CS 6041
Theory of Computation

Decidability

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Revisit: Turing-recognizable and Turing-decidable

- Turing-recognizable: A=L(M)
  - $x \in A$, M accept x
  - $x \notin A$, M reject x or loop
- Turing-decidable: A=L(M)
  - $x \in A$, M accept x
  - $x \notin A$, M reject x

Halt no matter what is the input

- Turing-recognizable $\neq$ Turing-decidable

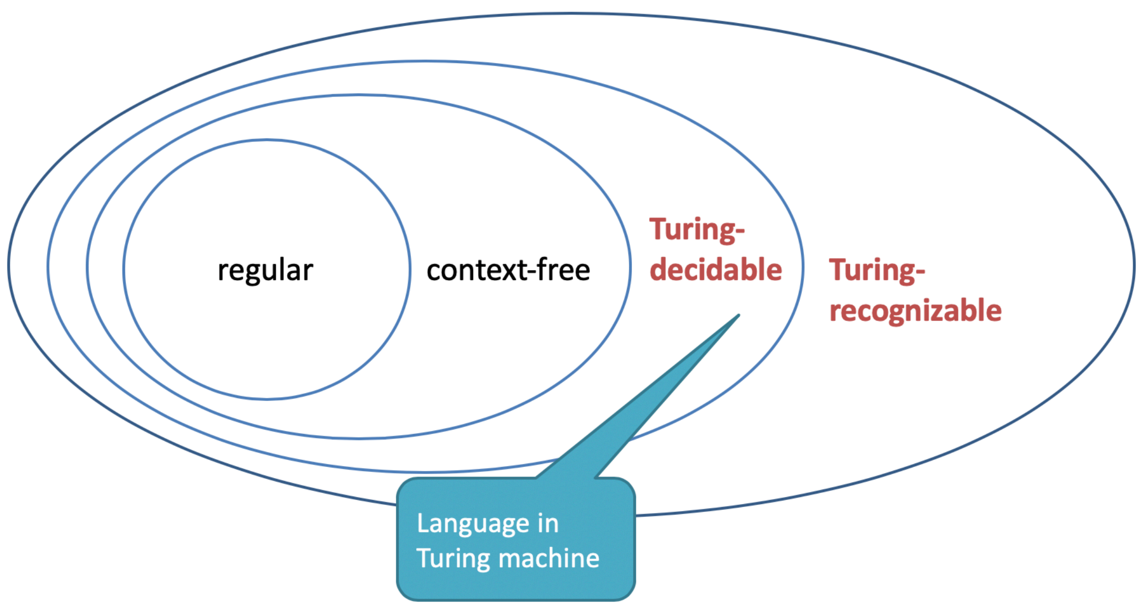
Revisit: The output of Turing Machine

- Accept
- Reject
- Loop

Halt -> Decidable

= Never Halt

Recognizable



How to prove a language is decidable

- Create a Turing machine M
  - For each $x \in L$, M either accept or reject x

- If such a TM exists, language L is decidable

Decidable problems concerning regular languages

1. Acceptance problem for DFAs
   - whether a DFA accepts a string
2. Acceptance problem for NFAs
   - whether a NFA accepts a string
3. Regular expression decidability
   - Whether a regular expression generates a string
4. Emptiness testing for DFAs
   - Whether a DFA is empty
5. Equivalence of DFAs
   - Whether two DFAs recognize the same language

1. Acceptance problem for DFAs

- whether a particular DFA accepts a given string

- Language
  - $A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
  - DFA B accept w $\Leftrightarrow$ $\langle B, w\rangle \in A_{DFA}$

THEOREM 4.1

- A_{DFA} is a decidable language.

- Proof idea:

design a M to decide $A_{DFA} = \{\langle B, w\rangle \mid B$ is a DFA that accepts input string $w\}$

M simulates B on input w

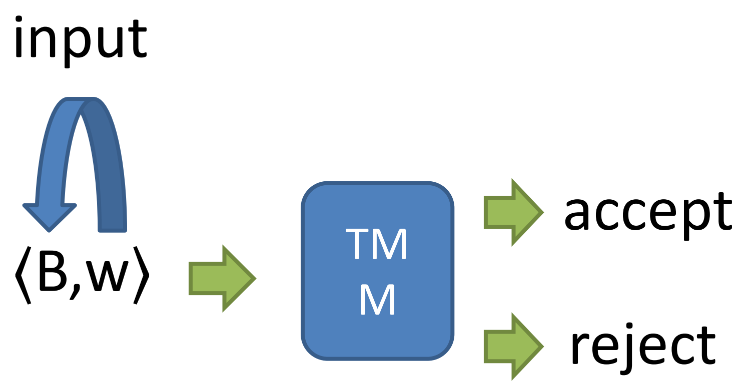
The input on M is <B, w>

M is a TM

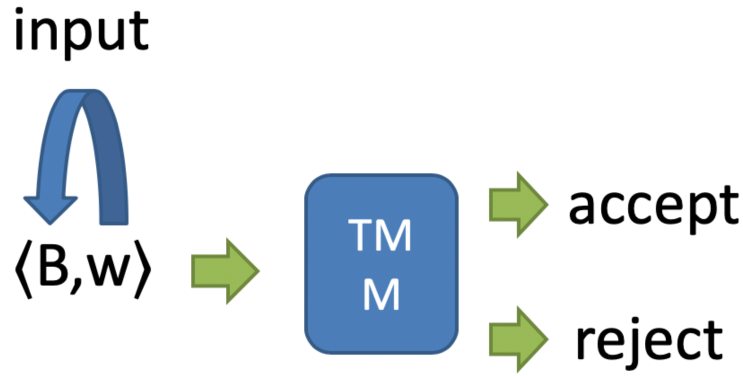
B is a DFA

w is a string

The input on B is w



THEOREM 4.1 proof



- Proof:

design a M to decide A_{DFA}

How to simulate B?

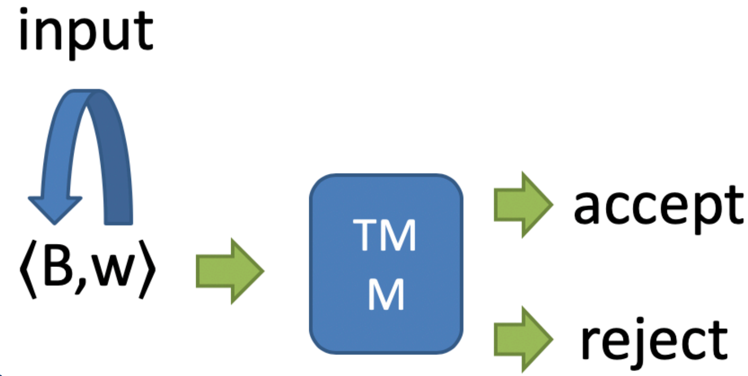
M = "On input ⟨B, w⟩, where B is a DFA and w is a string:

(1) Simulate B on input w.

(2) If the simulation ends in an accept state of B, accept.

If it ends in a nonaccepting state of B, reject."

THEOREM 4.1 proof (details)



- Proof:

TM M first check the input format <B, w>

If w is not string or B is not (Q,Σ,δ,q_0,F), reject

M carries out the simulation

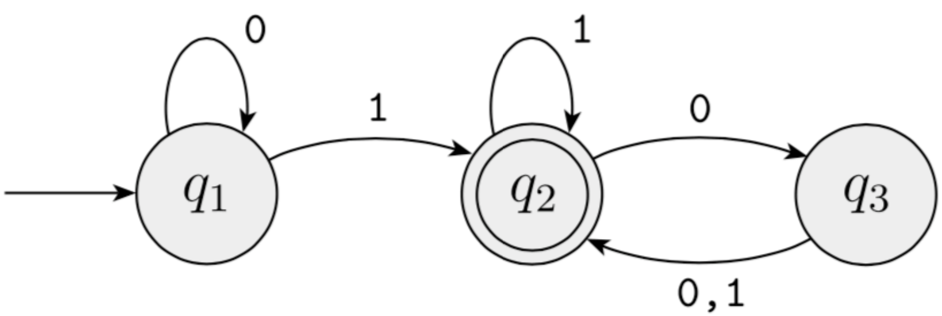
(1) M keeps track of B's current state and B's current position in the input w by writing this information down on its tape

(2) The states and position are updated according to the specified transition function δ

(3) When M finishes processing the last symbol of w, M accepts the input if B is in an accepting state; M rejects the input if B is in a nonaccepting state.

Example

B:

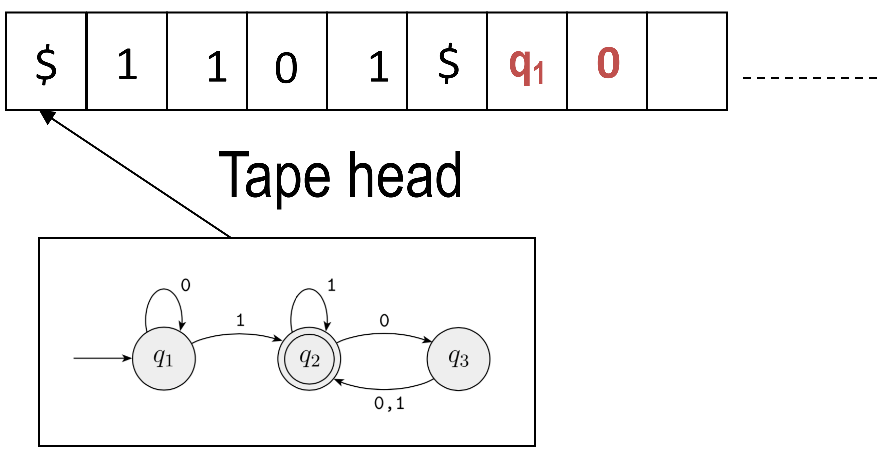


w: 1101

Input

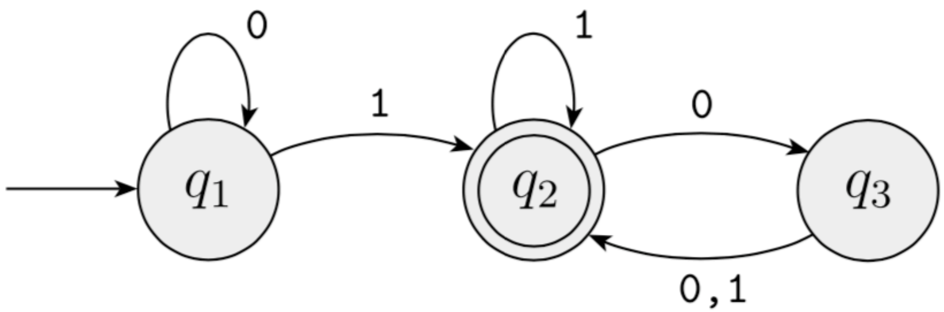
TM:

control



Example

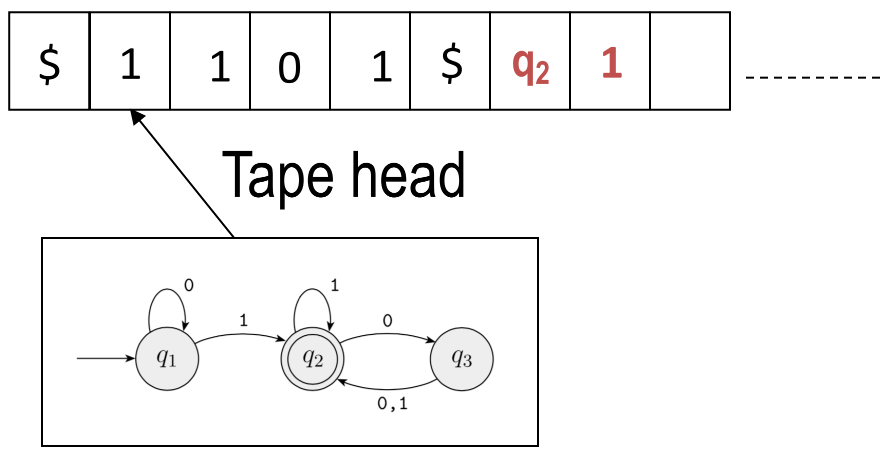
B:



w: 1101

TM:

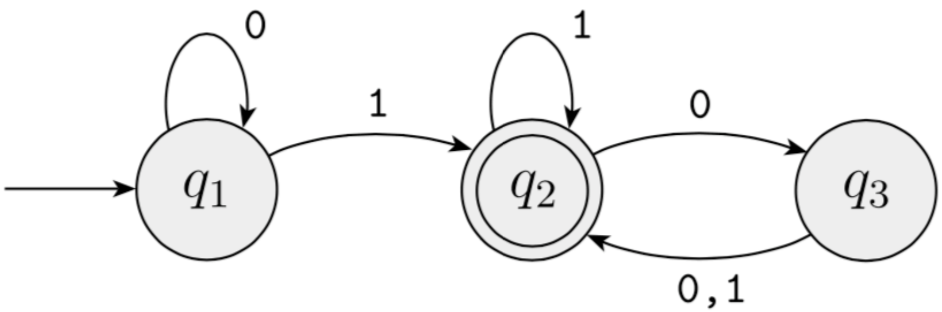
Input



control

Example

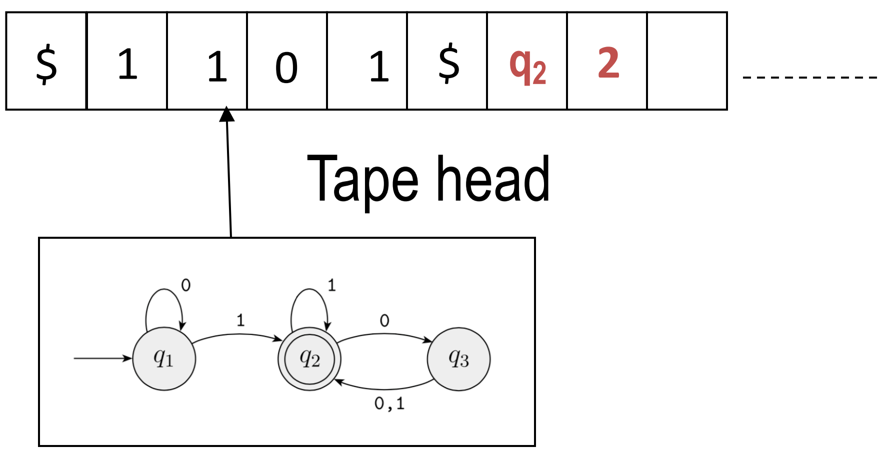
B:



w: 1101

TM:

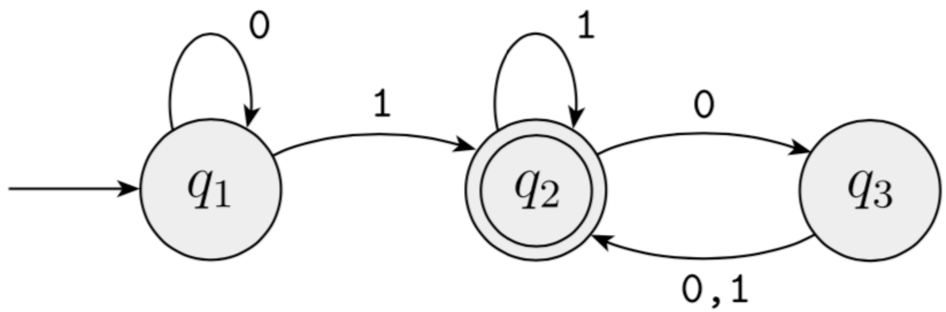
Input



control

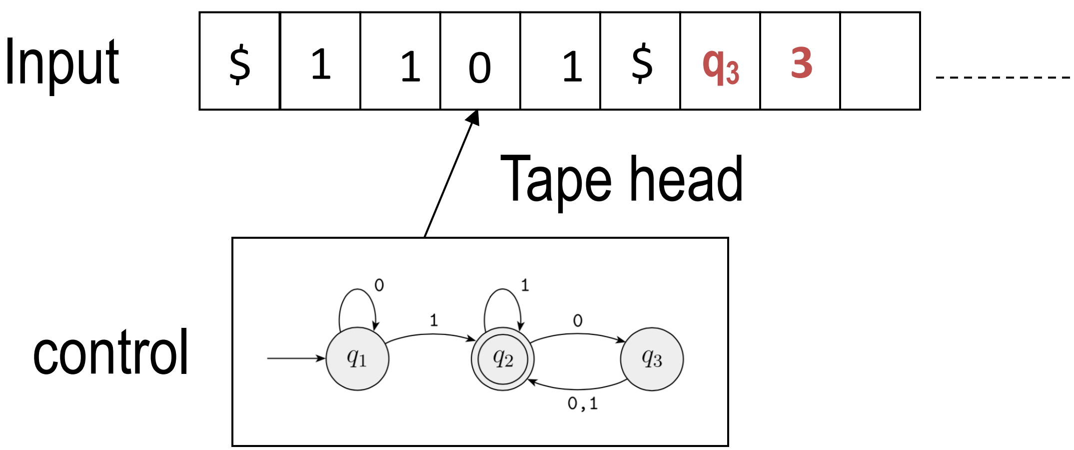
Example

B:



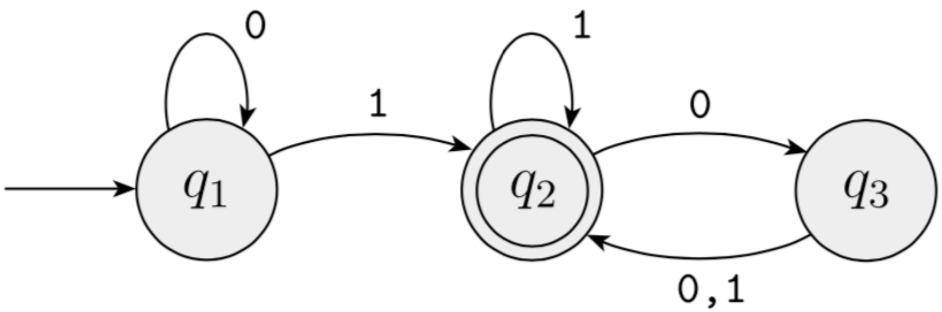
w: 1101

TM:



Example

B:



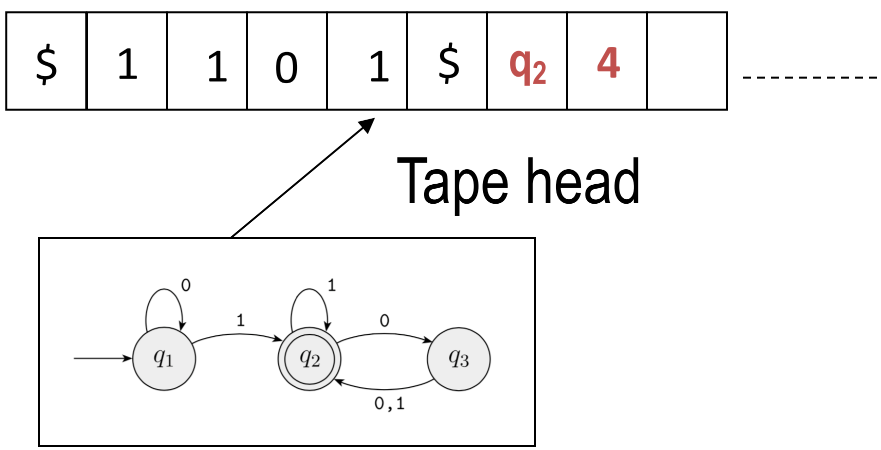
w: 1101

Input

Tape head

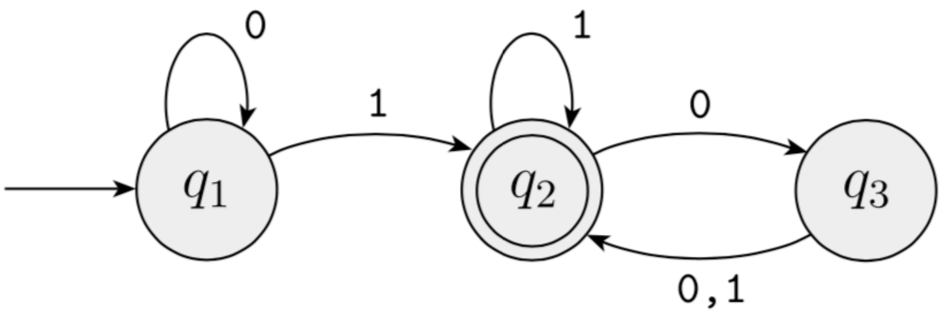
TM:

control



Example

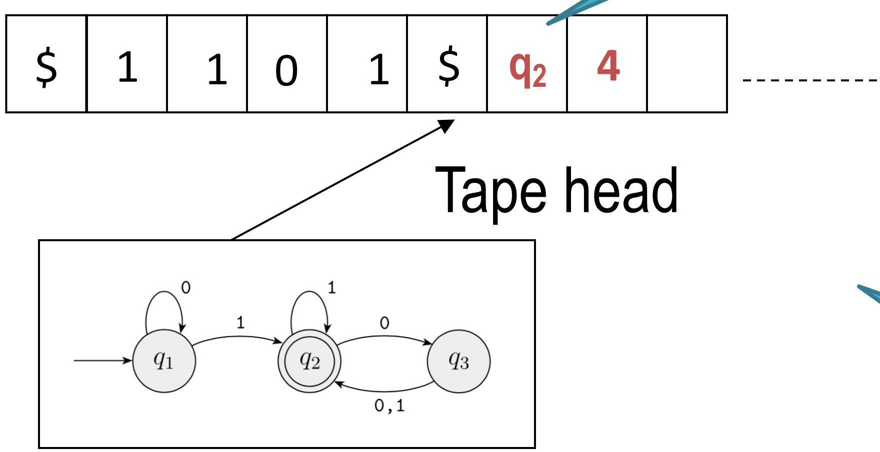
B:



w: 1101

B accepts on q_2

Input



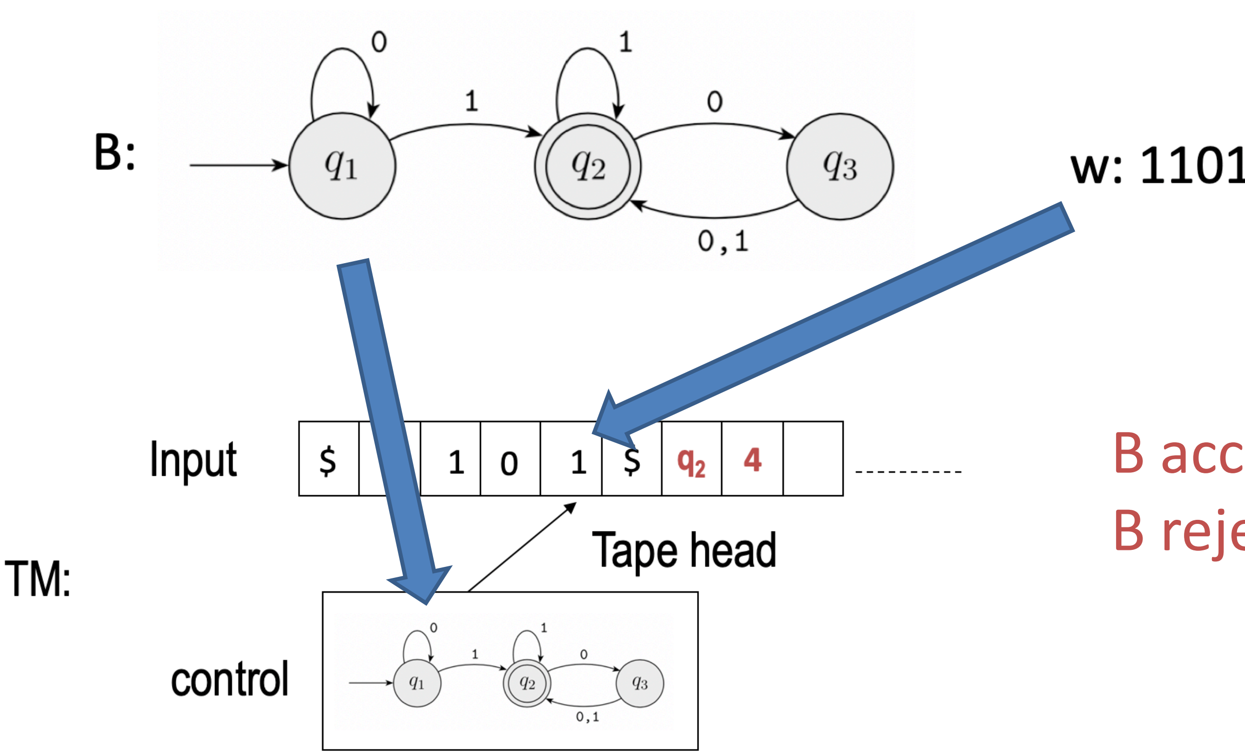
TM:

control

M outputs accept

1. Acceptance problem for DFAs

- A_{DFA} is a decidable language.
- A_{DFA} = {⟨B, w⟩| B is a DFA that accepts input string w}

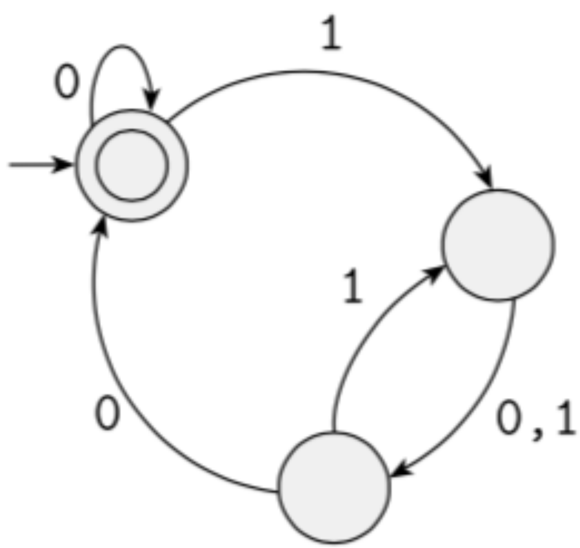


B accepts => M accepts

B rejects => M rejects

Question: True or False?

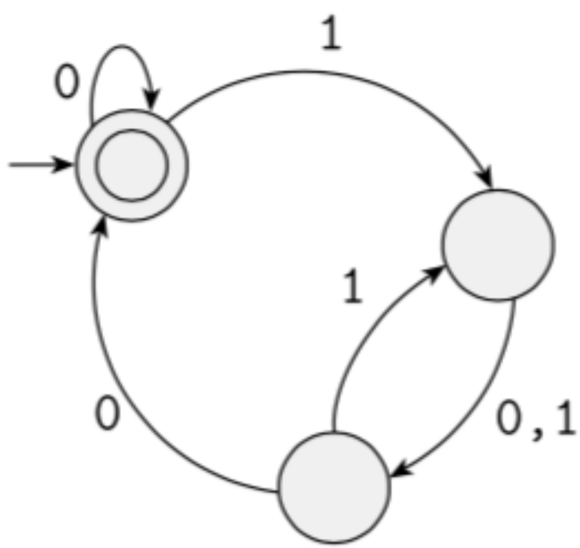
- For the following DFA M, $\langle M, 0100 \rangle \in A_{DFA}$



True

Question: True or False?

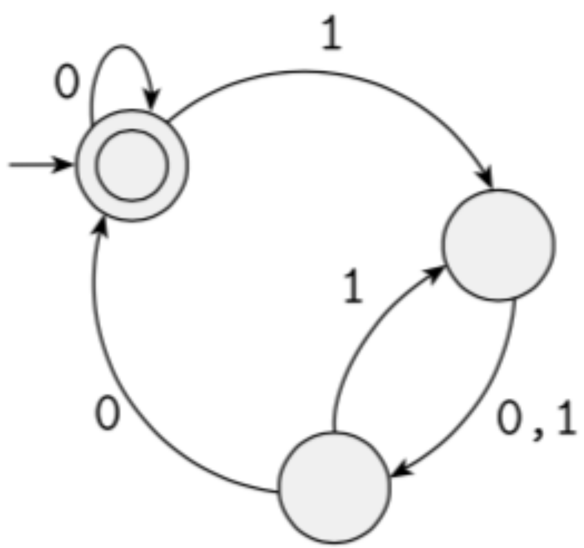
- For the following DFA M, $\langle M, 011\rangle \in A_{DFA}$



False

Question: True or False?

- For the following DFA M, $\langle M, \varepsilon \rangle \in A_{DFA}$



False

2. Acceptance problem for NFAs

- Acceptance problem for NFAs
  - whether a particular non-deterministic finite automaton accepts a given string

- Language
  - $A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$
  - NFA B accept w $\Leftrightarrow$ $<B, w> \in A_{NFA}$

Theorem 4.2

- A_{NFA} is a decidable language.

- Proof idea:

  First, transform NFA B into an equivalent DFA C

  Then use TM N to simulate C on input w

THEOREM 4.2 proof

- Proof:

design a N to decide A_{NFA}

N = "On input ⟨B, w⟩, where B is a NFA and w is a string:

(1) Transform NFA B into an equivalent DFA C

(2) Execute TM M on input <C, w>

(3) If C accepts, accept; otherwise, reject

3. Regular expression decidability

- Regular expression decidability
  - whether a regular expression generates a given string

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

Theorem 4.3

- A_{REX} is a decidable language.

- Proof :

  The following TM P decides A_{REX}.

  P = "On input ⟨R, w⟩, where R is a regular expression and w is a string:

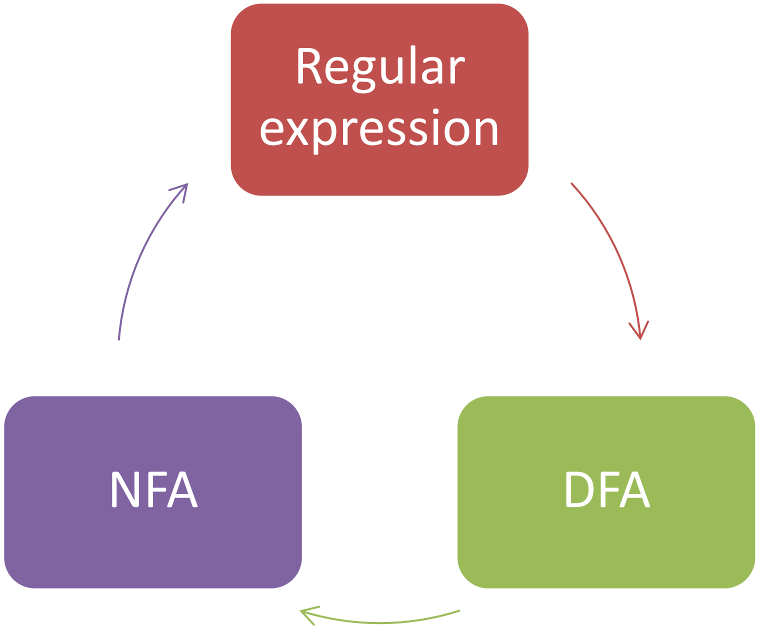
  (1) Convert regular expression R to an equivalent NFA A.

  (2) Run TM N on input ⟨A, w⟩.

  (3) If N accepts, accept; if N rejects, reject."

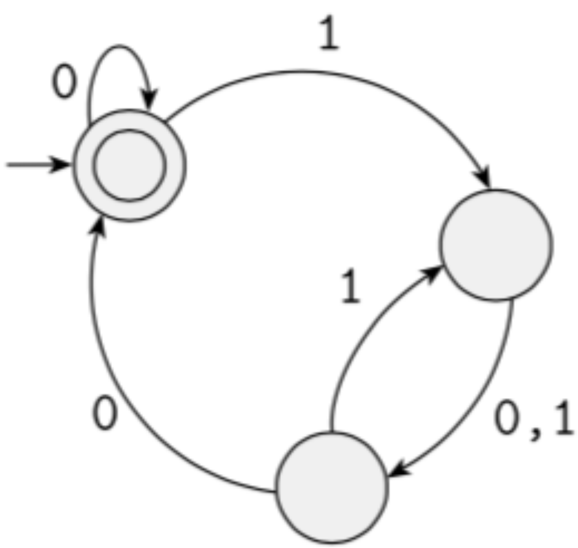
Decidability for DFA/NFA/REX

- For decidability purposes, it is equivalent to present the Turing machine with
  - a DFA
  - a NFA
  - a regular expression



Question: True or False?

- For the following DFA M, $\langle M,\ 0100\rangle \in A_{REX}$



True

4. Emptiness testing for DFA

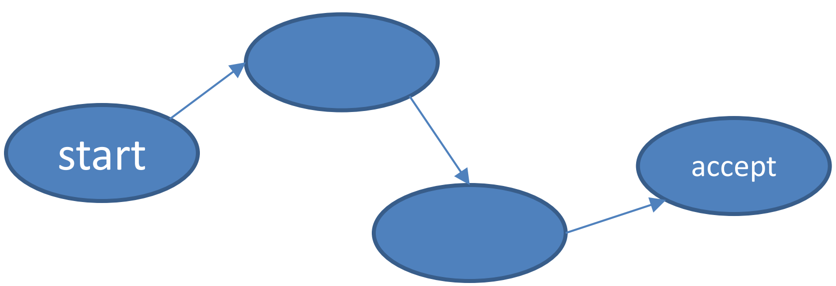
- Emptiness testing for DFA
  - whether or not a DFA does not accept any strings at all (is empty or not)

- Language
  - $E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \varnothing\}$

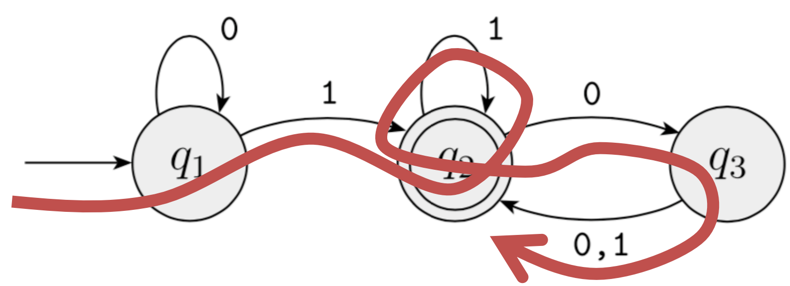
Theorem 4.4

- How to test whether a DFA does not accept any strings?
  - Check all strings => impossible

- Think the opposite: DFA accepts one string
  - DFA accepts some string if and only if reaching an accept state from the start state by traveling along the arrows of the DFA is possible



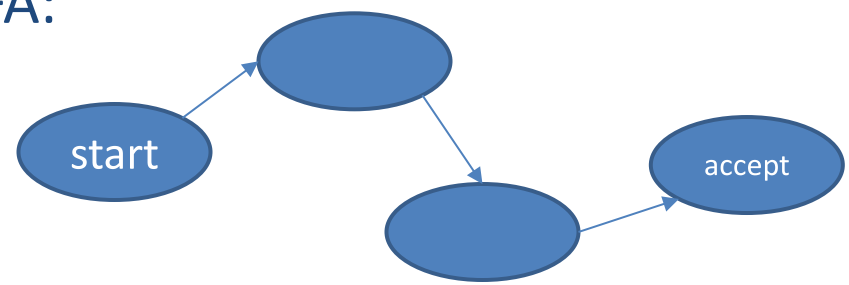
1101



Theorem 4.4 proof details

- Proof idea:

T = "On input ⟨A⟩, where A is a DFA:



(1) Mark the start state of A.

(2) Repeat until no new states get marked:

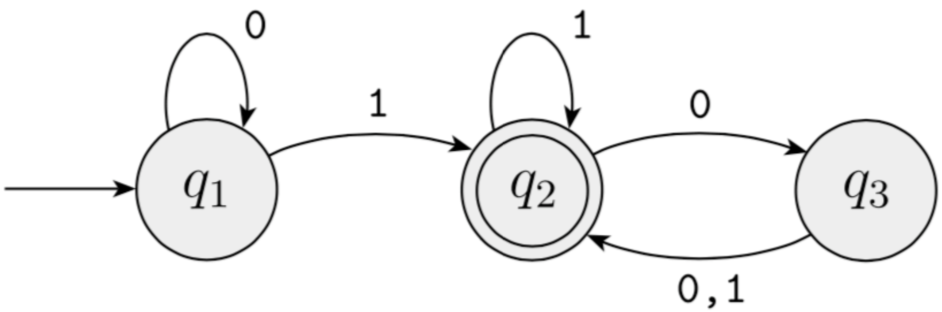
(3) Mark any state that has a transition coming into it from any state that is already marked.

(4) If no accept state is marked, accept; otherwise, reject."

Because the number of states is limited

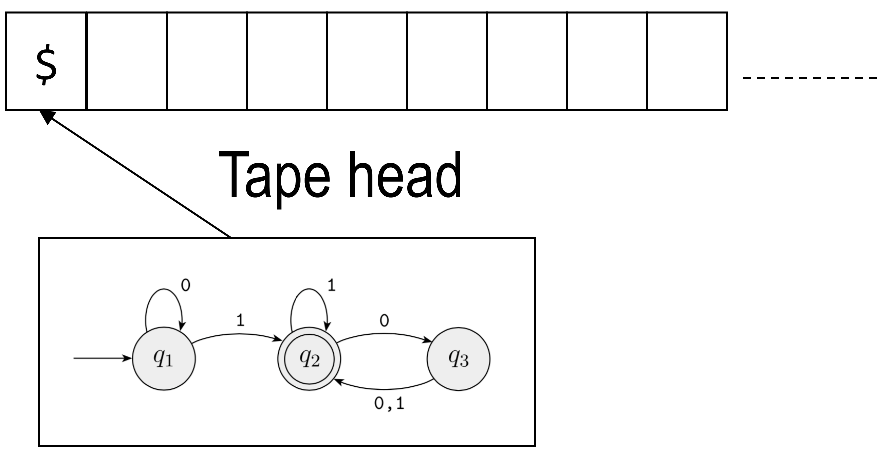
Example

B:



TM:

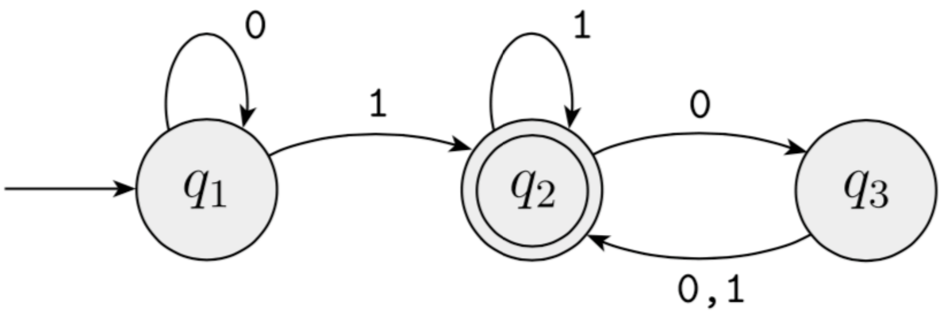
Input



control

Example

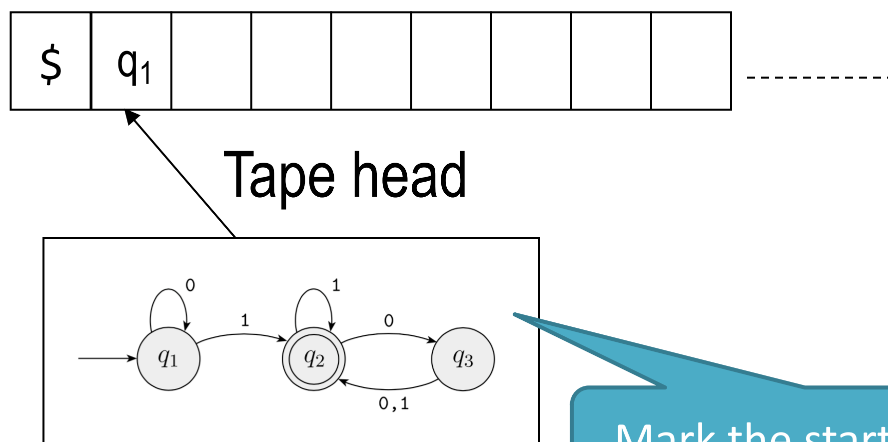
B:



TM:

Input

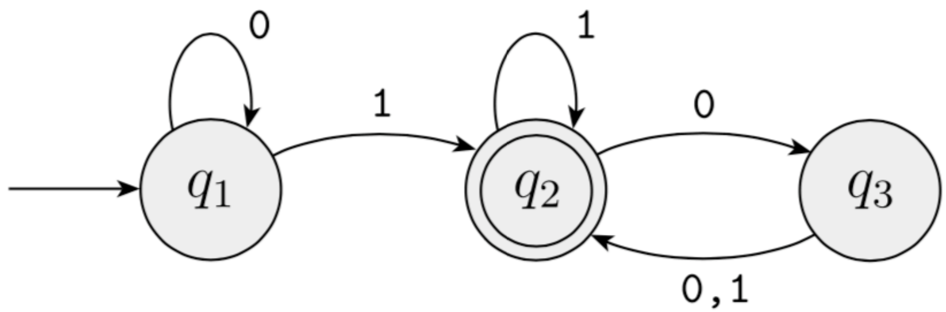
control



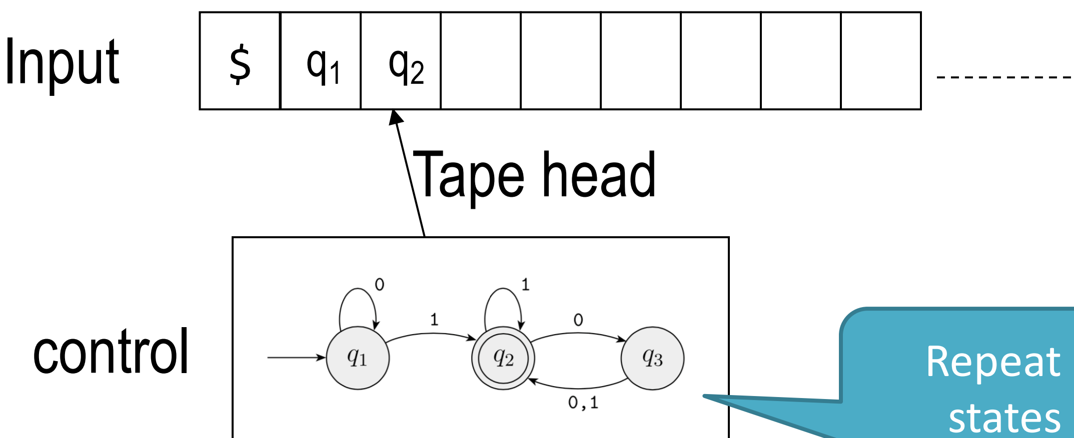
Mark the start state of A

Example

B:



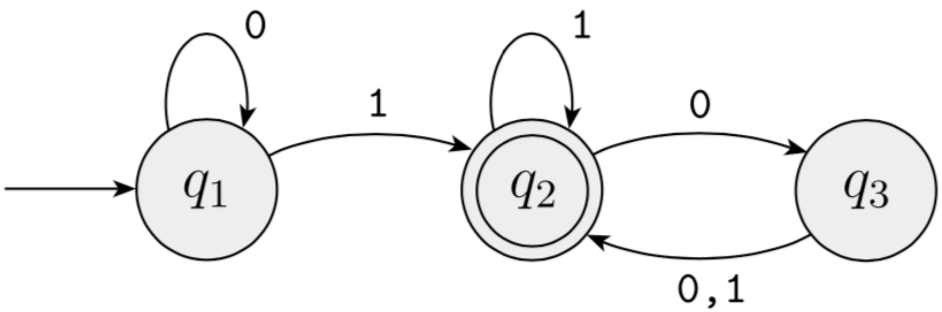
TM:



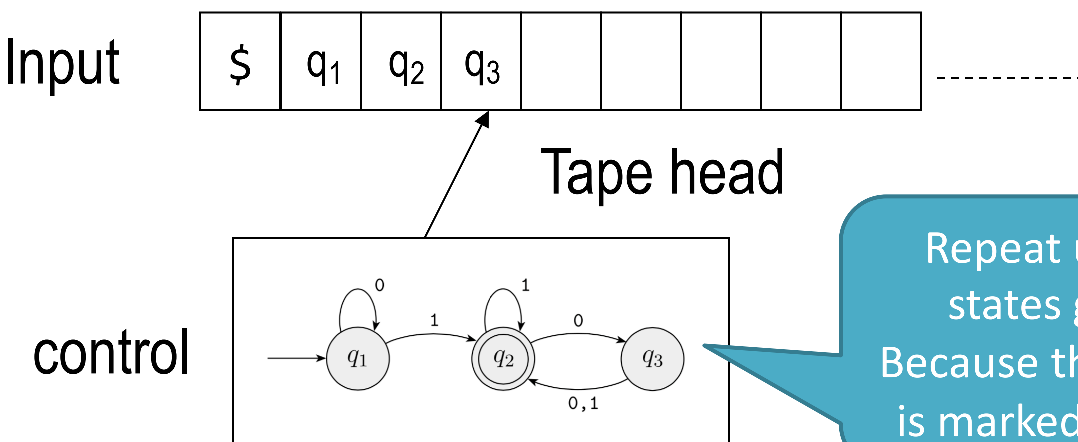
Repeat until no new states get marked

Example

B:



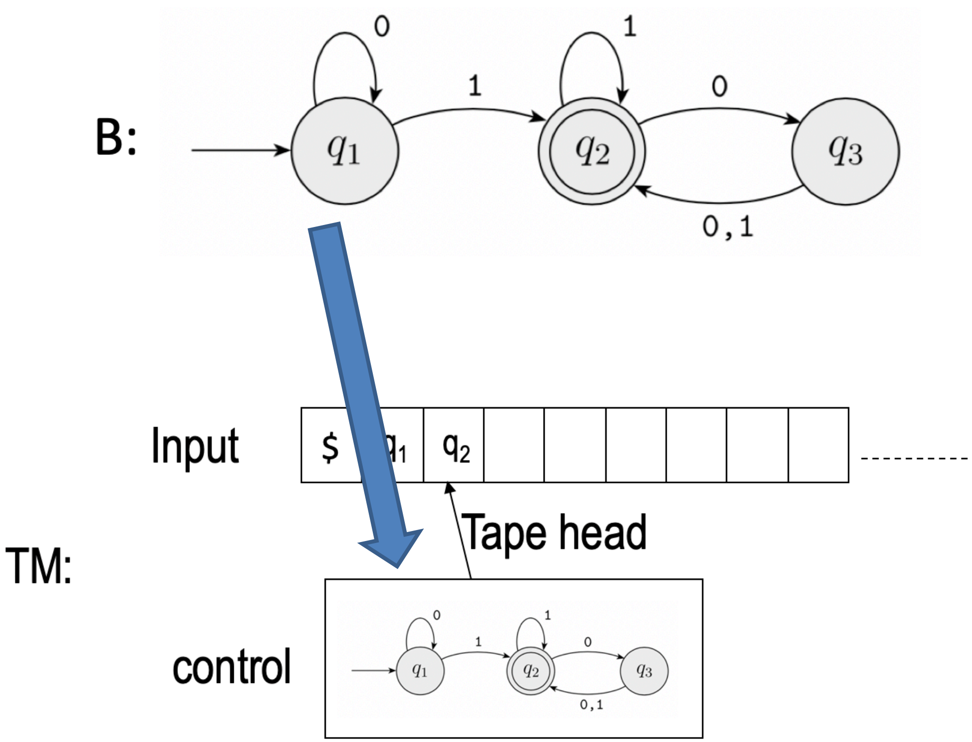
TM:



Repeat until no new states get marked
Because the accept state is marked, so M rejects

4. Emptiness testing for DFAs

- E_{DFA} is a decidable language.
- $E_{DFA} = \{\langle B \rangle \mid B \text{ is a DFA and } L(B) = \emptyset\}$

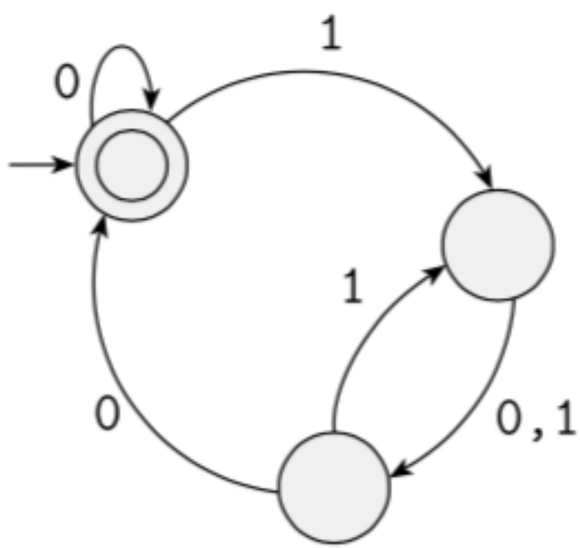


accept state in B is marked
=> M rejects

accept state in B is not marked
=> M accepts

Question: True or False?

- For the following DFA M, $\langle M \rangle \in E_{DFA}$



False

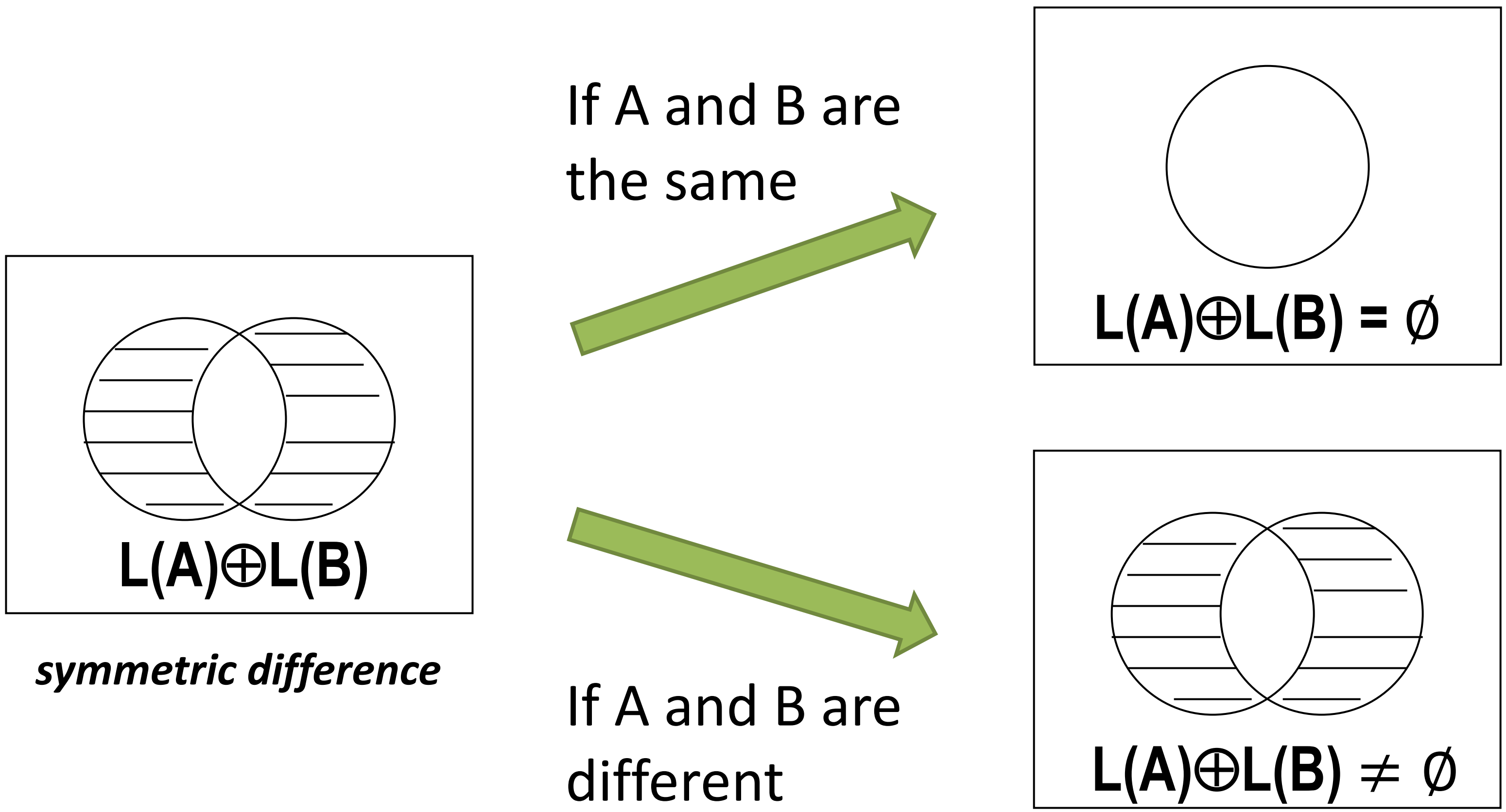
5. Equivalence of DFAs

- Equivalence of DFAs
  - whether two DFAs recognize the same language

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

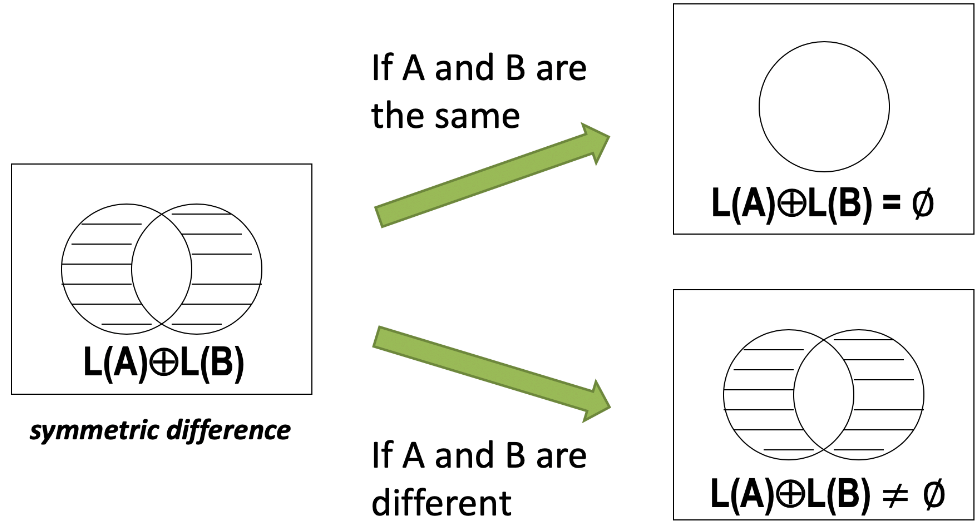
Theorem 4.5

- EQ_{DFA} is a decidable language.



Theorem 4.5

- EQ_{DFA} is a decidable language.



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

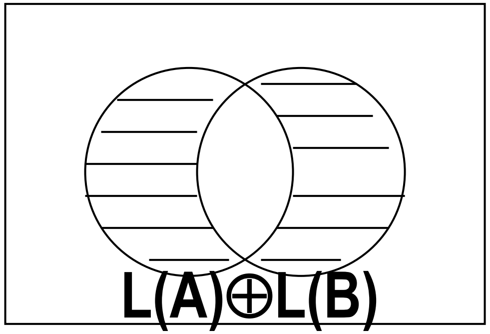
$E_{DFA} = \{\langle L(A)\oplus L(B) \rangle \mid$
$L(A)\oplus L(B)$ is a DFA
and
$L(L(A)\oplus L(B)) = \varnothing\}$

Regular operations

	DFA/NFA	PDA	TM
Union	close	?	?
Concatenation	close	?	?
Star	close	?	?
Complement	close	?	?
Boolean operation	close	?	?

Theorem 4.5

- EQ_{DFA} is a decidable language.



- Proof idea:

Regular language is closure on Boolean computation

Two language is the same if and only if their ***symmetric difference*** is empty

Whether a regular language is empty is decidable (theorem 4.4)

Theorem 4.5 proof

- Proof:

TM F = “On input ⟨A, B⟩,

where A and B are DFAs:

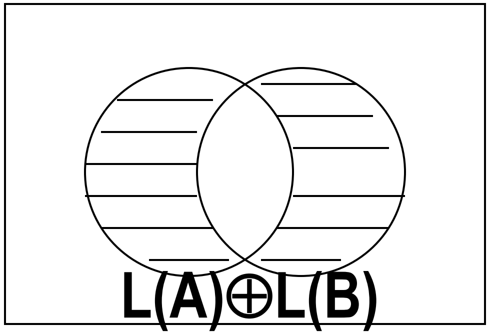
(1) Construct DFA C as:

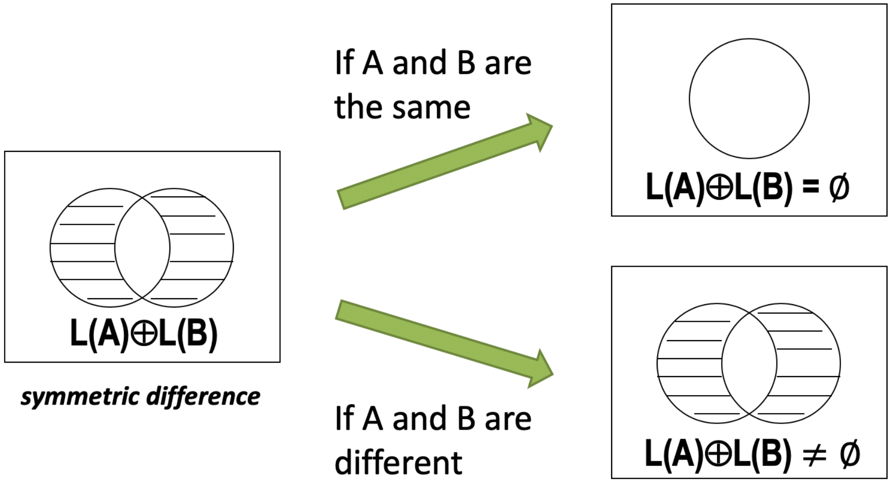
$L(C)=L(A)\oplus L(B)$

(2) Run TM T from Theorem 4.4 on input ⟨C⟩.

(3) If T accepts, accept.

If T rejects, reject.”





Conclusion

- A_{DFA} is a decidable language.
  - Whether a DFA accepts a string
- A_{NFA} is a decidable language.
  - Whether a NFA accepts a string
- A_{REX} is a decidable language.
  - Whether a regular expression generates a string
- E_{DFA} is a decidable language.
  - Whether a DFA is empty
- EQ_{DFA} is a decidable language.
  - Whether two DFAs recognize the same language

Decidable problems concerning CFL/CFGs

1. CFG generation decidability
   - Whether a CFG generates a particular string
2. Emptiness testing for CFGs
   - Whether a CFG is empty
3. Equivalence of CFGs
   - Whether two CFGs recognize the same language
4. CFL decidability
   - Whether a CFL is decidable

1. CFG generation decidability

- CFG generation
  - whether a CFG generates a particular string
- Language
  - $A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$

THEOREM 4.7

- A_{CFG} is a decidable language.

- Proof idea 1:

  use G to go through all derivations to determine whether any is a derivation of w

Grammar G:

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

derivation

$\{ 0^n\#1^n \}$

{
0#1,
00#11,
000#111,
0000#1111,
…
}

accept

000#111

00#111

Never stop because the element number is infinite

THEOREM 4.7

- A_{CFG} is a decidable language.

- Proof idea 1:

  use G to go through all derivations to determine whether any is a derivation of w,

  If G generate w, the algorithm halts

  If G does not generate w, the algorithm ***never halts*** (we might have infinite derivations)

  This only prove A_{CFG} is a turing recognizable language!

THEOREM 4.7

- A_{CFG} is a decidable language.

- Proof idea 2:

  Use Chomsky normal form, any derivation of length n string needs 2n-1 steps **(our homework)**

  Change G into equivalent form in CNF

  Check all derivations which lengths are equal to 2n-1

THEOREM 4.7 proof (details)

- Proof:

TM S = "On input ⟨G, w⟩, 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 $\max\{1, 2n - 1\}$ steps, where n is the length of w;

3. If any of these derivations generate w, accept;

   if not, reject."

Example

Never stop because the element number is infinite

Grammar G:

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

derivation

$\{ 0^n\#1^n \mid n \geq 0 \}$

accept

000#111

00#111

CNF

Grammar G:

$A \rightarrow CV$

$A \rightarrow \#$

$B \rightarrow \#$

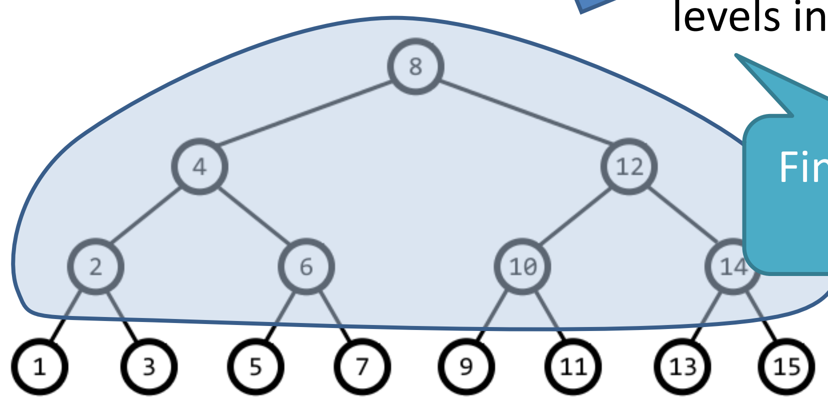
$C \rightarrow UA$

$U \rightarrow 0$

$V \rightarrow 1$

derivation

Search derivation above certain levels in trees



Finite steps

2n-1

Question: True or False?

- For the following CFG G, $\langle G, aa{+}a{*} \rangle \in A_{CFG}$

$S \rightarrow SS+ \mid SS* \mid a$

True

Question: True or False?

- For the following CFG G, $\langle G, abab \rangle \in A_{CFG}$

$S \rightarrow aSb \mid SS \mid \varepsilon$

True

Question: True or False?

- For the following CFG G, $\langle G, aababb \rangle \in A_{CFG}$

$$S \to aSb \mid SS \mid \varepsilon$$

$S \Rightarrow aSb$

.... //follow by $S \Rightarrow abab$

$\Rightarrow aababb$

True

2. Emptiness testing for CFG

- Emptiness testing for CFG
  - whether or not a CFG does not generate any strings (is empty or not)

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

Theorem 4.8

- E_{CFG} is a decidable language.

$S \rightarrow \ldots \rightarrow \ldots \rightarrow 0123$

- Proof idea:

Test whether the start variable can generate a string of terminals

Test each variable whether that variable is capable of generating a string of terminals

Mark the variable if it can generate some string of terminals

Theorem 4.8 proof details

- Proof idea:

R = “On input ⟨G⟩, 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 $A \rightarrow U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots, U_k$ has already been marked.

(4) If the start variable is not marked, accept; otherwise, reject.”

Example

Mark all terminal symbols in G.

(1) Grammar G:

$S \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

Repeat until no new variables get marked

Mark any variable A where G has a rule $A \rightarrow U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots, U_k$ has already been marked.

(2) Grammar G:

$S \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

(3) Grammar G:

$S \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

Start variable is marked, accept.

(4) Grammar G:

$S \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

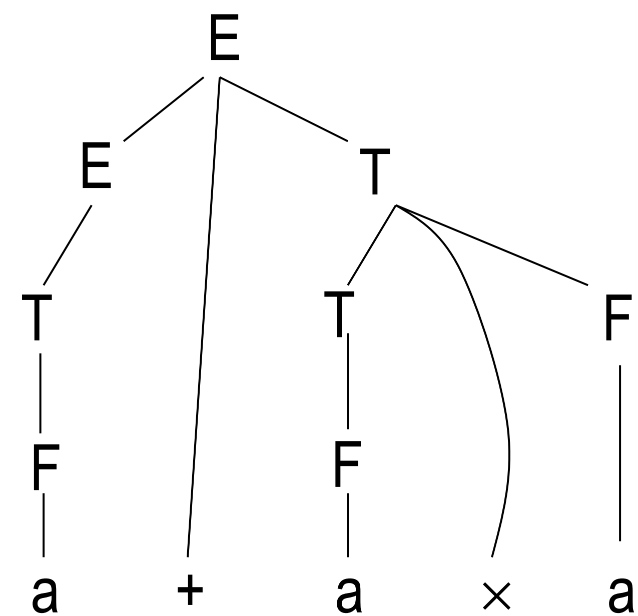
Question: True or False?

- For the following CFG G, $\langle G \rangle \in E_{CFG}$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow (E) \mid a$$



False

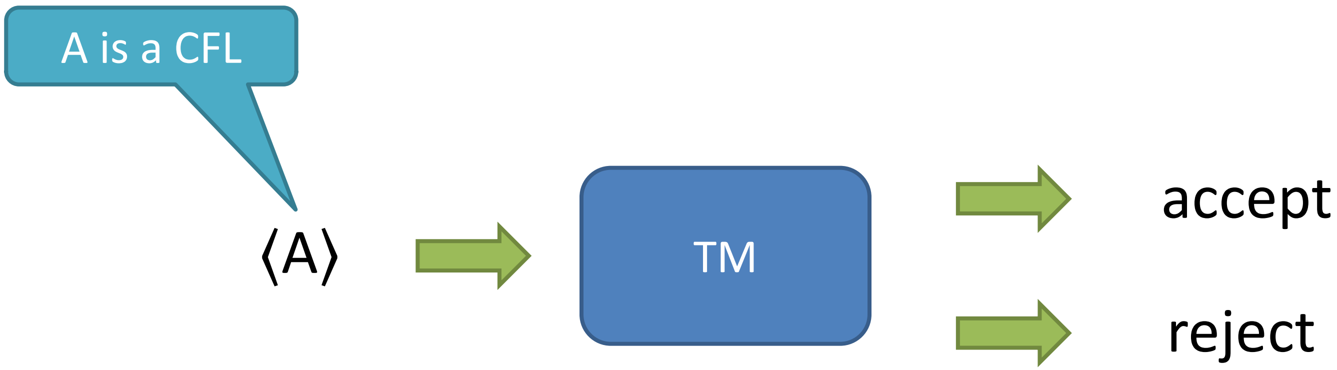
3. Equivalence of CFGs

- Equivalence of CFGs
  - whether two CFGs generate the same language

- Language
  - $EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$
  - EQ_{CFG} is ***not decidable***. Leave it in our homework in future.

4. CFL is a decidable lanagugе

- Every CFL is decidable.



Theorem 4.9

- Every CFL is decidable.

- Proof:

Suppose A is CFL, G is CFG of A. Design a TM M_G that decides A.

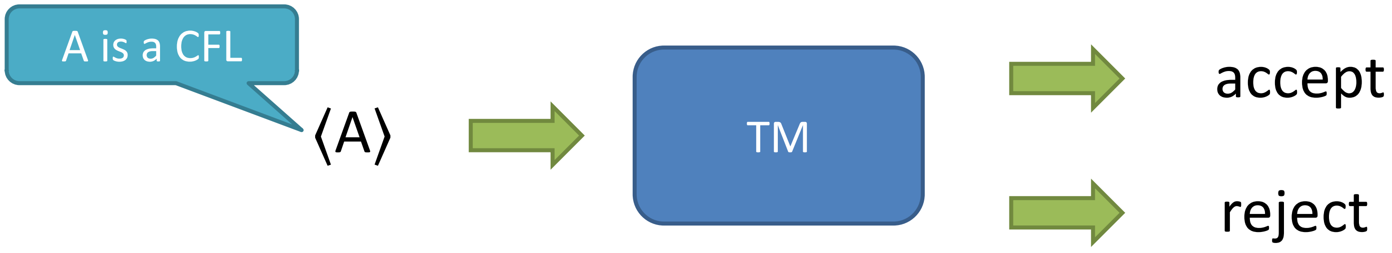
M_G = "On input w:

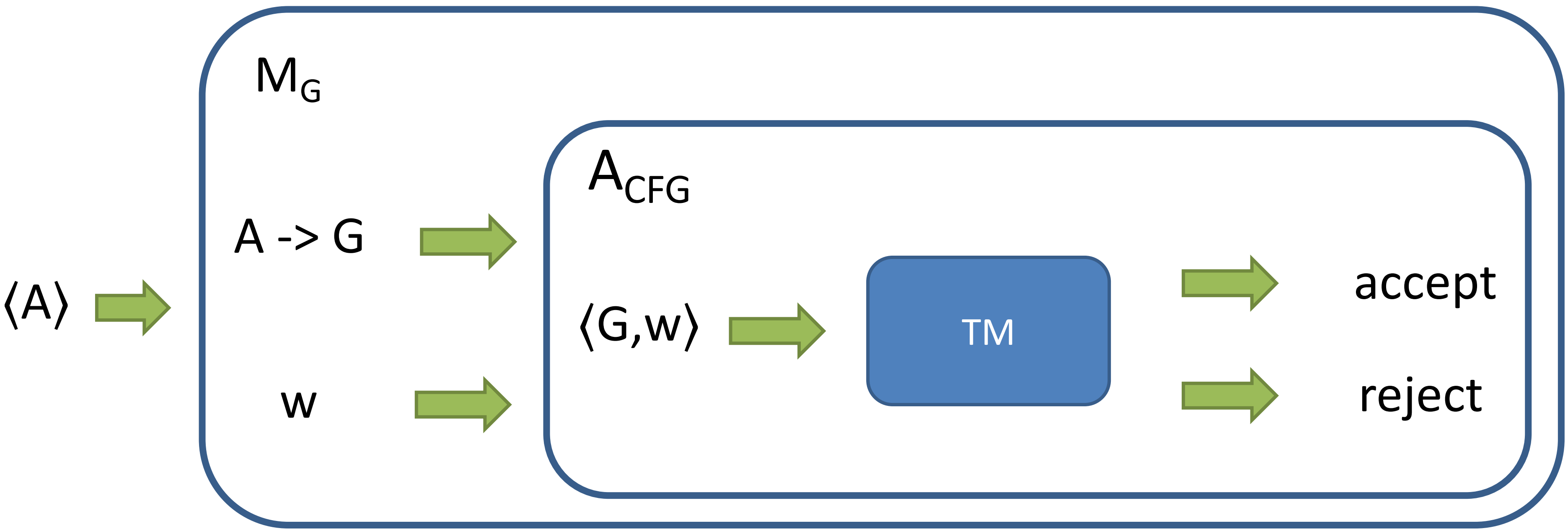
(1) Run TM S on input $\langle G, w \rangle$.

(2) If this machine accepts, accept;

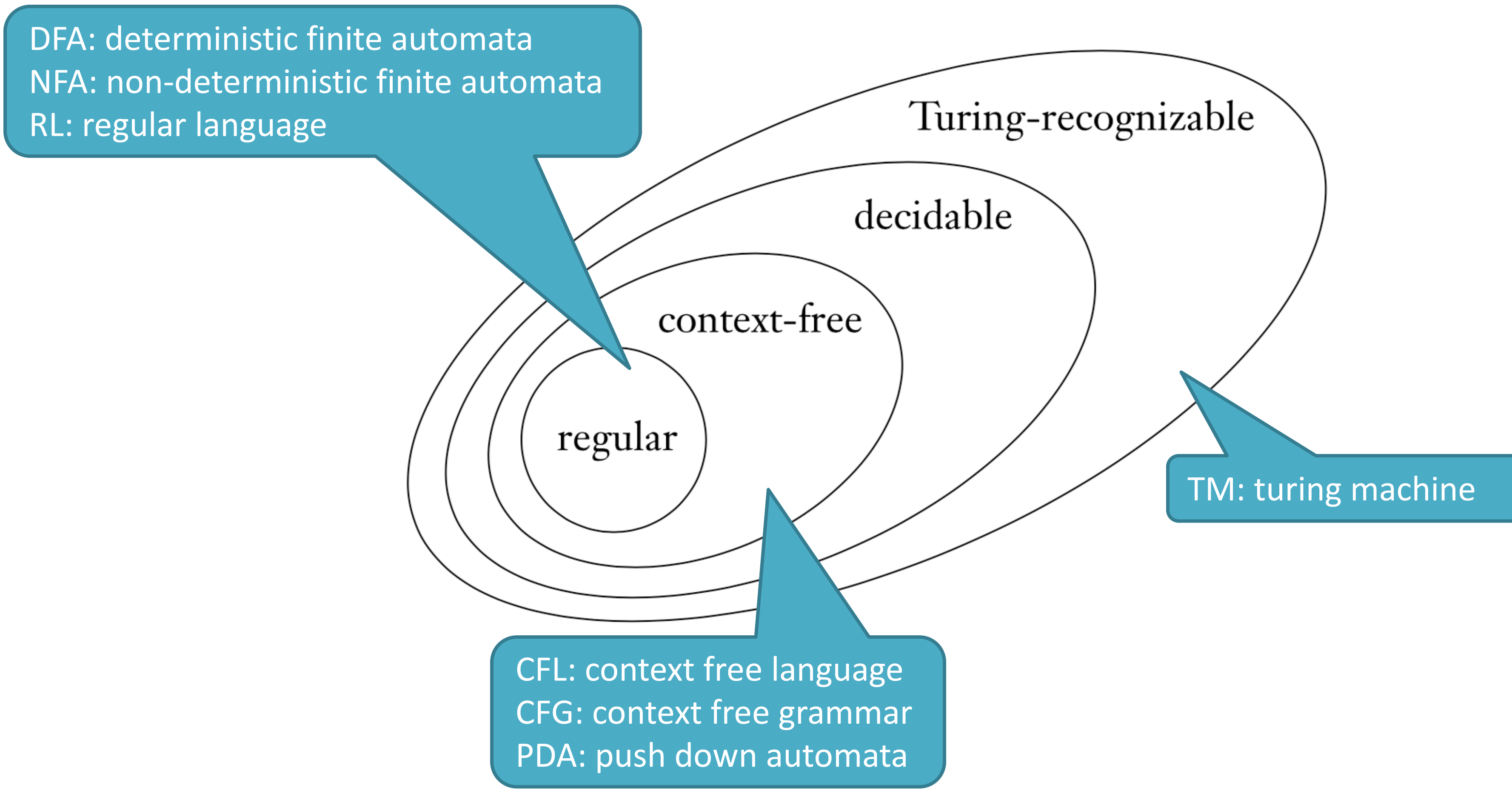
if it rejects, reject."

Theorem 4.9





Theorem 4.9


DFA: deterministic finite automata
NFA: non-deterministic finite automata
RL: regular language
Turing-recognizable
decidable
context-free
regular
TM: turing machine
CFL: context free language
CFG: context free grammar
PDA: push down automata

Operation on languages

	RL: DFA/NFA/RE	CFL: CFG/PDA	TM-decidable
Union	close	close	close
Concatenation	close	close	close
Intersection	close	not close	close
Star	close	close	close
Complement	close	not close	close
Boolean operation	close	/	close

Conclusion

- A_{CFG} is a decidable language.
  - Whether a CFG generates a particular string
- E_{CFG} is a decidable language.
  - Whether a CFG is empty
- EQ_{CFG} is not a decidable language.
  - Whether two CFGs recognize the same language
- CFL is a decidable language.
  - Whether a CFL is decidable