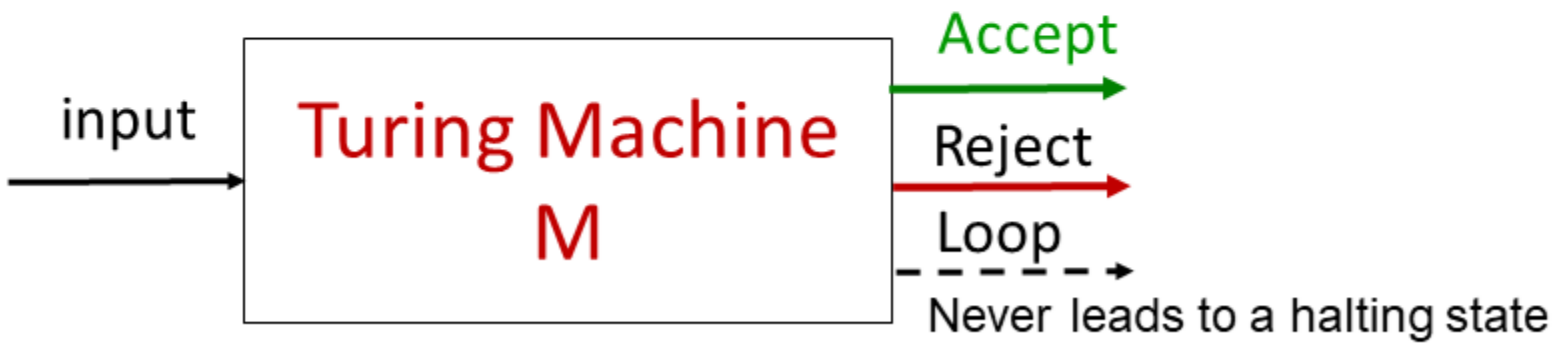
Part Two: Chapter 4

Decidability

Recall: Turing-recognizable

- The collection of strings that TM M accepts is the language of M, or the language recognized by M, denoted $L(M)$.

- Call a language Turing-recognizable if some Turing machine recognizes it.

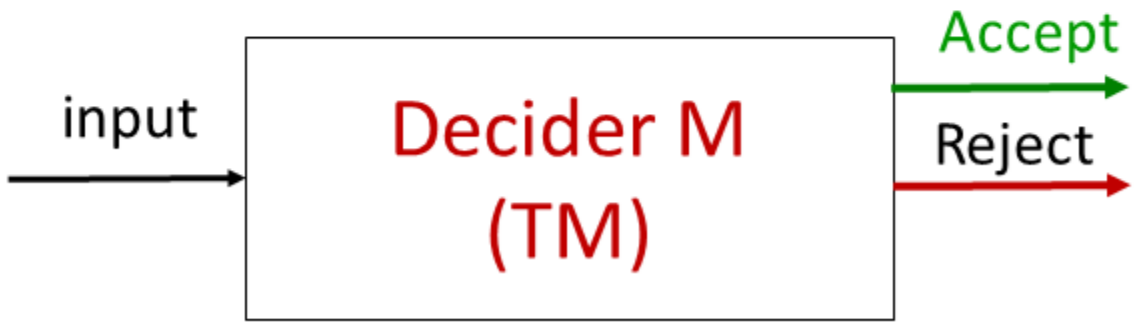


Recall: Decider

- we prefer Turing machines that halt on all inputs;
  - such machines never loop.

- These machines are called deciders because they always make a decision to accept or reject.

- A decider that recognizes some language also is said
  - to decide that language.



DECIDABILITY OVERVIEW

- Every question about regular languages is decidable.

DECIDABILITY OVERVIEW

- Every question about regular languages is decidable.
- Some questions about Context-free languages are decidable,
  - But some are not.

Chapter 4

DECIDABILITY OVERVIEW

- Every question about regular languages is decidable.
- Some questions about Context-free languages are decidable,
  - But some are not.
- The "HALTING PROBLEM" is not decidable.

Chapter 4

DECIDABILITY OVERVIEW

- Every question about regular languages is decidable.
- Some questions about Context-free languages are decidable,
  - But some are not.
- The "HALTING PROBLEM" is not decidable.
- Some languages are not Turing recognizable.

DECIDABILITY OVERVIEW

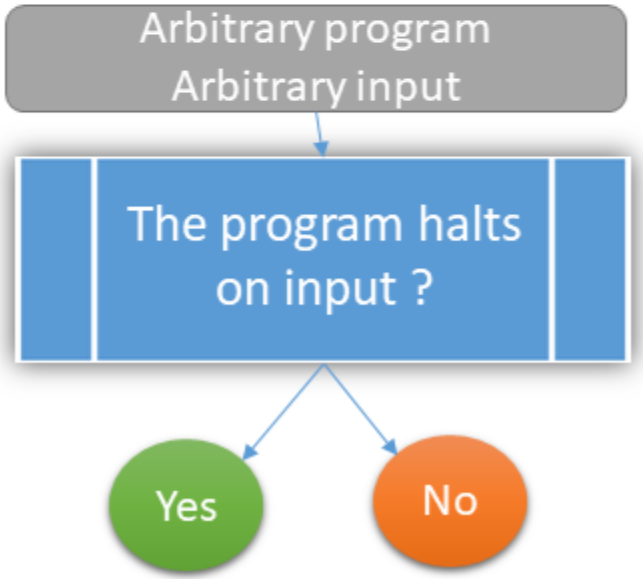
- Every question about regular languages is decidable.
- Some questions about Context-free languages are decidable,
  - But some are not.
- The "HALTING PROBLEM" is not decidable.
- Some languages are not Turing recognizable.
- Many **questions about Turing Machines** are not decidable.

THE HALTING PROBLEM

- Given a program ….
  - Will it HALT?

- Given a Turing machine,
  - Will it HALT when run on some particular input string.

In computability theory, the **halting problem** is the **problem** of determining, from a description of an arbitrary computer program and an input, whether the program will finish running (i.e., **halt**) or continue to run forever.



THE HALTING PROBLEM

- Given a program ….
  - Will it HALT?

- Given a Turing machine,
  - Will it HALT when run on some particular input string.

- For many programs, we can prove "It always HALT."
- For many programs, we can prove "It may sometimes LOOP."

- But for programs in general, We cannot always know.
  - The question is undecidable.

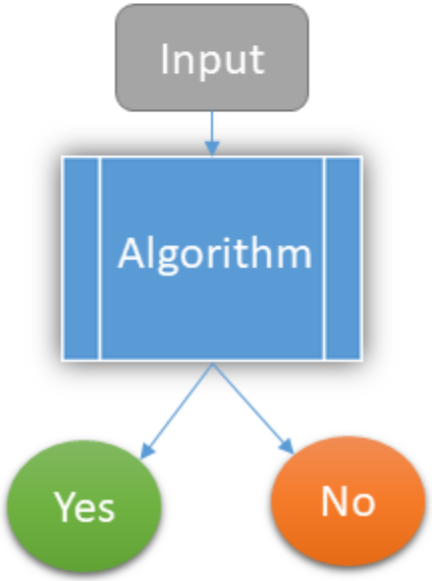
DECISION PROBLEMS

- We may be interested in following questions:
  - Is a given natural number prime?
  - Is an undirected graph connected?
  - Does the computation of TM halt **before 10th transition**?

DECISION PROBLEMS

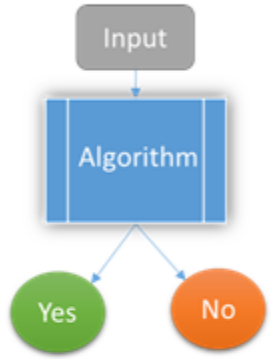
- We may be interested in following questions:
  - Is a given natural number prime?
  - Is an undirected graph connected?
  - Does the computation of TM halt before 10th transition?

- Each of these general question describe a decision problem.
- Decision problem = any problem whose **answer** is one bit: "yes" or "no".



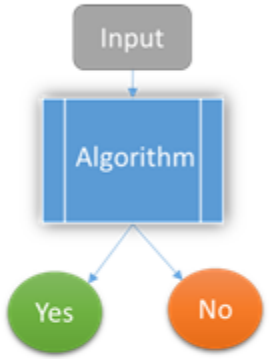
DECISION PROBLEMS

- A decision problem P is a set of related questions p_i, each of which has Yes/no answer. For example:
  - p_0: Is 0 a perfect square?
  - p_1: Is 1 a perfect square?
  - p_2: Is 2 a perfect square?



DECISION PROBLEMS

- A decision problem P is a set of related questions p_i , each of which has Yes/no answer. For example:
  - p_0: Is 0 a perfect square?
  - p_1: Is 1 a perfect square?
  - p_2: Is 2 a perfect square?



- Each of the p_i is an instance of the problem P.
- The solution of a decision problem P is an algorithm that determines the answer of every question $p_i \in P$.

- A **decision problem** is decidable, if it has a solution.

THE ACCEPTANCE PROBLEM FOR FINITE AUTOMATA

- We begin with certain computational problems concerning finite automata.

- We give algorithms for testing
  - whether a finite automaton accepts a string,
  - whether the language of a finite automaton is empty,
  - and whether two finite automata are equivalent.

- For example, the acceptance problem for DFAs of testing whether a particular deterministic finite automaton accepts a given string can be expressed as a language, ADFA.

$$A_{\text{DFA}} = \{\langle B, w\rangle | \ B \text{ is a DFA that accepts input string } w\}.$$

THE ACCEPTANCE PROBLEM FOR FINITE AUTOMATA

- We begin with certain computational problems concerning finite automata.

- We give algorithms for testing
  - whether a finite automaton accepts a string,
  - whether the language of a finite automaton is empty,
  - and whether two finite automata are equivalent.

THE ACCEPTANCE PROBLEM FOR DFAs

- The acceptance problem for DFAs of testing whether **a particular deterministic finite automaton** accepts a given string can be expressed as a language, A_{DFA}.

$$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}.$$

THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{DFA} is a decidable language.

$$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}.$$

PROOF IDEA We simply need to present a TM M that decides A_{DFA}.

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*."

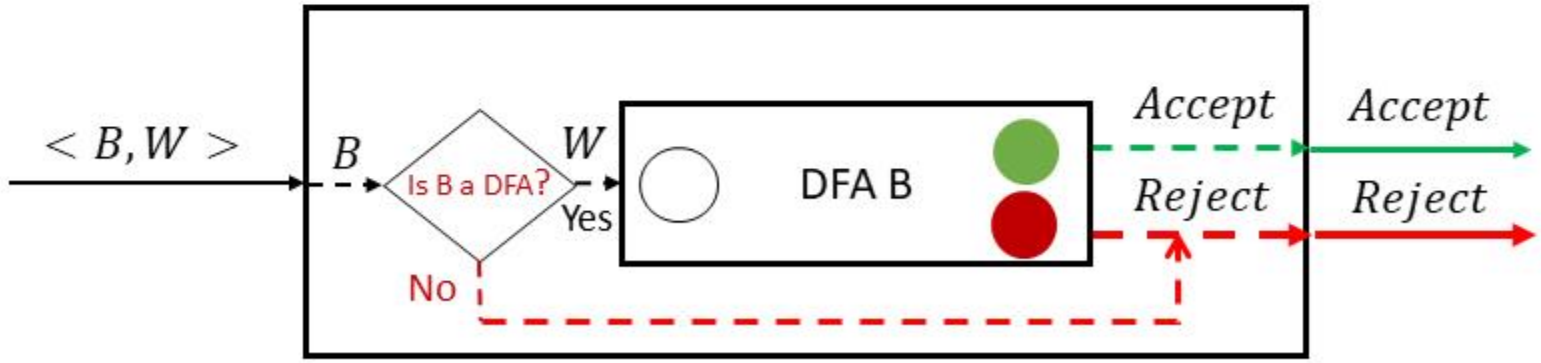
THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{DFA} is a decidable language.

PROOF IDEA We simply need to present a TM M that decides A_{DFA}.

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*."



TM M

THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{NFA} is a decidable language.

$$A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}.$$

THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{NFA} is a decidable language.

$$A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}.$$

N = "On input $\langle B, w\rangle$, where B is an NFA and w is a string:

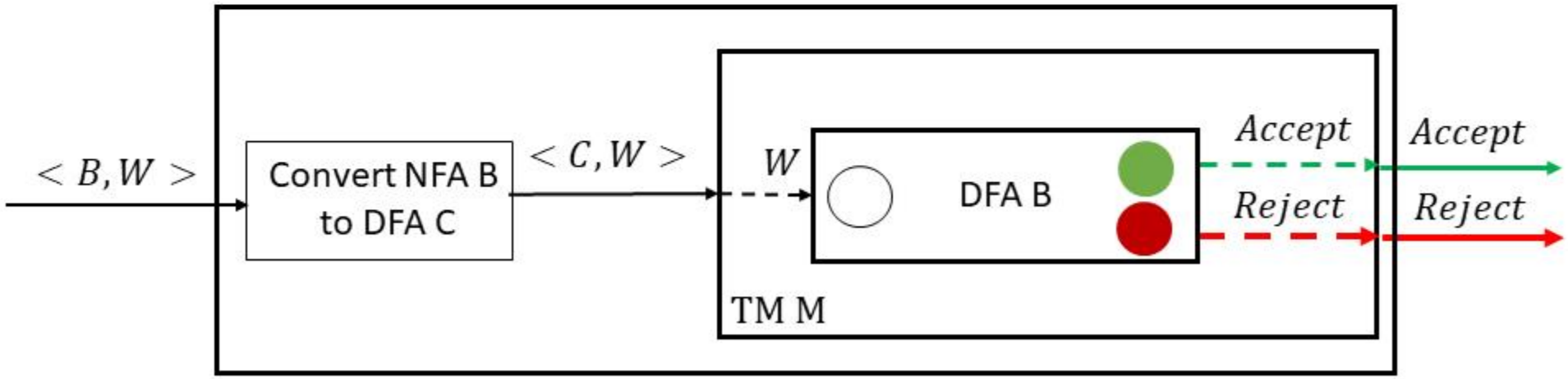
1. Convert NFA B to an equivalent DFA C, using the procedure for this conversion given in Theorem 1.39.
2. Run TM M from Theorem 4.1 on input $\langle C, w\rangle$.
3. If M accepts, *accept*; otherwise, *reject*."

THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{NFA} is a decidable language.

N = "On input $\langle B, w \rangle$, where B is an NFA and w is a string:

1. Convert NFA B to an equivalent DFA C, using the procedure for this conversion given in Theorem 1.39.
2. Run TM M from Theorem 4.1 on input $\langle C, w \rangle$.
3. If M accepts, *accept*; otherwise, *reject*."



THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{REX} is a decidable language.

$$A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}.$$

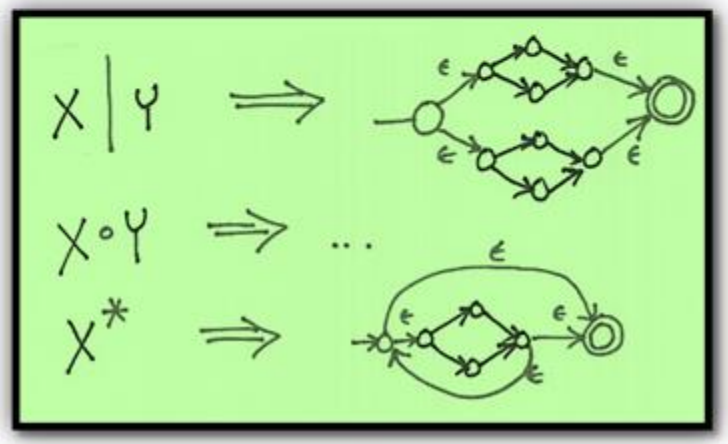
THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{REX} is a decidable language.

$$A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}.$$

P = "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A by using the procedure for this conversion given in Theorem 1.54.
2. Run TM N on input $\langle A, w\rangle$.
3. If N accepts, *accept*; if N rejects, *reject*."

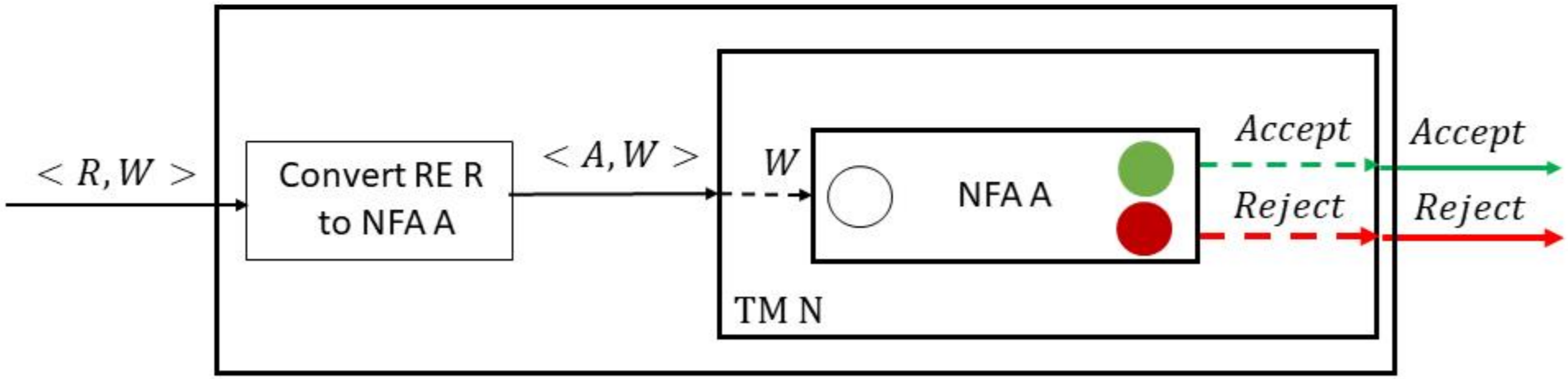


THE ACCEPTANCE PROBLEM FOR TMs

Theorem : A_{REX} is a decidable language.

P = "On input $\langle R, w \rangle$, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A by using the procedure for this conversion given in Theorem 1.54.
2. Run TM N on input $\langle A, w \rangle$.
3. If N accepts, *accept*; if N rejects, *reject*."



EMPTINESS PROBLEM

- In the preceding three theorems we had to determine
  - whether a finite automaton accepts a particular string.
  - (acceptance problem)

- Now we turn to a different kind of problem concerning finite automata:
  - emptiness testing for the **language of a finite automaton**.
  - (emptiness problem)
-

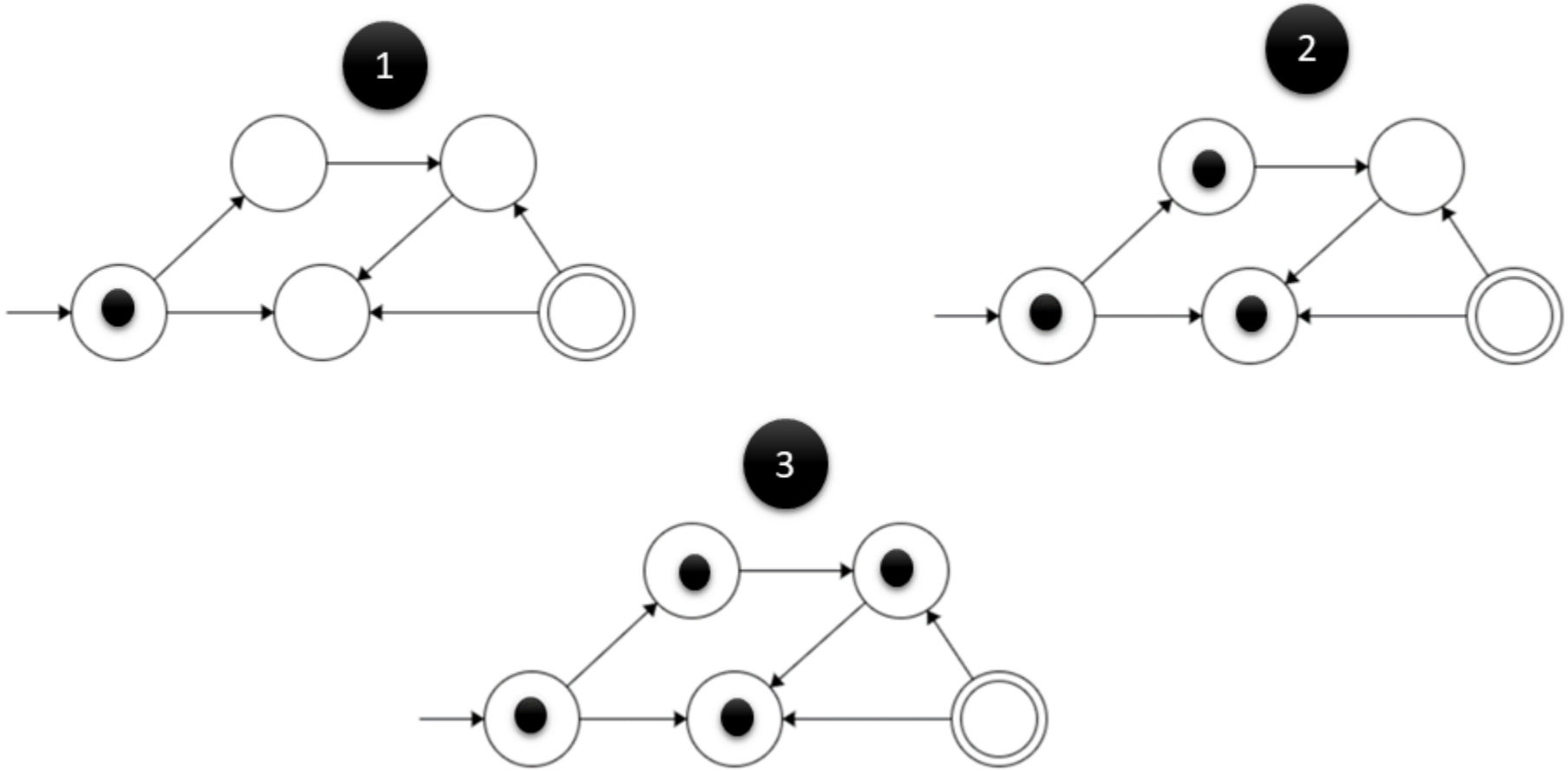
DECIDABILITY

Theorem : E_{DFA} is a decidable language.

$$E_{\text{DFA}} = \{\langle A \rangle | \ A \text{ is a DFA and } L(A) = \emptyset\}.$$

DECIDABILITY

Marking Algorithm:



- Mark any state that has a transition coming into it from any state that is already marked.

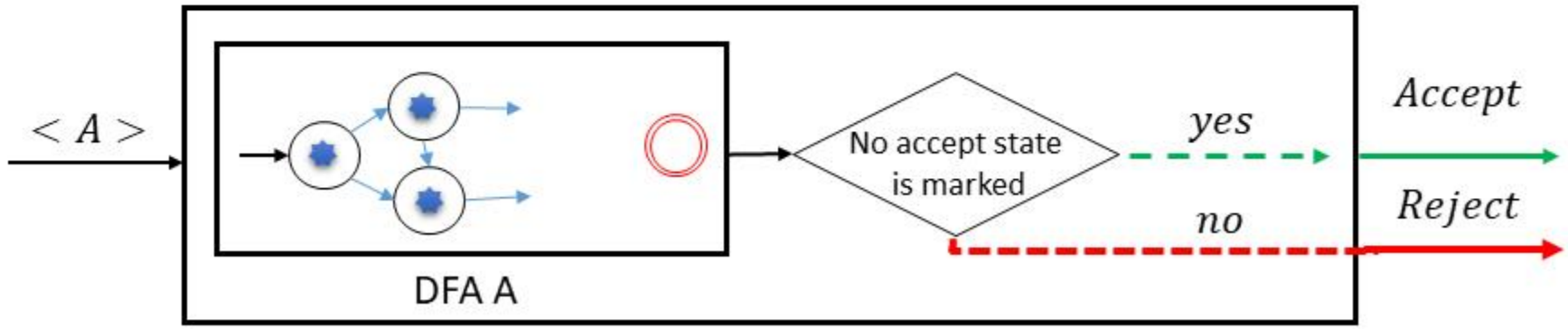
DECIDABILITY

Theorem : E_{DFA} is a decidable language.

$$E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}.$$

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*."



TM T

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

$$EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}.$$

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof:

- We know that E_{DFA} is a decidable language.

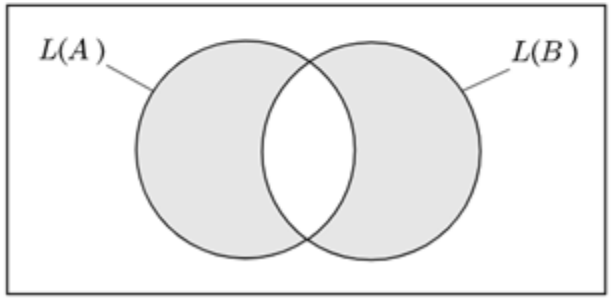
DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof:

- We know that E_{DFA} is a decidable language.

- We construct a new DFA C from A and B,
  - where C accepts only those strings that are accepted by either A or B but not by both.



$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

symmetric difference of L(A) and L(B)

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof:

- We know that E_{DFA} is a decidable language.
- We construct a new DFA C from A and B,
  - where C accepts only those strings that are accepted by either A or B but not by both.
- Thus, if A and B recognize the same language,
  - C will accept nothing.

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof(cont.):

- $L(C) = \emptyset$ iff $L(A) = L(B)$.

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof(cont.):

- $L(C) = \emptyset$ iff $L(A) = L(B)$.
- We can construct C from A and B.
  - (the class of regular languages are closed under complement, union, and intersection)

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof(cont.):

- $L(C) = \emptyset$ iff $L(A) = L(B)$.
- We can construct C from A and B.
  - (the class of regular languages are closed under complement, union, and intersection)

- These constructions are algorithms
  - that can be carried out by Turing machines.

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof(cont.):

- $L(C) = \emptyset$ iff $L(A) = L(B)$.
- We can construct C from A and B.
  - (the class of regular languages are closed under complement, union, and intersection)
- These constructions are algorithms that can be carried out by Turing machines.

- Once we have constructed C,
  - we can test whether L(C) is empty.
    - If it is empty, L(A) and L(B) must be equal.

DECIDABILITY

Theorem : EQ_{DFA} is a decidable language.

Proof(cont.):

- $L(C) = \emptyset$ iff $L(A) = L(B)$.

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

F = "On input $\langle A, B \rangle$, where A and B are DFAs:

1. Construct DFA C as described.
2. Run TM T from Theorem 4.4 on input $\langle C \rangle$.
3. If T accepts, *accept*. If T rejects, *reject*."

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*."

THE ACCEPTANCE PROBLEM FOR TMs :CFG

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

THE ACCEPTANCE PROBLEM FOR TMs :CFG

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof:

- We want to determine whether G **generates** w.
- Approach 1: use G to **go through all derivations** to determine whether any is a derivation of w.

THE ACCEPTANCE PROBLEM FOR TMs :CFG

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof:

- We want to determine whether G **generates** w.

- Approach 1: use G to **go through all derivations** to determine whether any is a derivation of w.
  - It is not a good idea!!

THE ACCEPTANCE PROBLEM FOR TMs :CFG

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof:

- We want to determine whether G **generates** w.

- Approach 1: use G to **go through all derivations** to determine whether any is a derivation of w.
  - It is not a good idea!!
    - If G does **not generate** w, this algorithm would never halt.
    - The **Turing machine** will be a **recognizer**, but not a decider.

DECIDABILITY

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof (cont.):

- We want to determine whether G **generates** w.

- Approach 2: We need to ensure that the **algorithm tries** only finitely many derivations.

DECIDABILITY

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof (cont.):

- We want to determine whether G **generates** w.

- Approach 2: We need to ensure that the **algorithm tries** only finitely many derivations.
  - **In Problem 2.26 (Textbook) :**
    - if G is in Chomsky normal form, any derivation of w has $2n-1$ steps, where n is the length of w.

DECIDABILITY

Proof (cont.):

- Example: Consider the following Chomsky normal form (For simplicity we didn't follow the first rule in this example.)

$$S \rightarrow SS$$
$$S \rightarrow a$$

- At each step, the length grows by exactly 1
  - $S \Rightarrow SS \Rightarrow SSS \Rightarrow SSSS \Rightarrow SSSSS$ (N-1 steps for length N)

- Plus 1 additional step for each terminal symbol
- $\Rightarrow aSSSS \Rightarrow aaSSS \Rightarrow aaaSS \Rightarrow aaaaS \Rightarrow aaaaa$ (N steps for length N)

- Every derivation has 2N-1 steps.

DECIDABILITY

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof (cont.):

- We want to determine whether G **generates** w.

- Approach 2: We need to ensure that the algorithm tries only finitely many derivations.
  - See Problem 2.26 (Textbook) that says
    - if G is in Chomsky normal form, any derivation of w has 2n−1 steps, where n is the length of w.

      - In that case, checking only derivations with 2n−1 steps to determine whether G generates w would be sufficient.

DECIDABILITY

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof (cont.):

- The TM S for A_{CFG} follows.

$S =$ "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. List all derivations with $2n-1$ steps, where n is the length of w; except if $n = 0$, then instead list all derivations with one step.
3. If any of these derivations generate w, *accept*; if not, *reject*."

DECIDABILITY

Theorem : A_{CFG} is a decidable language.

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}.$$

Proof (cont.):

- The TM S for A_{CFG} follows.

> 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. List all derivations with $2n - 1$ steps, where n is the length of w; except if $n = 0$, then instead list all derivations with one step.
> 3. If any of these derivations generate w, *accept*; if not, *reject*."

- The **algorithm** in TM S is very inefficient
- But the **efficiency** is not our concern here.

DECIDABILITY IN PDAs

- Recall that we have given procedures for converting back and forth between CFGs and PDAs.

- Hence everything we say about the decidability of problems concerning CFGs applies equally well to PDAs.

DECIDABILITY in CFGs

Theorem : E_{CFG} is a decidable language.

$$E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}.$$

DECIDABILITY in CFGs

Theorem : E_{CFG} is a decidable language.

$$E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}.$$

Proof :

- In order to determine whether the **language of a grammar is empty**,
  - we need to test whether the start variable can generate a string of terminals.

$$S \overset{*}{\Rightarrow} (string\ of\ termnals)$$

  - **Approach:** Marking algorithm

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, C and D Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, and C Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, and C Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, and C Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, and C Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, and C Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, and C Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

DECIDABILITY in CFGs

Proof (cont.):

- Marking algorithm Example:
- Consider the following grammar. Which nonterminal can generate a string of terminals? (Nonterminals: A,B, and C Terminals: a, b, c and d)

- S → ABCD
- A→ BCA
- A→ abc
- B→ CA
- B→AB
- B→BBd
- C→ CB
- C→ dd
- D→DD
- D→BD
- D→DC

Since D doesn't generate any string, then **S** cannot generate any string.

DECIDABILITY

Theorem : E_{CFG} is a decidable language.

$$E_{\text{CFG}} = \{\langle G \rangle | \; G \text{ is a CFG and } L(G) = \emptyset\}.$$

DECIDABILITY

Theorem : E_{CFG} is a decidable language.

$$E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}.$$

Proof (cont.) :

$R =$ "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 $U_1, \ldots, U_k$ has already been marked.
4. If the start variable is not marked, *accept*; otherwise, *reject*."

DECIDABILITY

Is EQ_{CFG} a decidable language ?

$$EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}.$$

DECIDABILITY

Is EQ_{CFG} a decidable language ?

$$EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}.$$

- We have already seen an algorithm that decides the language EQ_{DFA} for finite automata.
- We used the **decision procedure** for E_{DFA} to prove that EQ_{DFA} is decidable.

DECIDABILITY

Is EQ_{CFG} a decidable language ?

$$EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}.$$

- We have already seen an algorithm that decides the language EQ_{DFA} for finite automata.
- We used the **decision procedure** for E_{DFA} to prove that EQ_{DFA} is decidable.

- Because E_{CFG} also is decidable, can we use a similar strategy to prove that EQ_{CFG} is decidable?

DECIDABILITY

Is EQ_{CFG} a decidable language ?

$$EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}.$$

- We have already seen an algorithm that decides the language EQ_{DFA} for finite automata.
- We used the **decision procedure** for E_{DFA} to prove that EQ_{DFA} is decidable.
- Because E_{CFG} also is decidable, can we use a similar strategy to prove that EQ_{CFG} is decidable?

- NO. something is wrong with this idea!

DECIDABILITY

Is EQ_{CFG} a decidable language ?

$$EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}.$$

- We have already seen an algorithm that decides the language EQ_{DFA} for finite automata.
- We used the **decision procedure** for E_{DFA} to prove that EQ_{DFA} is decidable.
- Because E_{CFG} also is decidable, can we use a similar strategy to prove that EQ_{CFG} is decidable?

- NO. something is wrong with this idea!
- The **class of context-free languages** is not closed under complementation or intersection.
- In fact, EQ_{CFG} is not decidable. (the prove is presented in Chapter 5)

DECIDABILITY

Theorem : Every context-free language is a decidable language.

DECIDABILITY

Theorem : Every context-free language is a decidable language.

Proof :

- Approach1 :
- Let A be a CFL.
- **Convert** a PDA for A directly into a TM.
  - simulating a stack with the TM's more tape

DECIDABILITY

Theorem : Every context-free language is a decidable language.

Proof :

- Approach1 :
- Let A be a CFL.
- **Convert** a PDA for A directly into a TM.
  - simulating a stack with the TM's more tape
  - The PDA for A may be **nondeterministic**
    - convert it into a **nondeterministic TM**
    - any nondeterministic TM can be converted into an **equivalent deterministic TM**.
    - But, **some branches** of the PDA's computation may go on forever
    - **The simulating TM** then would also have some non-halting branches in its computation, and so the TM would not be a decider.

DECIDABILITY

Theorem : Every context-free language is a decidable language.

Proof (cont.) :

- Approach2 :
- Let G be a CFG for A and design a TM M_G that decides A.
- We build a copy of G into M_G. It works as follows.

DECIDABILITY

Theorem : Every context-free language is a decidable language.

Proof (cont.) :

- Approach2 :
- Let G be a CFG for A and design a TM M_G that decides A.
- We build a copy of G into M_G. It works as follows.

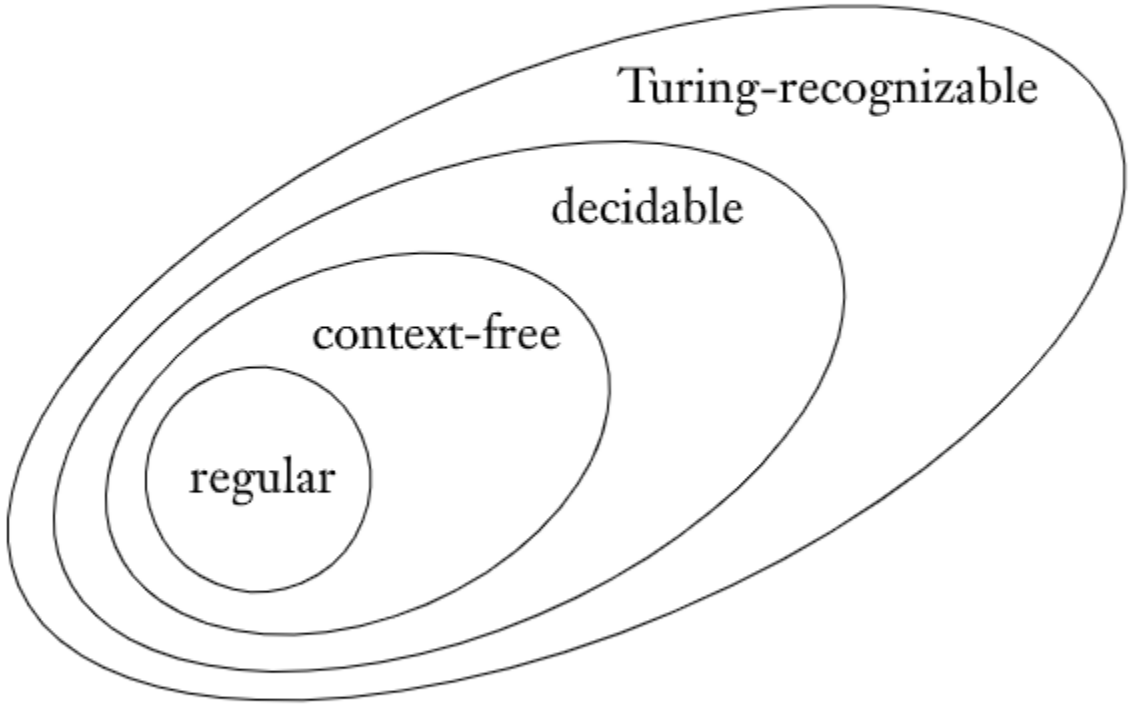
M_G = "On input w:

1. Run TM S on input $\langle G, w \rangle$.
2. If this machine accepts, *accept*; if it rejects, *reject*."

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. List all derivations with $2n-1$ steps, where n is the length of w; except if $n = 0$, then instead list all derivations with one step.
3. If any of these derivations generate w, *accept*; if not, *reject*."

THE RELATIONSHIP AMONG THE FOUR MAIN CLASSES OF LANGUAGES



Chapter 4

4.2 Undecidability

Chapter 4

UNDECIDABILITY

- What sorts of problems are unsolvable by computer?
- The unsolvable problems are only in the minds of theoreticians?

UNDECIDABILITY

- What sorts of problems are unsolvable by computer?
- The unsolvable problems are only in the minds of theoreticians?

- Answer: There are some ordinary problems that they are computationally unsolvable.

UNDECIDABILITY

- **In this section we will see:**
  - There is a specific problem that is algorithmically unsolvable.

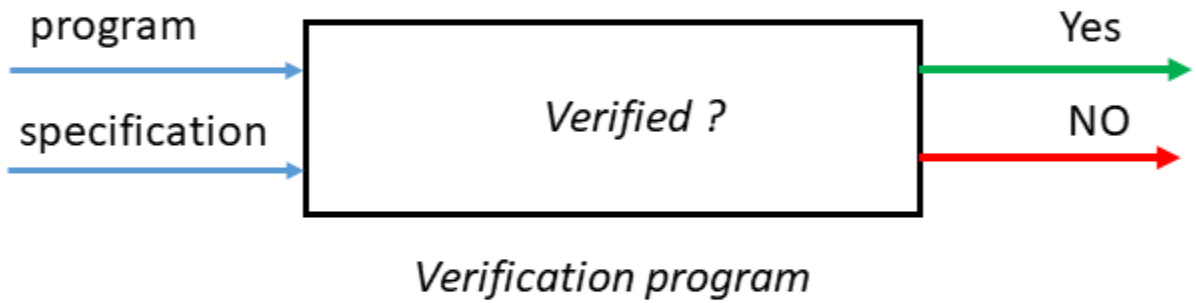
- **The theorem presented here:**
  - demonstrates that computers are limited in a fundamental way.

SOFTWARE VERIFICATION

- One example of unsolvable problem:
  - **Given :**
    - a computer program and
    - a precise specification of what that program is supposed to do.
  - **Verify that**
    - the **program** performs as specified.

SOFTWARE VERIFICATION

- One example of unsolvable problem: (Cont.)



The general problem of software verification is not solvable by computer.

THE HALTING PROBLEM

Theorem : A_{TM} is undecidable.

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

- **Q:** Given the description of a TM and some input,
  - can we **determine** whether the machine accepts the input?

THE HALTING PROBLEM

Theorem : A_{TM} is undecidable.

$$A_{\text{TM}} = \{\langle M, w\rangle | \ M \text{ is a TM and } M \text{ accepts } w\}.$$

- **Q:** Given the description of a TM and some input,
  - can we **determine** whether the machine accepts the input?
- **A:** **Yes**. Just simulate/ Run the TM on the input.

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 ever enters its accept state, *accept*; if M ever enters its reject state, *reject*."

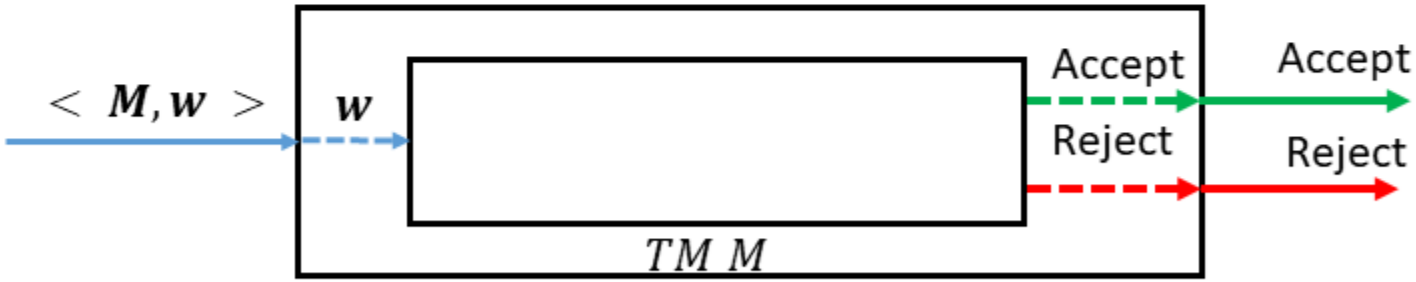
DECIDABILITY

Theorem : A_{TM} is undecidable.

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

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 ever enters its accept state, *accept*; if M ever enters its reject state, *reject*."



- This machine loops on input $\langle M, w\rangle$ if M loops on w. A_{TM} is Turing-recognizable.

DECIDABILITY

Theorem : A_{TM} is undecidable.

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

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 ever enters its accept state, *accept*; if M ever enters its reject state, *reject*."

- This machine loops on input $\langle M, w\rangle$ if M loops on w.
- A_{TM} is Turing-recognizable.

- Is A_{TM} decidable ?

THE DIAGONALIZATION METHOD

- The proof of the undecidability of A_{TM} uses a technique
  - called diagonalization
  - discovered by mathematician Georg Cantor in 1873.
  - Based on the problem of measuring the sizes of infinite sets.

THE DIAGONALIZATION METHOD

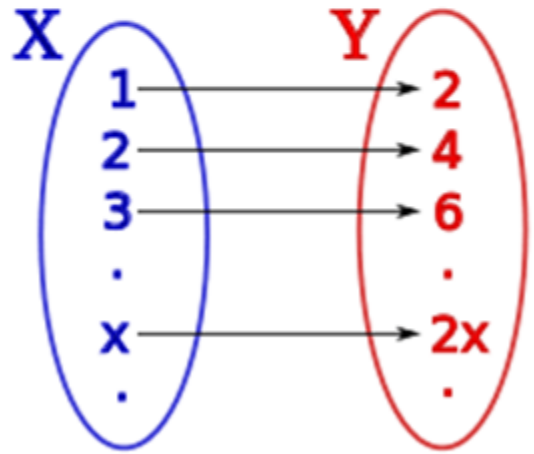
- Assume that we have two infinite sets, how can we tell
  - whether one is larger than the other or
  - whether they are of the same size?

- **For finite set:** by counting the elements in a finite set
- **For infinite set:** we can't use the counting method to determine the relative sizes of infinite sets.

THE DIAGONALIZATION METHOD

- Cantor observed that two **finite sets** have the same size
  - if the **elements** of one set can be paired with the **elements** of the other set.

- This method compares the sizes without resorting to counting.
- We can **extend** this idea to infinite sets.



RECALL: FUNCTIONS

- **Assume that we have sets A and B and a function f from A to B.**

- f is one-to-one : if it never maps two different elements to the same place
  - that is, if $f(a) \neq f(b)$ whenever $a \neq b$.

RECALL: FUNCTIONS

- **Assume that we have sets A and B and a function f from A to B.**

- f is onto : if it hits every element of B
  - that is, if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$.

RECALL: FUNCTIONS

- **Assume that we have sets A and B and a function f from A to B.**

- A and B are the same size : if there is a one-to-one, onto function $f: A \to B$.
  - or a correspondence.

  - In a correspondence :
    - every element of A **maps** to a unique element of B and
    - each element of B has a unique element of A mapping to it.

  - A correspondence is simply a way of pairing the **elements of A** with the **elements of B**.

THE DIAGONALIZATION METHOD

Example:

- Let N be the set of **natural numbers** {1,2,3,...} and let E be the set of **even natural numbers** {2,4,6,...}.

- The correspondence f mapping N to E is simply $f(n) = 2n$.

n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

COUNTABLE SET

- A set A is countable
  - if either it is finite or it has the same size as N.

Example:

- $Q = \{\frac{m}{n} | m, n \in N\}$
  - the set of positive rational numbers
  - **Q** seems to be **much larger than N**.

  - these two sets are the same size according to our definition.

COUNTABLE SET

Example: (Cont.)

- The ith row contains all numbers with numerator i and
- The jth column has all numbers with denominator j.

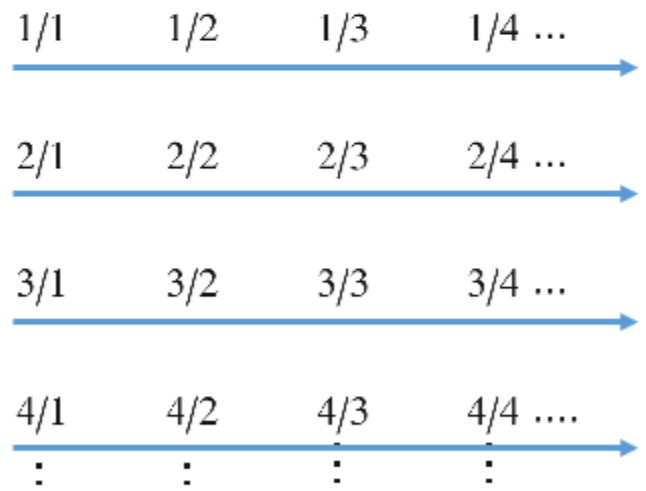
- Now we turn this matrix into a list.

$$\begin{array}{cccc} 1/1 & 1/2 & 1/3 & 1/4 \ldots \\ 2/1 & 2/2 & 2/3 & 2/4 \ldots \\ 3/1 & 3/2 & 3/3 & 3/4 \ldots \\ 4/1 & 4/2 & 4/3 & 4/4 \ldots. \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

COUNTABLE SET

Example: (Cont.)

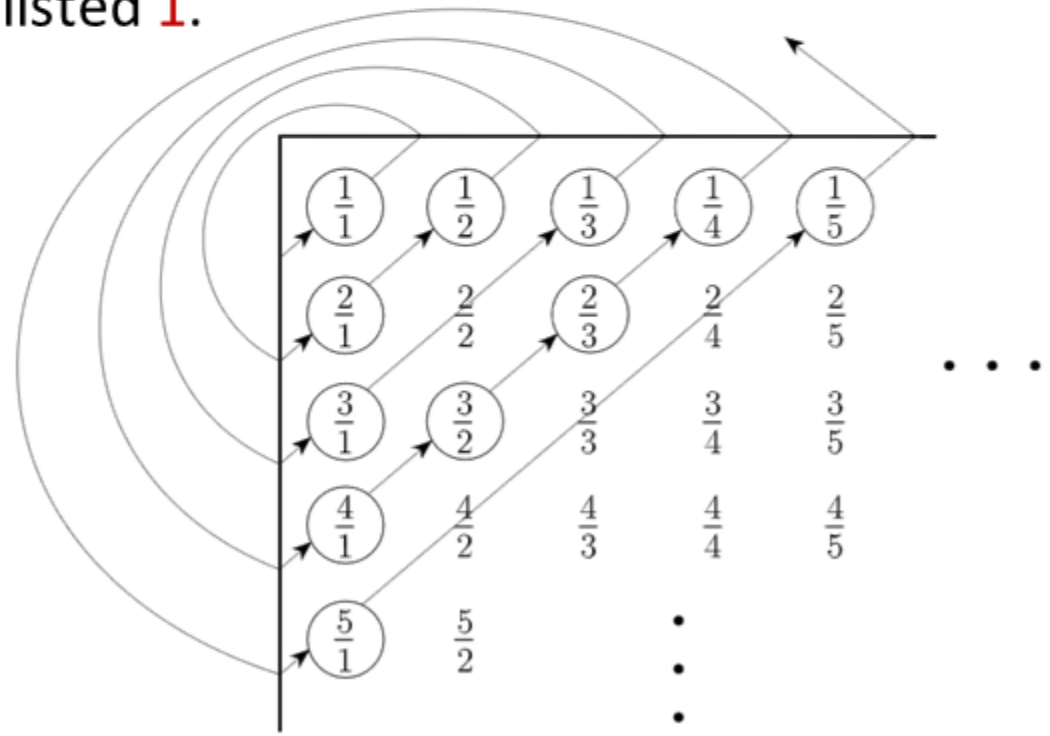
- **Approach 1:**
  - listing row by row
    - Never goes to the 2nd row



COUNTABLE SET

Example: (Cont.)

- **Approach 2:**
- listing diagonal Based on i+j
- And removing the redundant values
  - Such as 2/2, since we had already listed 1.



UNCOUNTABLE SET

- F or some infinite sets, no correspondence with N exists.
- These sets are simply too big.
- Such sets are called uncountable.

- The set of real numbers is an example of an **uncountable** set.

- The numbers $\pi = 3.1415926$ … and $\sqrt{2}$=1.4142135…are examples of real numbers.

UNCOUNTABLE SET

- Theorem: R is uncountable.

- Proof:
  - we show that no correspondence exists between N and R.
  - The proof is by contradiction.

UNCOUNTABLE SET

- Proof (Cont.):
  - Suppose that a correspondence f existed between N and R.
  - f must pair all the members of N with all the members of R.
  - But we will **find** an x in R that is not paired with anything in N.
  - The way we find this x is by actually constructing it.

UNCOUNTABLE SET

- Proof Idea (cont.):
  - The following table shows a few values of a hypothetical correspondence f between N and R.
  - We will show that there is an x in R that
    - **is not f(n) for any n**.

n	$f(n)$
1	3.14159...
2	55.55555...
3	0.12345...
4	0.50000...
⋮	⋮

UNCOUNTABLE SET

- Proof Idea (cont.):
  - **How to construct the desired x ?**
  - It is a number between 0 and 1,
    - so all its **significant digits** are fractional digits following the decimal point.

n	$f(n)$
1	$3.\underline{1}4159\ldots$
2	$55.5\underline{5}555\ldots$
3	$0.12\underline{3}45\ldots$
4	$0.5000\underline{0}\ldots$
$\vdots$	$\vdots$

UNCOUNTABLE SET

- Proof Idea (cont.):
  - How to construct the desired x ?
  - It is a number between 0 and 1,
    - so all its **significant digits** are fractional digits following the decimal point.
  - To ensure that $x \neq f(1)$, we let the first digit of x be any thing different from the first fractional digit 1 of f(1)=3.14159
    - Arbitrarily, we let it be 4.

n	$f(n)$
1	3.14159...
2	55.55555...
3	0.12345...
4	0.50000...
⋮	⋮

X = 0.4

UNCOUNTABLE SET

- Proof Idea (cont.):
  - How to construct the desired x ?
  - It is a number between 0 and 1,
    - so all its **significant digits** are fractional digits following the decimal point.
  - To ensure that $x \neq f(1)$, we let the first digit of x be any thing different from the first fractional digit 1 of f(1)=3.14159
    - Arbitrarily, we let it be 4.
  - The 2nd fractional digit of f(2)=55.555555..is 5, so we let x be anything different—say,6.
  - We know that x is not f(n) for any n.

n	$f(n)$
1	3.14159...
2	55.55555...
3	0.12345...
4	0.50000...
⋮	⋮

X = 0.46

UNCOUNTABLE SET

- Proof Idea (cont.):
  - How to construct the desired x ?
  - It is a number between 0 and 1,
    - so all its **significant digits** are fractional digits following the decimal point.
  - To ensure that $x \neq f(1)$, we let the first digit of x be any thing different from the first fractional digit 1 of f(1)=3.14159
    - Arbitrarily, we let it be 4.
  - The 2nd fractional digit of f(2)=55.555555..is 5, so we let x be anything different—say,6.
  - we select all of the other digits in the **same way**.
  - We know that x is not f(n) for any n.

n	$f(n)$
1	3.14159...
2	55.55555...
3	0.12345...
4	0.50000...
⋮	⋮
X =	0.4652...

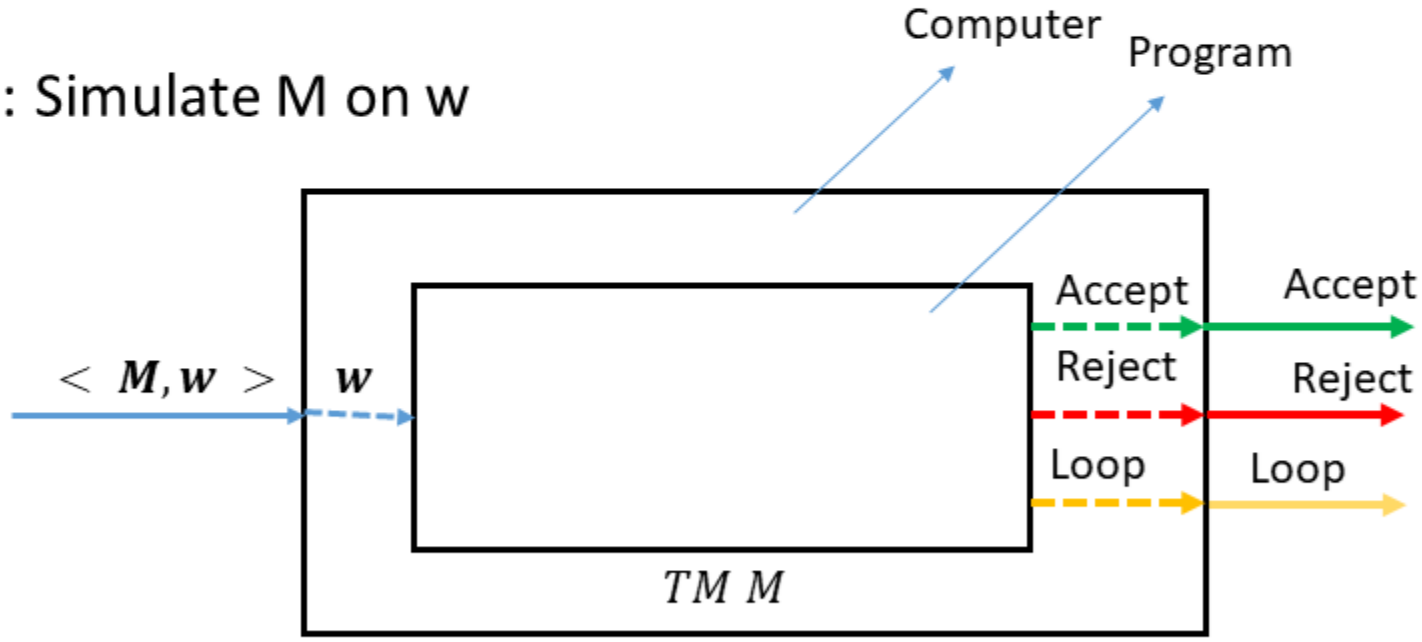
UNCOUNTABLE SET

- Corollary: The set of irrational numbers is uncountable.
- Proof:
  - **By using the same technique**

THE UNIVERSAL TURING MACHINE

- **Input :** $< M, w >$
  - M: the description of some TM
  - w: an input string for M

- **Action**: Simulate M on w



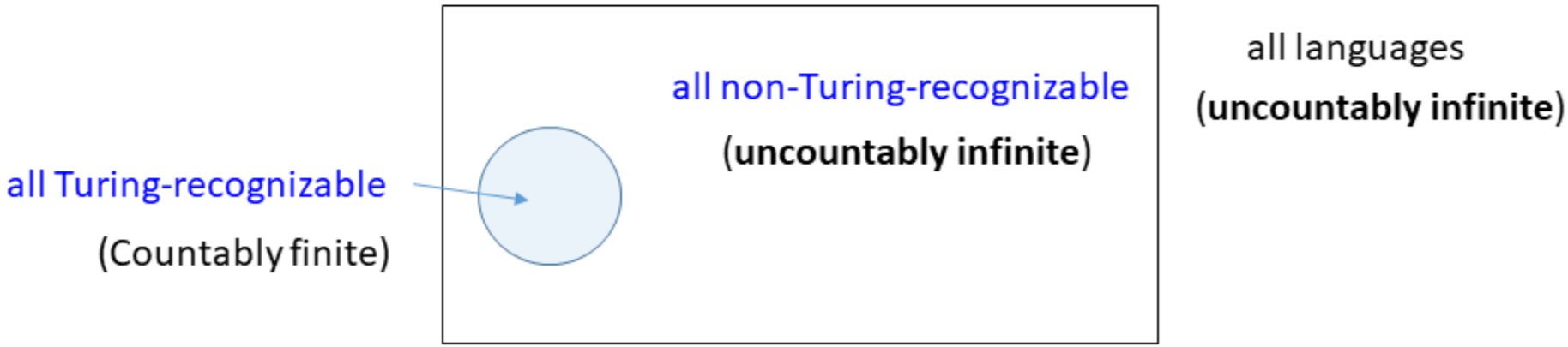
UTM (Universal Turing Machine)

THE HALTING PROBLEM

- Corollary: Some languages are not Turing-recognizable.

- Proof:

- **Part a)** We first show that the set of all Turing machines is countable.
- **Part b)** Then, we show that the set of all languages is uncountable.
- **Part c)** Finally, we will prove that Some languages are not Turing-recognizable.



UNCOUNTABLE SET

- **Part a) The set of all Turing machines is countable.**
  - Every **Turing machine** can be encoded into a string of finite length, Because:
    - The number of the states are finite.
    - The input alphabet is finite.
    - The tape alphabet is finite.
    - The transitions are in form of $a \rightarrow b, dir$, which a, b, and dir are from finite sets.

UNCOUNTABLE SET

- Part a) The set of all Turing machines is countable.
  - Every **Turing machine** can be encoded into a string of finite length, Because:
    - The number of the states are finite.
    - The input alphabet is finite.
    - The tape alphabet is finite.
    - The transitions are in form of a → b, dir , which a, b, and dir are from finite sets.

- Any string is a valid TM representation or NOT.

- How to list the TMs:
  - Generate all string, one after the other
  - Check to see if it is a valid TM.

UNCOUNTABLE SET

- **Part b) The set of all languages is not countable.**

- Theorem : The set of all infinite length strings over {0,1} is uncountable infinite.
- Proof: by Diagonalization method

UNCOUNTABLE SET

- **Part b) The set of all languages is not countable.**

- Theorem : The set of all infinite length strings over {0,1} is uncountable infinite.
- Proof: by Diagonalization method

  - **Assume** the set of infinite binary strings can be enumerated. (correspondence with N)

UNCOUNTABLE SET

- Part b) The set of all languages is not countable.

- Theorem : The set of all infinite length strings over {0,1} is uncountable infinite.
- Proof: by Diagonalization method

  - **Assume** the set of infinite binary strings can be enumerated. (correspondence with N)

- We can construct an infinite length binary string x which is not equal to any string on this list.
- That is a contradiction.

- Then, the set of all languages is not countable.

1 →	0	0	0	0	0	1	…
2 →	1	1	1	0	0	1	…
3 →	0	0	0	1	1	0	…
4 →	1	0	0	0	1	0	…
5 →	0	0	0	1	1	1	…
6 →	1	0	0	0	1	1	…
X =	1	0	1	1	0	0	…

UNCOUNTABLE SET

- **Part c)** Finally, we will prove that some languages are not Turing-recognizable.

UNCOUNTABLE SET

- **Part c)** Finally, we will prove that Some languages are not Turing-recognizable.
- The set of all finite length string over $\Sigma = \{0,1\}$ is countable.

$$\Sigma^* = \{\ \varepsilon,\ \ 0,\ \ 1,\ \ 00,\ \ 01,\ \ 10,\ \ 11,\ 000,\ 001,\ \cdots\ \}\ ;$$

Lexicographical order

UNCOUNTABLE SET

- **Part c)** Finally, we will prove that Some languages are not Turing-recognizable.
- The set of all finite length string over $\Sigma = \{0,1\}$ is countable.
- **Language A** contains some of these strings and not others.

$$A = \{\quad 0,\quad 00,\quad 01,\quad 000,\ 001,\ \cdots\ \}\ ;$$

$$\Sigma^* = \{\ \varepsilon,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ 001,\ \cdots\ \}\ ;$$

UNCOUNTABLE SET

- **Part c)** Finally, we will prove that Some languages are not Turing-recognizable.
- The set of all finite length string over $\Sigma = \{0,1\}$ is countable.
- **Language A** contains some of these strings and not others.

Lexicographical order

- **Language A** can be fully specified by giving an infinite length binary string.

$$\Sigma^* = \{\ \varepsilon,\ \ 0,\ \ 1,\ \ 00,\ \ 01,\ \ 10,\ \ 11,\ \ 000,\ 001,\ \cdots\ \}\ ;$$
$$A = \{\ \ \ \ 0,\ \ \ \ 00,\ \ 01,\ \ \ \ \ \ \ \ 000,\ 001,\ \cdots\ \}\ ;$$
$$\chi_A = \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad \cdots \quad .$$

characteristic sequence of A

- In X_A, $a_i = (0)1$, means that A (doesn't) contain(s) the ith element of Σ^*.

UNDECIDABILITY

- Let L be the set of all languages over alphabet Σ.
- Let B be the set of all infinite binary sequences.

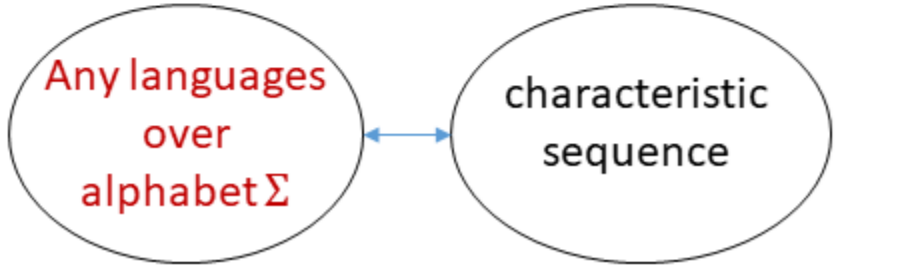
L B : uncountable

UNDECIDABILITY

- Let L be the set of all languages over alphabet Σ.
- Let B be the set of all infinite binary sequences.

- The function $f: L \to B$, where $f(A)$ equals the characteristic sequence of A, is one-to-one and onto, and hence is a correspondence.
- Therefore, as B is uncountable, L is uncountable as well.

$$\begin{array}{rccccccccccc}
\Sigma^* = \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & 001, & \cdots & \} ; \\
A = \{ & & 0, & & 00, & 01, & & & 000, & 001, & \cdots & \} ; \\
\chi_A = & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots & .
\end{array}$$



UNDECIDABILITY

- Let L be the set of all languages over alphabet Σ.
- Let B be the set of all infinite binary sequences.

- The function $f: L \to B$, where $f(A)$ equals the characteristic sequence of A, is one-to-one and onto, and hence is a correspondence.
- Therefore, as B is uncountable, L is uncountable as well.

THE HALTING PROBLEM

- Now we are ready to prove the following Theorem:

- Theorem : A_{TM} is undecidable.

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

THE HALTING PROBLEM

- Now we are ready to prove the following Theorem:

- Theorem : A_{TM} is undecidable.

$$A_{\text{TM}} = \{\langle M, w\rangle |\ M \text{ is a TM and } M \text{ accepts } w\}.$$

- Proof: We **assume** that A_{TM} is decidable and obtain a contradiction.

THE HALTING PROBLEM

- Now we are ready to prove the following Theorem:

- Theorem : A_{TM} is undecidable.

$$A_{\text{TM}} = \{\langle M, w\rangle |\ M \text{ is a TM and } M \text{ accepts } w\}.$$

- Proof: We **assume** that A_{TM} is decidable and obtain a contradiction.

- **Suppose** that H is a decider for A_{TM}.
- On input ⟨M, w⟩, where M is a TM and w is a string,

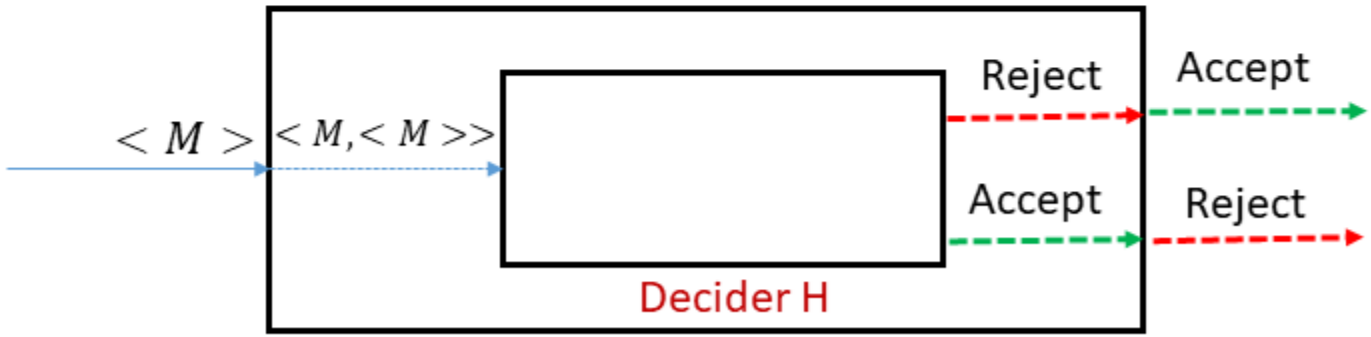
$$H(\langle M, w\rangle) = \begin{cases} accept & \text{if } M \text{ accepts } w \\ reject & \text{if } M \text{ does not accept } w. \end{cases}$$

THE HALTING PROBLEM

- Proof(Cont.):
- Now we construct a new Turing machine D with H as a subroutine.

D = "On input $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, *reject*; and if H rejects, *accept*."



TM D

THE HALTING PROBLEM

- Proof(Cont.):
- Now we construct a new Turing machine D with H as a subroutine.

D = "On input $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, *reject*; and if H rejects, *accept*."

- In summary:

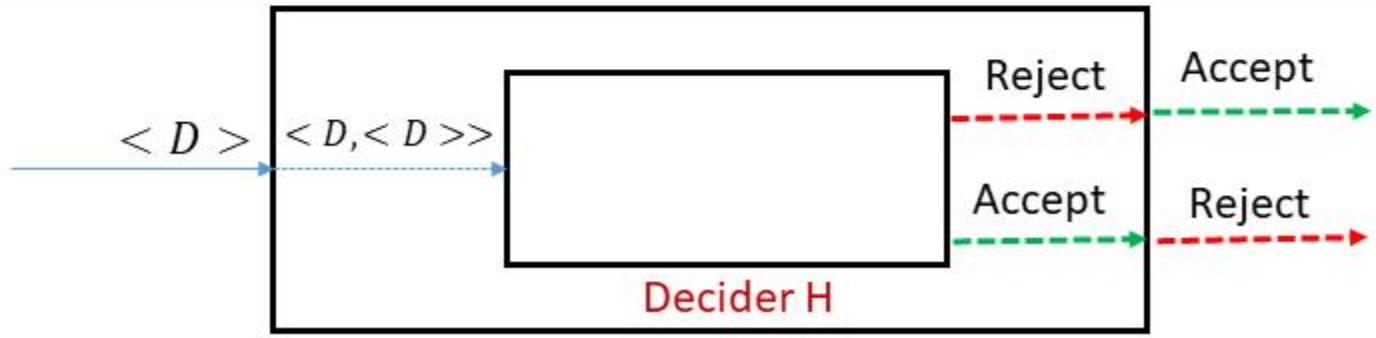
$$D(\langle M \rangle) = \begin{cases} accept & \text{if } M \text{ does not accept } \langle M \rangle \\ reject & \text{if } M \text{ accepts } \langle M \rangle. \end{cases}$$

THE HALTING PROBLEM

- Proof(Cont.):
- What happens when we run D with its own description ⟨D⟩ as input?

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

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



TM D

THE HALTING PROBLEM

- Proof(Cont.):
- What happens when we run D with its own description ⟨D⟩ as input?

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

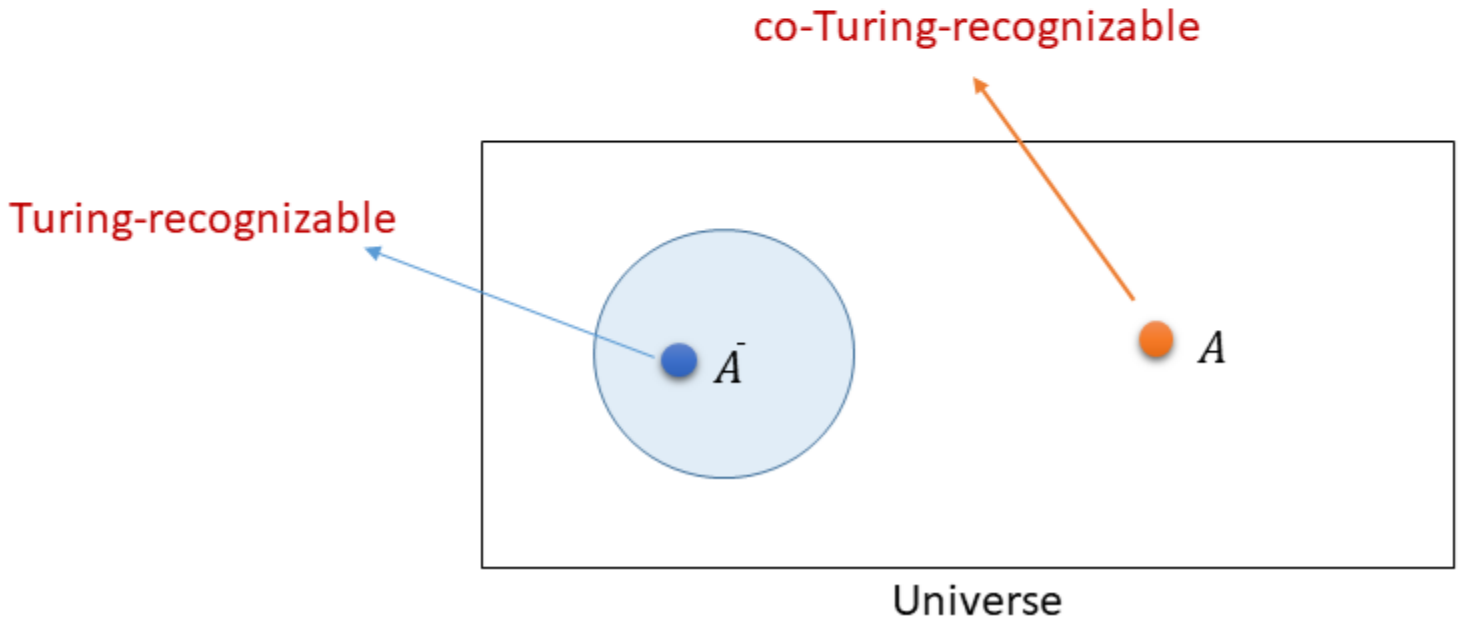
$$D(\langle D \rangle) = \begin{cases} accept & \text{if } D \text{ does not accept } \langle D \rangle \\ reject & \text{if } D \text{ accepts } \langle D \rangle. \end{cases}$$

- **No matter what D does**, it is forced to do the opposite,
  - which is obviously a contradiction.

- Thus, neither TM D nor TM H can exist.

CO-TURING-RECOGNIZABLE

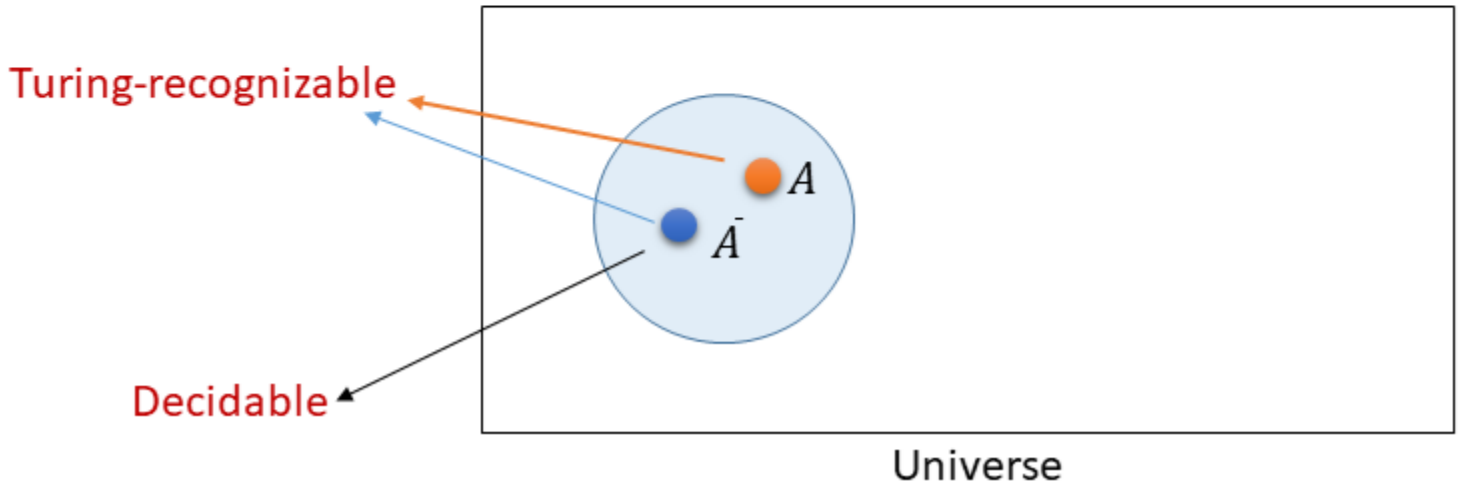
- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.



CO-TURING-RECOGNIZABLE

- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

- Theorem: A language is decidable **iff** it is Turing-recognizable and co-Turing-recognizable.



CO-TURING-RECOGNIZABLE

- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

- Theorem: A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.

Proof:

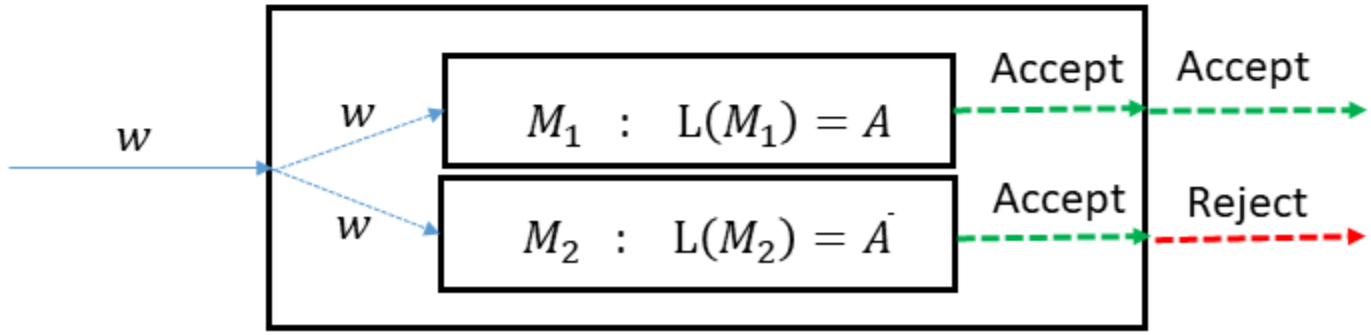
- If part:
- Any decidable language is Turing-recognizable.
- and the complement of a decidable language also is decidable.
  - So, it will be Turing-recognizable.

CO-TURING-RECOGNIZABLE

Proof (Cont.):

- Else-If part:
- if both A and $\bar{A}$ are Turing-recognizable, we let M_1 be the recognizer for A and M_2 be the recognizer for $\bar{A}$.
- The following Turing machine M is a decider for A.

> 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*."



TM M

(M has two tapes)

CO-TURING-RECOGNIZABLE

Theorem:

- $\overline{A_{TM}}$ is not Turing-recognizable.

CO-TURING-RECOGNIZABLE

Theorem:

- $\overline{A_{TM}}$ is not Turing-recognizable.

Proof:

- We know that $\boldsymbol{A_{TM}}$ is Turing-recognizable.
- If $\overline{\boldsymbol{A_{TM}}}$ also were Turing-recognizable,
-

$\boldsymbol{A_{TM}}$ would be decidable.

- We proved that A_{TM} is not decidable,
  - so $\overline{A_{TM}}$ must not be Turing-recognizable