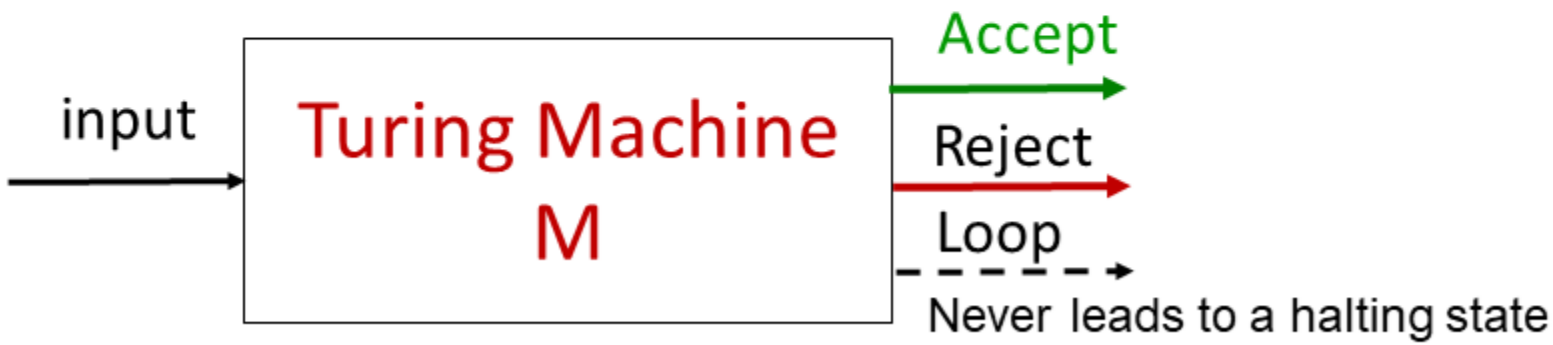
Part Two: Chapter 4

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.

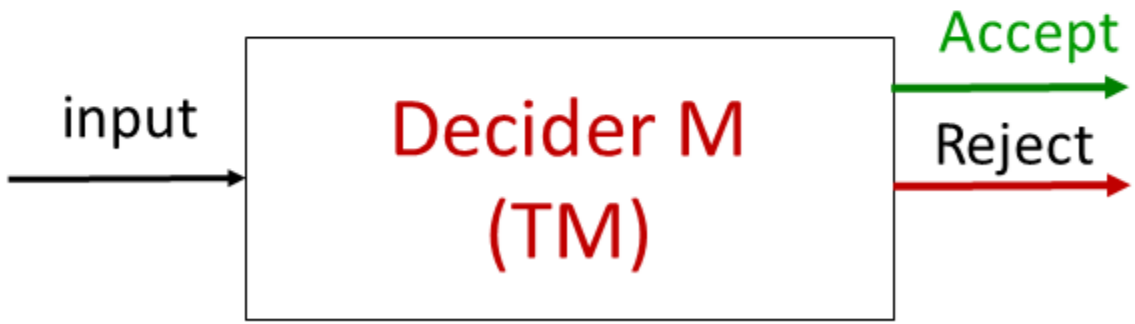


Recall: Decider

- 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

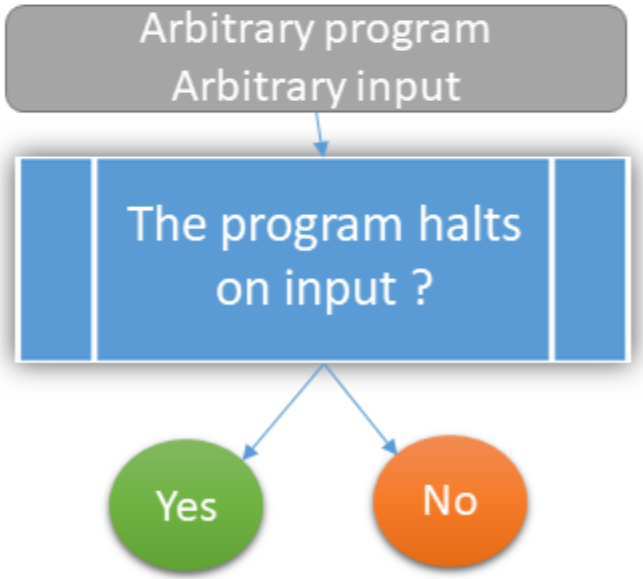
- Every question about regular languages is decidable.
- Some questions about Context-free languages are decidable,
  - But some are not.
- The "HALTING PROBLEM" is not decidable.
- Some languages are not Turing recognizable.
- Many **questions about Turing Machines** are not decidable.

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.

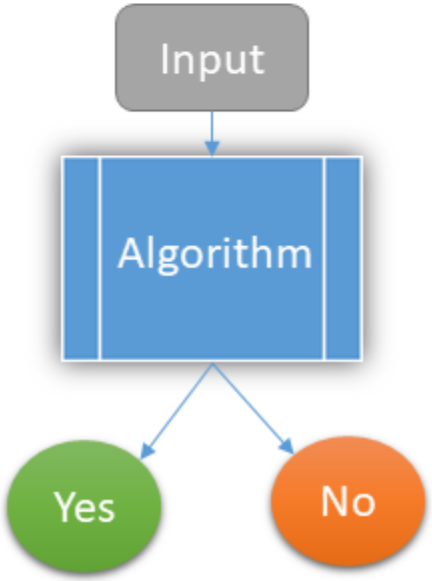
DECISION PROBLEMS

- 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

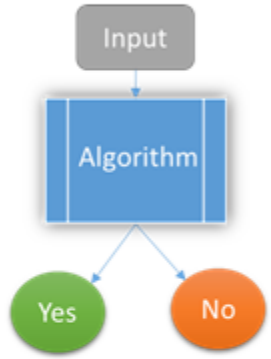
- We may be interested in following questions:
  - Is a given natural number prime?
  - Is an undirected graph connected?
  - Does the computation of TM halt before 10th transition?

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



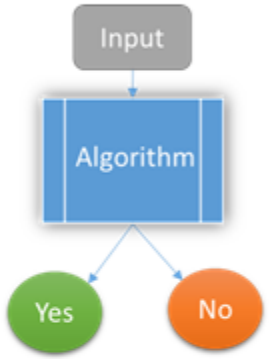
DECISION PROBLEMS

- 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*."

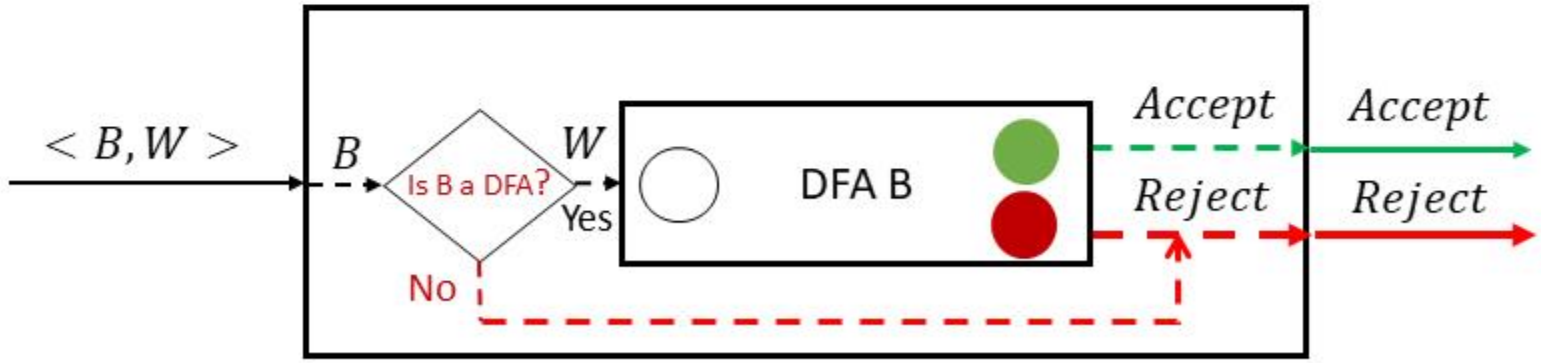
THE ACCEPTANCE PROBLEM FOR TMs

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:

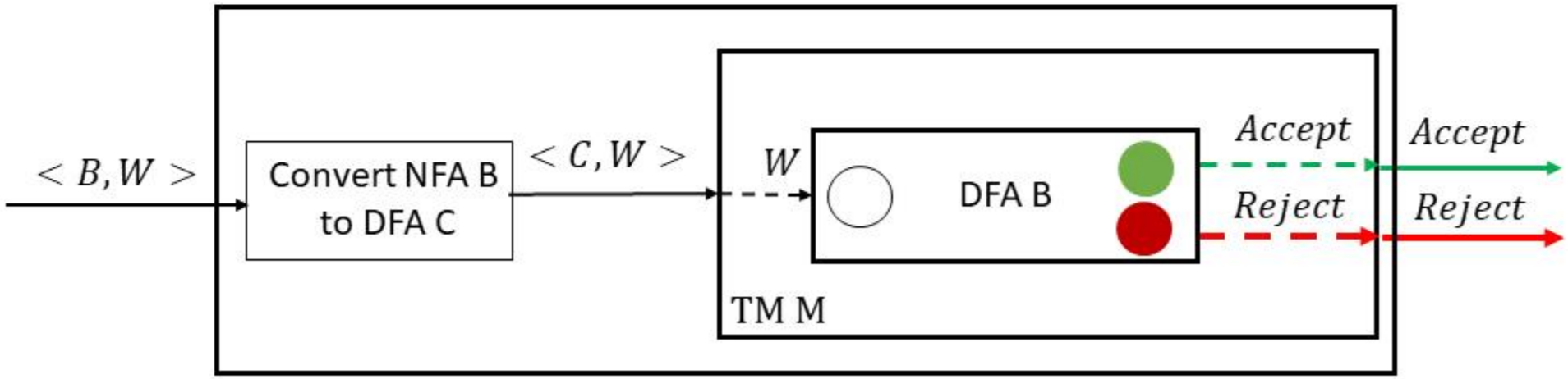
1. Convert NFA B to an equivalent DFA C, using the procedure for this conversion given in Theorem 1.39.
2. Run TM M from Theorem 4.1 on input $\langle C, w\rangle$.
3. If M accepts, *accept*; otherwise, *reject*."

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.

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

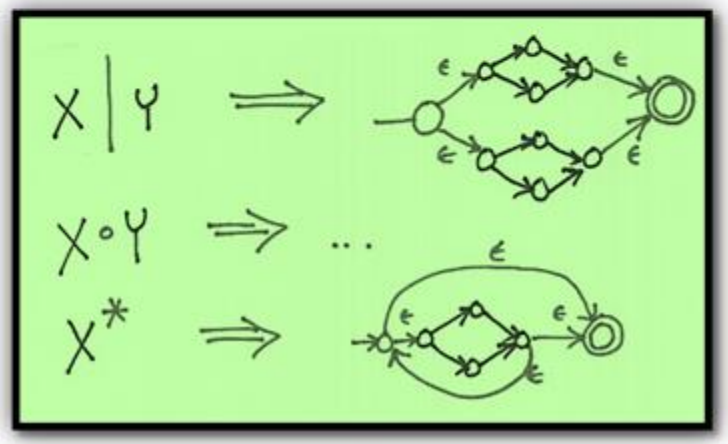
THE ACCEPTANCE PROBLEM FOR TMs

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*."

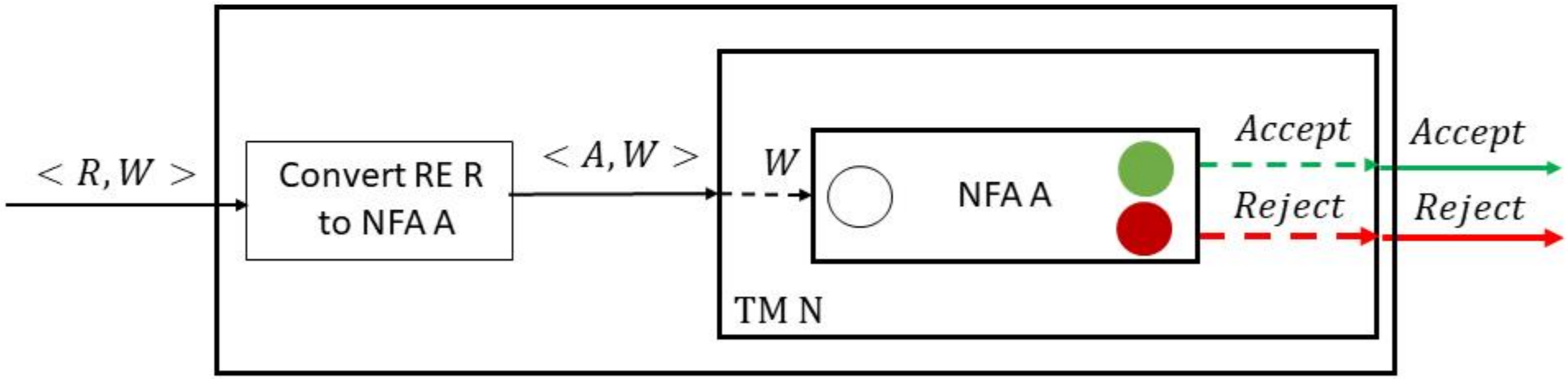


THE ACCEPTANCE PROBLEM FOR TMs

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)
-

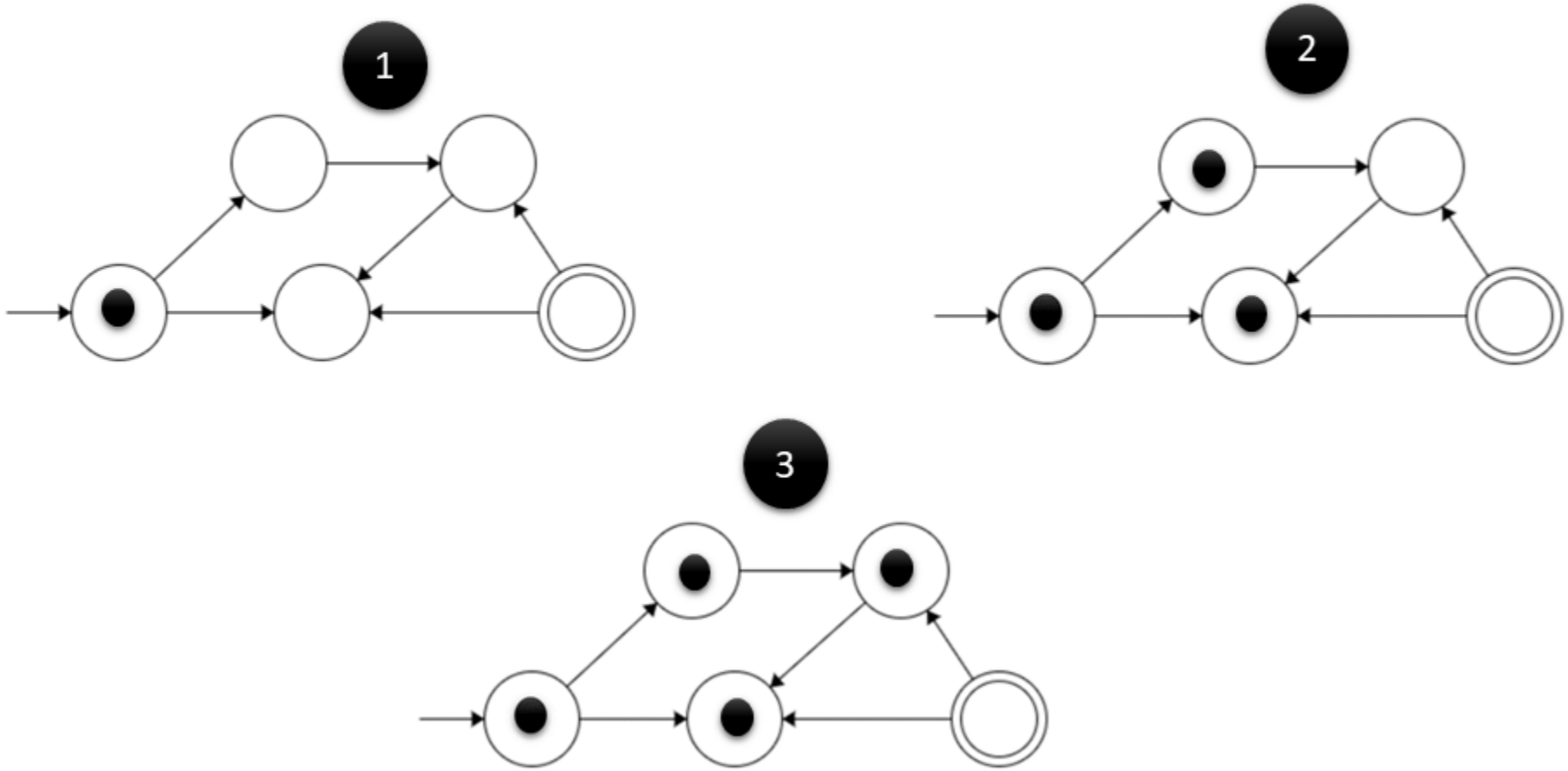
DECIDABILITY

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.

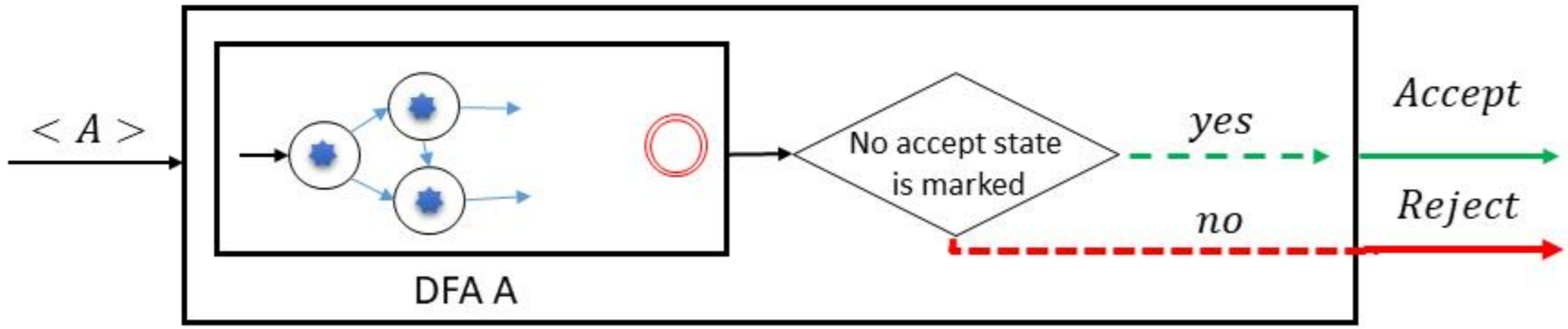
DECIDABILITY

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.

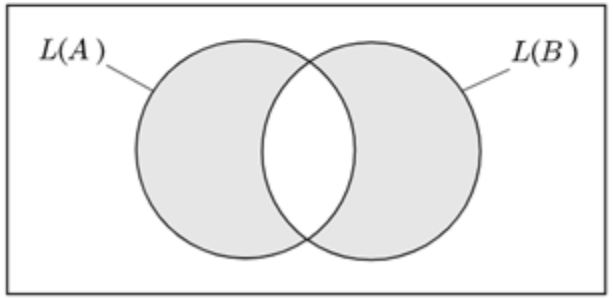
DECIDABILITY

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.



symmetric difference of L(A) and 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:

- 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