R$ is a regular expression that generates string $w\}$
- **Theorem 4.3**: A_{REX} is a decidable language.
- **Machine**: Let $P =$ "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:
  1. Convert R to an equivalent NFA B, using the procedure from Theorem 1.54.
  2. Run N from Theorem 4.2 on input $\langle B, w\rangle$.
  3. If N accepts, *accept*; otherwise, *reject*."

A_{REX}

- **Language**:
  $A_{\text{REX}} = \{\langle R, w\rangle | R$ is a regular expression that generates string $w\}$
- **Theorem 4.3**: A_{REX} is a decidable language.
- **Machine**: Let $P =$ "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:
  1. Convert R to an equivalent NFA B, using the procedure from Theorem 1.54.
  2. Run N from Theorem 4.2 on input $\langle B, w\rangle$.
  3. If N accepts, *accept*; otherwise, *reject*."
- **Proof Sketch**: The procedure from Theorem 1.54 is finite. N is a decider. P must also be a decider.

E_{DFA}

- **Language**: $E_{\text{DFA}} = \{\langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset\}$
- **Theorem 4.4**: E_{DFA} is a decidable language.
- **Machine**: Let $T_{\text{bad}} =$ "On input $\langle A \rangle$, where A is a DFA:
    1. For each string $w \in \Sigma^*$:
    2. Run M on $\langle A, w \rangle$.
    3. If M accepts, *reject*.
    4. If no w is accepted, *accept*."

E_{DFA}

- **Language**: $E_{\text{DFA}} = \{\langle A\rangle | A$ is a DFA and $L(A) = \emptyset\}$
- **Theorem 4.4**: E_{DFA} is a decidable language.
- **Machine**: Let T_{bad} = "On input $\langle A\rangle$, where A is a DFA:
  1. For each string $w \in \Sigma^*$:
  2. Run M on $\langle A, w\rangle$.
  3. If M accepts, *reject*.
  4. If no w is accepted, *accept*."
- **Proof Flaw?**: What is the flaw in this machine?

E_{DFA}

- **Language**: $E_{\text{DFA}} = \{\langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset\}$
- **Theorem 4.4**: E_{DFA} is a decidable language.

E_{DFA}

- **Language**: $E_{\text{DFA}} = \{\langle A\rangle | A \text{ is a DFA and } L(A) = \emptyset\}$
- **Theorem 4.4**: E_{DFA} is a decidable language.
- **Machine**: Let $T =$ "On input $\langle A\rangle$, where A is a DFA:
  1. Mark the start state of A.
  2. Repeat until no new states get marked:
  3. Mark any state that has a transition coming into it from any state that is already marked.
  4. If no accept state is marked, *accept*; otherwise, *reject*."

E_{DFA}

- **Language**: $E_{\text{DFA}} = \{\langle A \rangle | A$ is a DFA and $L(A) = \emptyset\}$
- **Theorem 4.4**: E_{DFA} is a decidable language.
- **Machine**: Let $T =$ "On input $\langle A \rangle$, where A is a DFA:
  1. Mark the start state of A.
  2. Repeat until no new states get marked:
  3. Mark any state that has a transition coming into it from any state that is already marked.
  4. If no accept state is marked, *accept*; otherwise, *reject*."
- **Proof Sketch**: The number of states is finite. The loop will terminate. T is a decider.

EQ_{DFA}

- **Language**:
  $EQ_{\text{DFA}} = \{\langle A, B\rangle | A$ and B are DFAs and $L(A) = L(B)\}$
- **Theorem 4.5**: EQ_{DFA} is a decidable language.
- **Machine**: Let $F_{\text{bad}} =$ "On input $\langle A, B\rangle$, where A and B are DFAs:
  1. For each $w \in \Sigma^*$:
  2.   Run M from Theorem 4.1 on $\langle A, w\rangle$ and $\langle B, w\rangle$.
  3.   If the results of the two runs differ, *reject*.
  4. If all results match, *accept*"

EQ_{DFA}

- **Language**:
  $EQ_{\text{DFA}} = \{\langle A, B\rangle | A$ and B are DFAs and $L(A) = L(B)\}$
- **Theorem 4.5**: EQ_{DFA} is a decidable language.
- **Machine**: Let $F_{\text{bad}} =$ "On input $\langle A, B\rangle$, where A and B are DFAs:
  1. For each $w \in \Sigma^*$:
  2. Run M from Theorem 4.1 on $\langle A, w\rangle$ and $\langle B, w\rangle$.
  3. If the results of the two runs differ, *reject*.
  4. If all results match, *accept*"
- **Proof Flaw?**: What is the flaw with this machine?

EQ_{DFA}

- **Language**:
  $EQ_{\text{DFA}} = \{\langle A, B\rangle | A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
- **Theorem 4.5**: EQ_{DFA} is a decidable language.

EQ_{DFA}

- **Language**:
  $EQ_{\text{DFA}} = \{\langle A, B\rangle | A$ and B are DFAs and $L(A) = L(B)\}$
- **Theorem 4.5**: EQ_{DFA} is a decidable language.
- **Machine**: Let $F =$ "On input $\langle A, B\rangle$, where A and B are DFAs:
  1. Construct DFA C to recognize $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$.
  2. Run T from Theorem 4.4 on $\langle C\rangle$.
  3. If T accepts, *accept*; otherwise, *reject*."

EQ_{DFA}

- **Language**:
  $EQ_{\text{DFA}} = \{\langle A, B\rangle | A$ and B are DFAs and $L(A) = L(B)\}$
- **Theorem 4.5**: EQ_{DFA} is a decidable language.
- **Machine**: Let $F =$ "On input $\langle A, B\rangle$, where A and B are DFAs:
  1. Construct DFA C to recognize $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$.
  2. Run T from Theorem 4.4 on $\langle C\rangle$.
  3. If T accepts, *accept*; otherwise, *reject*."
- **Proof Sketch**: The algorithms used to construct C are from chapter 1, and all are finite. This relies on the closure of regular languages under complement, intersection and union. Draw a sketch to illustrate the resulting set.

A_{CFG}

- **Language**: $A_{\text{CFG}} = \{\langle G, w\rangle | G$ is a CFG that generates string $w\}$
- **Theorem 4.7**: A_{CFG} is a decidable language.
- **Machine**: Let $S_{\text{bad}} =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:
  1. Generate all derivations of G.
  2. If any of these derivations match w, *accept*; otherwise, *reject*."

A_{CFG}

- **Language**: $A_{\text{CFG}} = \{\langle G, w\rangle | G \text{ is a CFG that generates string } w\}$
- **Theorem 4.7**: A_{CFG} is a decidable language.
- **Machine**: Let $S_{\text{bad}} =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:
    1. Generate all derivations of G.
    2. If any of these derivations match w, *accept*; otherwise, *reject*."
- **Proof Flaw?**: What is the flaw in this machine?

A_{CFG}

- **Language**: $A_{\mathsf{CFG}} = \{\langle G, w\rangle | G$ is a CFG that generates string $w\}$
- **Theorem 4.7**: A_{CFG} is a decidable language.

A_{CFG}

- **Language**: $A_{\text{CFG}} = \{\langle G, w\rangle | G \text{ is a CFG that generates string } w\}$
- **Theorem 4.7**: A_{CFG} is a decidable language.
- **Machine**: Let $S =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:
  1. Convert G to an equivalent grammar in Chomsky normal form.
  2. Generate all derivations with $2n - 1$ steps, where $n = |w|$. If $n = 0$, generate length 1 derivations.
  3. If any of these derivations match w, *accept*; otherwise, *reject*."

A_{CFG}

- **Language**: $A_{\text{CFG}} = \{\langle G, w\rangle | G \text{ is a CFG that generates string } w\}$
- **Theorem 4.7**: A_{CFG} is a decidable language.
- **Machine**: Let $S =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:
  1. Convert G to an equivalent grammar in Chomsky normal form.
  2. Generate all derivations with $2n - 1$ steps, where $n = |w|$. If $n = 0$, generate length 1 derivations.
  3. If any of these derivations match w, *accept*; otherwise, *reject*."
- **Proof Sketch**: The Chomsky normal form conversion is finite. The $2n - 1$ proof is in chapter 2 problems. This makes the number of strings generated finite.

E_{CFG}

- **Language**: $E_{\text{CFG}} = \{\langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset\}$
- **Theorem 4.8**: E_{CFG} is a decidable language.

E_{CFG}

- **Language**: $E_{\text{CFG}} = \{\langle G\rangle | G$ is a CFG and $L(G) = \emptyset\}$
- **Theorem 4.8**: E_{CFG} is a decidable language.
- **Machine**: Let $R =$ "On input $\langle G\rangle$, where G is a CFG:
  1. Mark all terminal symbols in G.
  2. Repeat until no new variables get marked.
  3. Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, ..., U_k$ is marked.
  4. If the start variable is not marked, *accept*; otherwise, *reject*."

E_{CFG}

- **Language**: $E_{\text{CFG}} = \{\langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset\}$
- **Theorem 4.8**: E_{CFG} is a decidable language.
- **Machine**: Let $R =$ "On input $\langle G \rangle$, where G is a CFG:
  1. Mark all terminal symbols in G.
  2. Repeat until no new variables get marked.
  3. Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots, U_k$ is marked.
  4. If the start variable is not marked, *accept*; otherwise, *reject*."
- **Proof Sketch**: The number of symbols is finite. The loop will terminate in a finite amount of time. R is a decider.

EQ_{CFG}

- **Language**:
  $EQ_{\text{CFG}} = \{\langle G, H\rangle | G$ and H are CFGs and $L(G) = L(H)\}$
- **Discussion**:

EQ_{CFG}

- **Language**:
  $EQ_{\text{CFG}} = \{\langle G, H\rangle | G$ and H are CFGs and $L(G) = L(H)\}$
- **Discussion**: Why not use the strategy from EQ_{DFA},
  $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$?

EQ_{CFG}

- **Language**: $EQ_{\text{CFG}} = \{\langle G, H\rangle | G$ and H are CFGs and $L(G) = L(H)\}$
- **Discussion**: Why not use the strategy from EQ_{DFA}, $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$?
- The class of context-free languages is not closed under complementation nor under intersection.

EQ_{CFG}

- **Language**: $EQ_{\text{CFG}} = \{\langle G, H\rangle | G$ and H are CFGs and $L(G) = L(H)\}$
- **Discussion**: Why not use the strategy from EQ_{DFA}, $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$?
- The class of context-free languages is not closed under complementation nor under intersection.
- EQ_{CFG} is not decidable. Proof to come later.

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.
- **Machine**: Let $M_{G_{\text{bad}}} =$ "On input $\langle G \rangle$ and string w:
  1. Convert G into PDA B.
  2. Simulate B on input w.
  3. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.
- **Machine**: Let $M_{G_{\text{bad}}} =$ "On input $\langle G \rangle$ and string w:
  1. Convert G into PDA B.
  2. Simulate B on input w.
  3. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."
- **Proof Flaw?**: What is the flaw in this machine?

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.
- **Machine**: Let $M_{G_{\text{bad}}} =$ "On input $\langle G \rangle$ and string w:
  1. Convert G into PDA B.
  2. Simulate B on input w.
  3. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."
- **Proof Flaw?**: What is the flaw in this machine?
- Hint: Deciders must halt.

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.
- **Machine**: Let $M_{G_{\text{bad}}} =$ "On input $\langle G \rangle$ and string w:
  1. Convert G into PDA B.
  2. Simulate B on input w.
  3. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."
- **Proof Flaw?**: What is the flaw in this machine?
- Hint: Deciders must halt.
- Hint: Why can we not guarantee that the simulation of B will halt?

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.
- **Machine**: Let $M_G =$ "On input string w:
  1. Run S from Theorem 4.7 on input $\langle G, w \rangle$.
  2. If S accepts, *accept*; otherwise, *reject*."

Context-free languages are decidable

- **Theorem 4.9**: Every context-free language is decidable.
- **Machine**: Let $M_G =$ "On input string w:
  1. Run S from Theorem 4.7 on input $\langle G, w\rangle$.
  2. If S accepts, *accept*; otherwise, *reject*."
- **Proof Sketch**: The only work is to encode the finite grammar G. M_G is a decider.

Undecidable Languages

Reading: Sipser §4.2.

A_{TM} is recognizable

- **Language**:
  $A_{\mathsf{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$

A_{TM} is recognizable

- **Language**:
  $A_{\text{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$
- **Theorem**: A_{TM} is recognizable.

A_{TM} is recognizable

- **Language**: $A_{\text{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$
- **Theorem**: A_{TM} is recognizable.
- **Machine**: Let $U =$ "On input $\langle M, w\rangle$, where M is a TM and w is a string:
  1. Simulate M on input w.
  2. If M enters its accept state, *accept*; if M enters its reject state, *reject*."

A_{TM} is recognizable

- **Language**:
  $A_{\text{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$
- **Theorem**: A_{TM} is recognizable.
- **Machine**: Let $U =$ "On input $\langle M, w\rangle$, where M is a TM and w is a string:
  1. Simulate M on input w.
  2. If M enters its accept state, *accept*; if M enters its reject state, *reject*."
- **Proof Sketch**: What are the possible outcomes?

A_{TM} is recognizable

- **Language**:
  $A_{\mathsf{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$
- **Theorem**: A_{TM} is recognizable.
- **Machine**: Let $U =$ "On input $\langle M, w\rangle$, where M is a TM and w is a string:
  1. Simulate M on input w.
  2. If M enters its accept state, *accept*; if M enters its reject state, *reject*."
- **Proof Sketch**: What are the possible outcomes? If M accepts w, U will accept. If M does not accept w, M will either reject or loop, and U will either reject or loop. These are the conditions for a recognizer TM. U recognizes A_{TM}.

A_{TM} is undecidable

- **Language**:
  $A_{\mathsf{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$

A_{TM} is undecidable

- **Language**:
  $A_{\mathsf{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$
- **Theorem 4.11**: A_{TM} is undecidable.

A_{TM} is undecidable

- **Language**:
  $A_{\mathsf{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$
- **Theorem 4.11**: A_{TM} is undecidable.
- **How can we prove undecidability?** Any ideas from the way we prove not context-free, or not regular?

A_{TM} is undecidable

- **Language**:
  $A_{\mathsf{TM}} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$
- **Theorem 4.11**: A_{TM} is undecidable.
- **How can we prove undecidability?** Any ideas from the way we prove not context-free, or not regular?
  Assume it is decidable, then show that assumption leads to a contradiction.

A_{TM} is undecidable

- Assume A_{TM} is decidable $\rightarrow$ Let H be a decider for A_{TM}.

$$H = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

A_{TM} is undecidable

- Assume A_{TM} is decidable $\rightarrow$ Let H be a decider for A_{TM}.

$$H = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

A_{TM} is undecidable

- Assume A_{TM} is decidable $\rightarrow$ Let H be a decider for A_{TM}.

$$H = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \\ & \end{cases}$$

A_{TM} is undecidable

- Assume A_{TM} is decidable $\rightarrow$ Let H be a decider for A_{TM}.

$$H = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ & \end{cases}$$

A_{TM} is undecidable

- Assume A_{TM} is decidable $\rightarrow$ Let H be a decider for A_{TM}.

$$H = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \end{cases}$$

A_{TM} is undecidable

- Assume A_{TM} is decidable $\to$ Let H be a decider for A_{TM}.

$$H = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

A_{TM} is undecidable

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

A_{TM} is undecidable

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

- **What is the result of $D(\langle D \rangle)$?**

A_{TM} is undecidable

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

- **What is the result of $D(\langle D \rangle)$?**

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \\ & \end{cases}$$

A_{TM} is undecidable

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

- **What is the result of $D(\langle D \rangle)$?**

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ & \end{cases}$$

A_{TM} is undecidable

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

- **What is the result of $D(\langle D \rangle)$?**

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \end{cases}$$

A_{TM} is undecidable

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

- **What is the result of $D(\langle D \rangle)$?**

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

A_{TM} is undecidable

- **Machine**: Let $D =$ "On input $\langle M \rangle$, where M is a TM:
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H rejects, *accept*; if H accepts, *reject*."

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

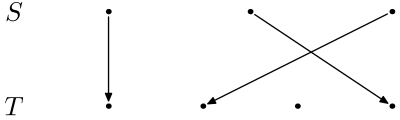
- **What is the result of $D(\langle D \rangle)$?**

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

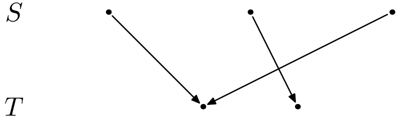
- This contradiction proves our assumption of "A_{TM} is decidable and H exists as a decider for A_{TM}" is false. A_{TM} is not decidable.

Correspondence

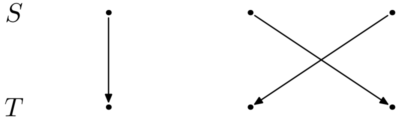
Special varieties of functions



1-1:
$s_1 \neq s_2 \Rightarrow$
$f(s_1) \neq f(s_2)$



Onto:
For every $t \in T$
there is an $s \in S$
such that $f(s) = t$



Bijection:
1-1 and onto
“1-1 Correspondence”

Formal definition of **cardinality**: S has (finite) cardinality $n \in \mathcal{N}$ iff there is a bijection $f : \{1, \dots, n\} \to S$.

Size of a set

co-Turing-recognizable

- **Definition**: Language A is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

co-Turing-recognizable

- **Definition**: Language A is co-Turing-recognizable if it is the complement of a Turing-recognizable language.
- **Alt. Definition**: Language A is co-Turing-recognizable if $\overline{A}$ is Turing-recognizable.

co-Turing-recognizable

- **Definition**: Language A is co-Turing-recognizable if it is the complement of a Turing-recognizable language.
- **Alt. Definition**: Language A is co-Turing-recognizable if $\overline{A}$ is Turing-recognizable.
- **Alt. Definition**: Language $\overline{A}$ is co-Turing-recognizable if A is Turing-recognizable.

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:
  - **Part 1:** If language A is decidable, then it is Turing-recognizable and co-Turing-recognizable.

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:
  - **Part 1:** If language A is decidable, then it is Turing-recognizable and co-Turing-recognizable. All decidable languages are Turing-recognizable, due to the accept requirement.

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:
  - **Part 1:** If language A is decidable, then it is Turing-recognizable and co-Turing-recognizable. All decidable languages are Turing-recognizable, due to the accept requirement. The complement of a decidable language is also decidable, by swapping accept and reject results.

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:
  - **Part 1:** If language A is decidable, then it is Turing-recognizable and co-Turing-recognizable. All decidable languages are Turing-recognizable, due to the accept requirement. The complement of a decidable language is also decidable, by swapping accept and reject results.
  - **Part 2:** If language A is Turing-recognizable and co-Turing-recognizable, then it is decidable.

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:
  - **Part 1:** If language A is decidable, then it is Turing-recognizable and co-Turing-recognizable. All decidable languages are Turing-recognizable, due to the accept requirement. The complement of a decidable language is also decidable, by swapping accept and reject results.
  - **Part 2:** If language A is Turing-recognizable and co-Turing-recognizable, then it is decidable. If A is Turing-recognizable by M_1, and $\overline{A}$ is Turing-recognizable by M_2, then we can make a decider for A.

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:
  - **Part 1:** If language A is decidable, then it is Turing-recognizable and co-Turing-recognizable. All decidable languages are Turing-recognizable, due to the accept requirement. The complement of a decidable language is also decidable, by swapping accept and reject results.
  - **Part 2:** If language A is Turing-recognizable and co-Turing-recognizable, then it is decidable. If A is Turing-recognizable by M_1, and $\overline{A}$ is Turing-recognizable by M_2, then we can make a decider for A.
    Machine: Let $M =$ "On input w:
    1. Run both M_1 and M_2 on input w in parallel.
    2. If M_1 accepts, *accept*; if M_2 accepts, *reject*."

Decidable, Turing-recognizable, and co-Turing-recognizable

- **Theorem 4.22**: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
- **Proof**:
  - **Part 1:** If language A is decidable, then it is Turing-recognizable and co-Turing-recognizable. All decidable languages are Turing-recognizable, due to the accept requirement. The complement of a decidable language is also decidable, by swapping accept and reject results.
  - **Part 2:** If language A is Turing-recognizable and co-Turing-recognizable, then it is decidable. If A is Turing-recognizable by M_1, and $\overline{A}$ is Turing-recognizable by M_2, then we can make a decider for A.
    Machine: Let $M =$ "On input w:
    1. Run both M_1 and M_2 on input w in parallel.
    2. If M_1 accepts, *accept*; if M_2 accepts, *reject*."

An unrecognizable language

- **Theorem 4.23**: $\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable.

An unrecognizable language

- **Theorem 4.23**: $\overline{A_{\text{TM}}}$ is not Turing-recognizable.
- **Proof**:

An unrecognizable language

- **Theorem 4.23**: $\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable.
- **Proof**: A_{TM} is Turing-recognizable (by unnamed Theorem). By Theorem 4.22, if $\overline{A_{\mathsf{TM}}}$ were Turing-recognizable, then A_{TM} would be decidable. A_{TM} is undecidable (by Theorem 4.11).

Language Nesting