Introduction to the Theory of Computation

Set 8 — Turing Machines / Decidability

What Is an Algorithm?

Intuitively, an algorithm is anything that can be simulated by a Turing machine (Church-Turing Thesis)

- Many algorithms can be simulated by Turing machines
- Inputs can be represented as strings
  - Graphs
  - Polynomials
  - Automata
  - Etc.

Example Algorithm

Depth-first walk-through of binary tree

Which nodes do you visit, and in what order, when doing a depth-first search?

- Visit each leaf node from left to right
- Recursive algorithm
- Stop after rightmost leaf node has been visited

Binary Tree Depth-First Walkthrough

Start at root

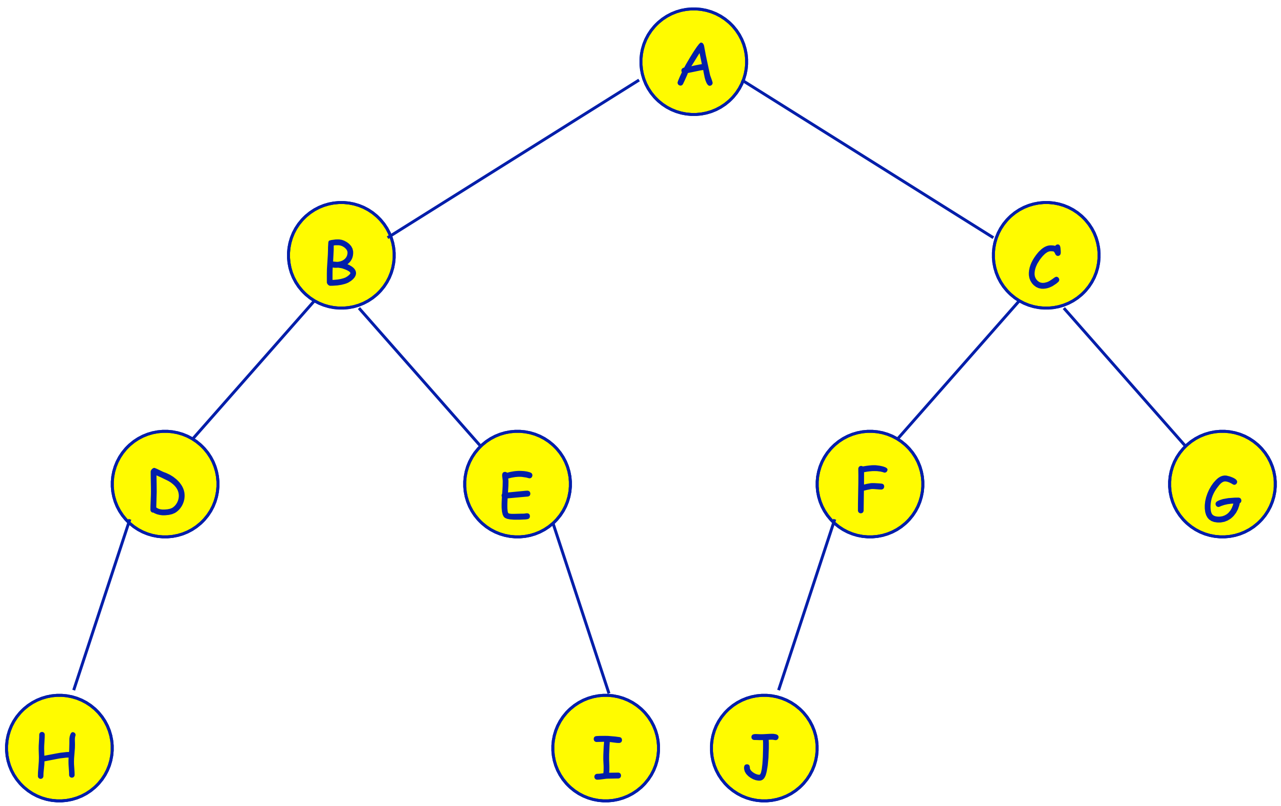
Process left subtree (if one exists)

Process right subtree (if one exists)

Process how?

- *Print* the node name
- If there is a left subtree then
  - Process the left subtree
  - *Print* the node name again
- If there is a right subtree then
  - Process the right subtree
  - *Print* the node name again

Example



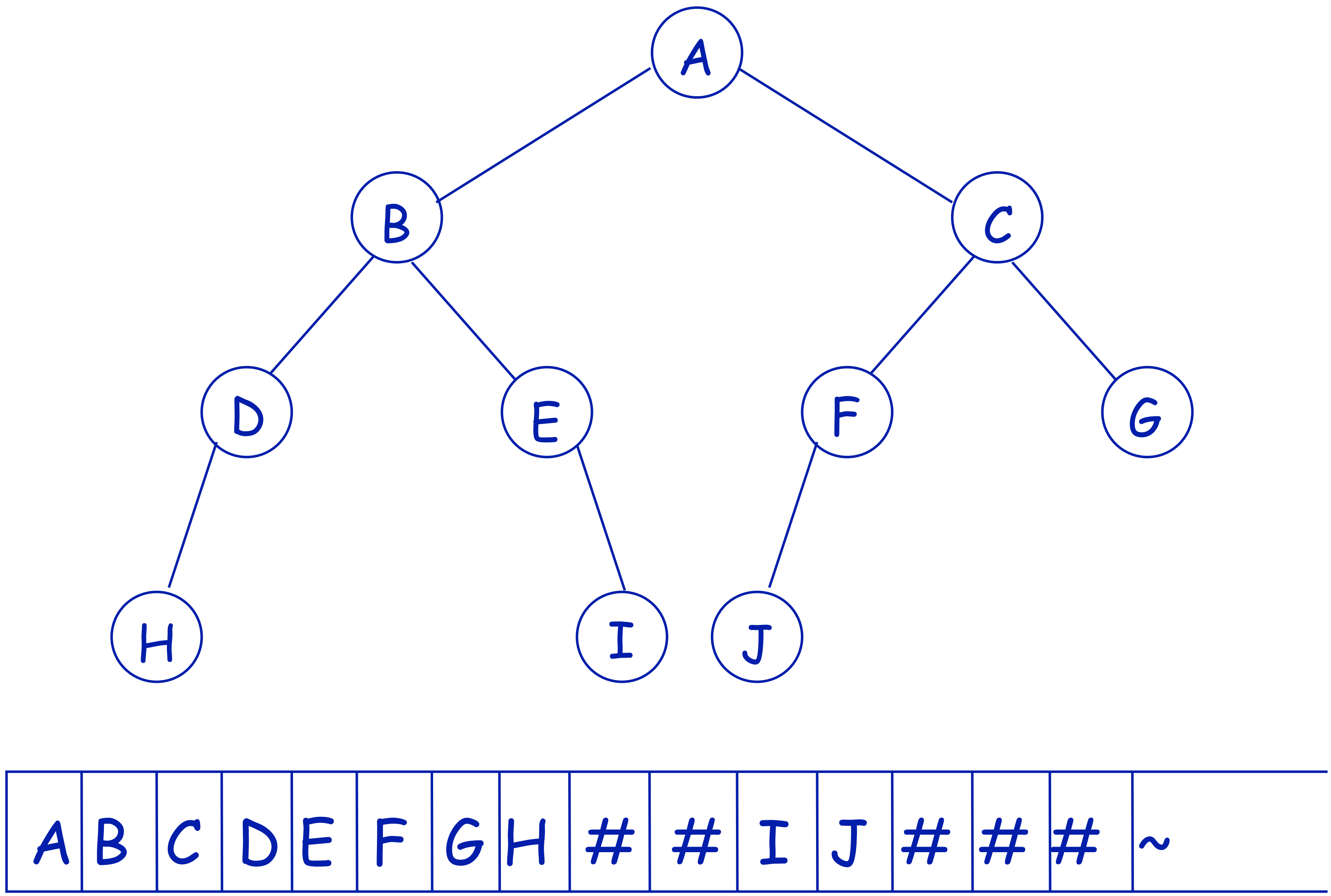
A B D H D B E I E B A C F J F C G

Can a Turing Machine Do This?

Input must be a string (not a tree)

- Can we represent a tree with a string?
- Yes.

String representation of a binary tree



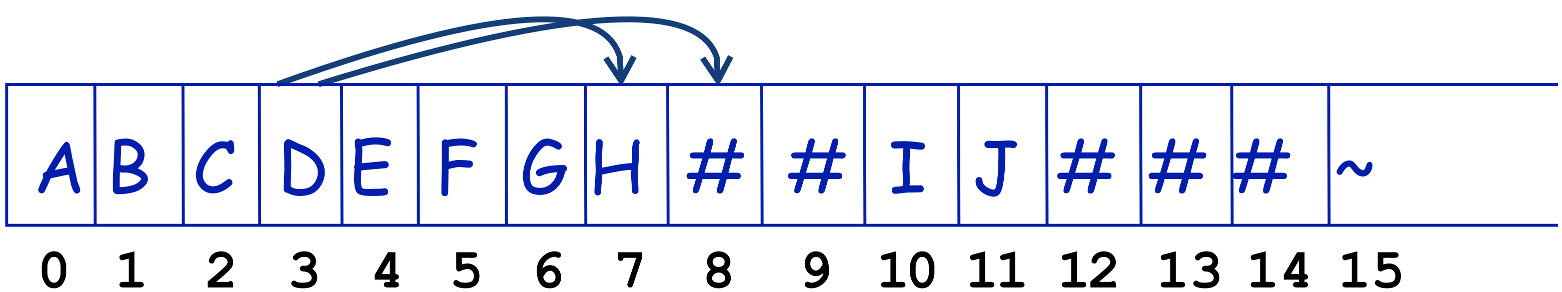
Can a Turing Machine Do This?

Input must be a string (not a tree)

- Can we represent a tree with a string?
- Yes

How do we know which node(s) are children of the current node?

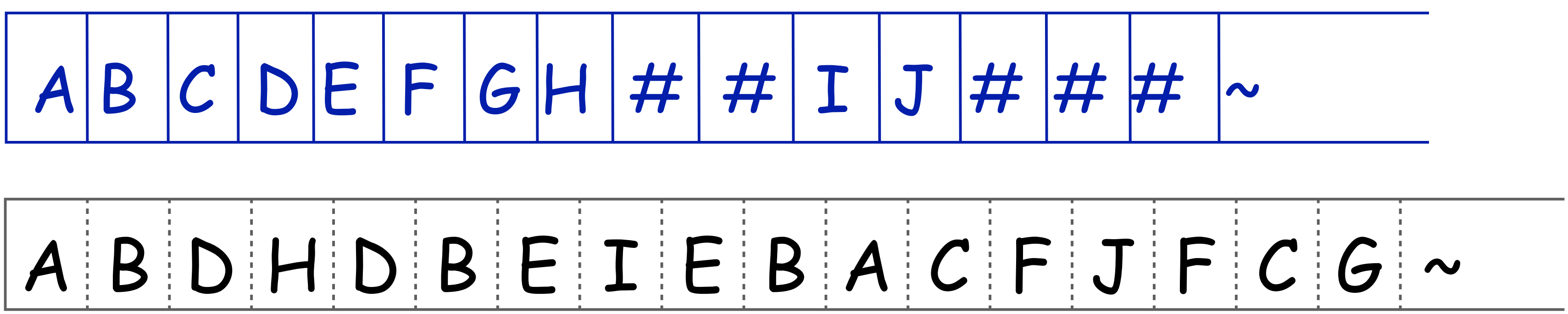
- The root node is at index 0.
- The children of node at index n are at indices 2n+1 and 2n+2



What About the Output?

Need to write out nodes in a particular order

- Can we do this with a TM?
- Yes. Add output tape
- A TM can move left and right on the input tape writing to the output tape whenever appropriate



Describing Turing Machines

From now on, we can describe Turing machines algorithmically

M = "On input w

1. …

2. …

…"

Decidability

A language is decidable if some Turing machine decides it

- **Every string in Σ^* is either accepted or rejected**

Not all languages are decidable

- **Not all languages can be decided by a Turing machine**
- **We will see examples of both decidable and undecidable languages**

Showing a Language Is Decidable

Write a decider that decides it

Must show the decider

- Halts on all inputs
- Accepts w ⇔ w is in the language

Can use algorithmic description

DFA Acceptance Problem

Consider the language

A_{DFA} = {<B,w> | B is a DFA that accepts the string w}

Theorem: A_{DFA} is a decidable language

Proof: Consider the following TM, M

M = "On input string <B,w>, where B is a DFA and w is an input to B

1. Simulate B on input w
2. If simulation ends in accept state, accept. Otherwise, reject."

NFA Acceptance Problem

Consider the language

A_{NFA} = {<B,w> | B is a NFA that accepts the string w}

Theorem: A_{NFA} is a decidable language

Proof: Consider the following TM, N

N = "On input string <B,w>

1. Convert B to a DFA C
2. Run TM M shown previously on <C,w>
3. If M accepts, accept. Otherwise, reject."

RE Acceptance Problem

Consider the language

A_{REX} = {<R,w> | R is an RE that generates the string w}

Theorem: A_{REX} **is a decidable language**

Proof: **Consider the following TM, P**

P = "On input string <R,w>

1. Convert R to a DFA C *(using algorithms discussed in class and in texts)*
2. Run TM M shown previously on <C,w>
3. If M accepts, **accept**. Otherwise, **reject**."

Some Decidable Languages

A_{DFA} = {<B,w> | B is a DFA that accepts input string w}

A_{NFA} = {<B,w> | B is an NFA that accepts input string w}

A_{REX} = {<R,w> | R is a regular expression that generates string w}

Emptiness Testing Problem

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

Theorem: E_{DFA} **is a decidable language**

Proof: **Consider the following TM, T**

T = "On input string <A>, where A is a DFA

1. **Mark the start state**
2. **Repeat until no new states get marked**
   - Mark any state that has a transition coming into it from any state already marked
3. **If *no* accept states are marked, accept. Otherwise, reject."**

DFA Equivalence Problem

EQ_{DFA} = {<A,B> | A and B are DFA's and L(A) = L(B)}

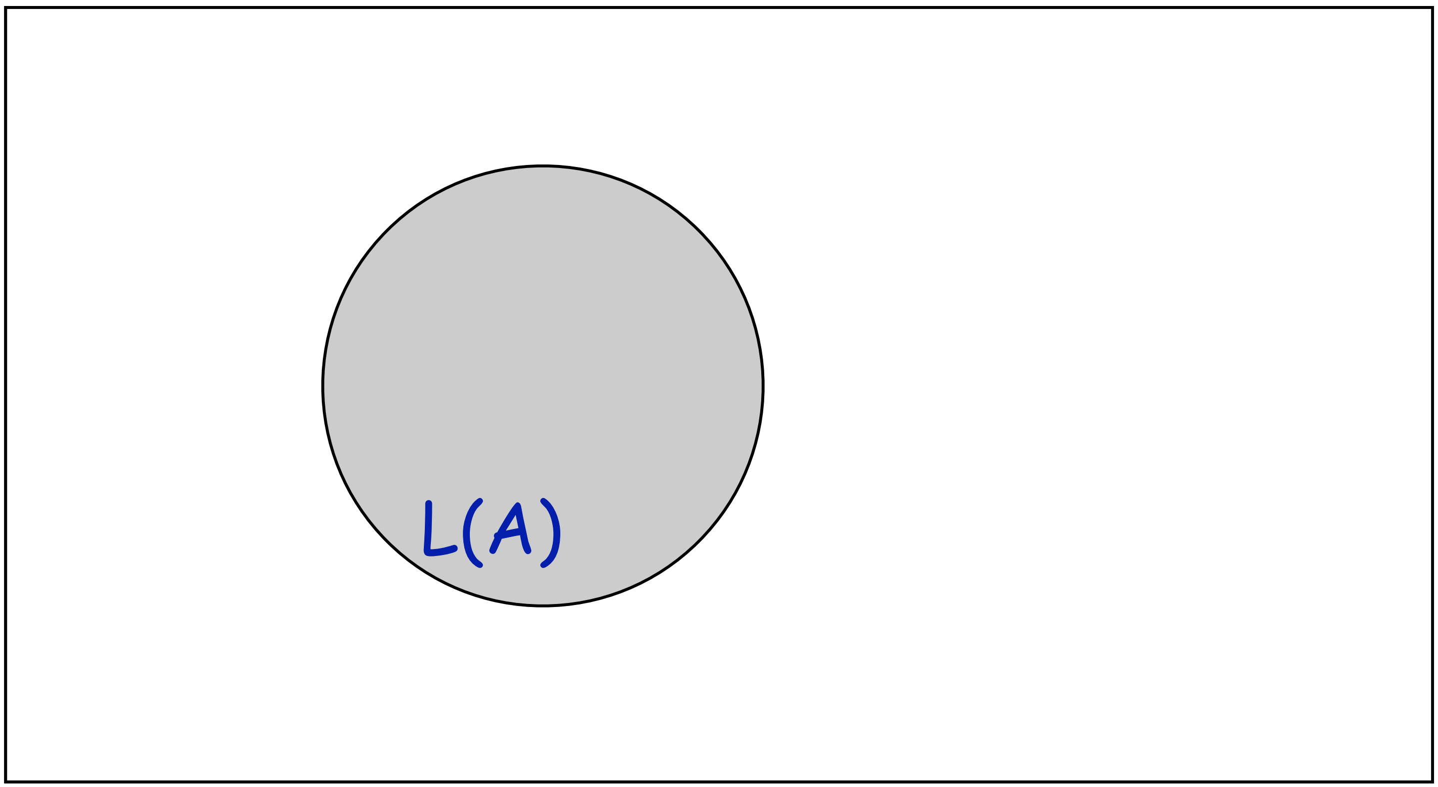
Theorem: EQ_{DFA} is a decidable language

Proof: Consider the following language

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

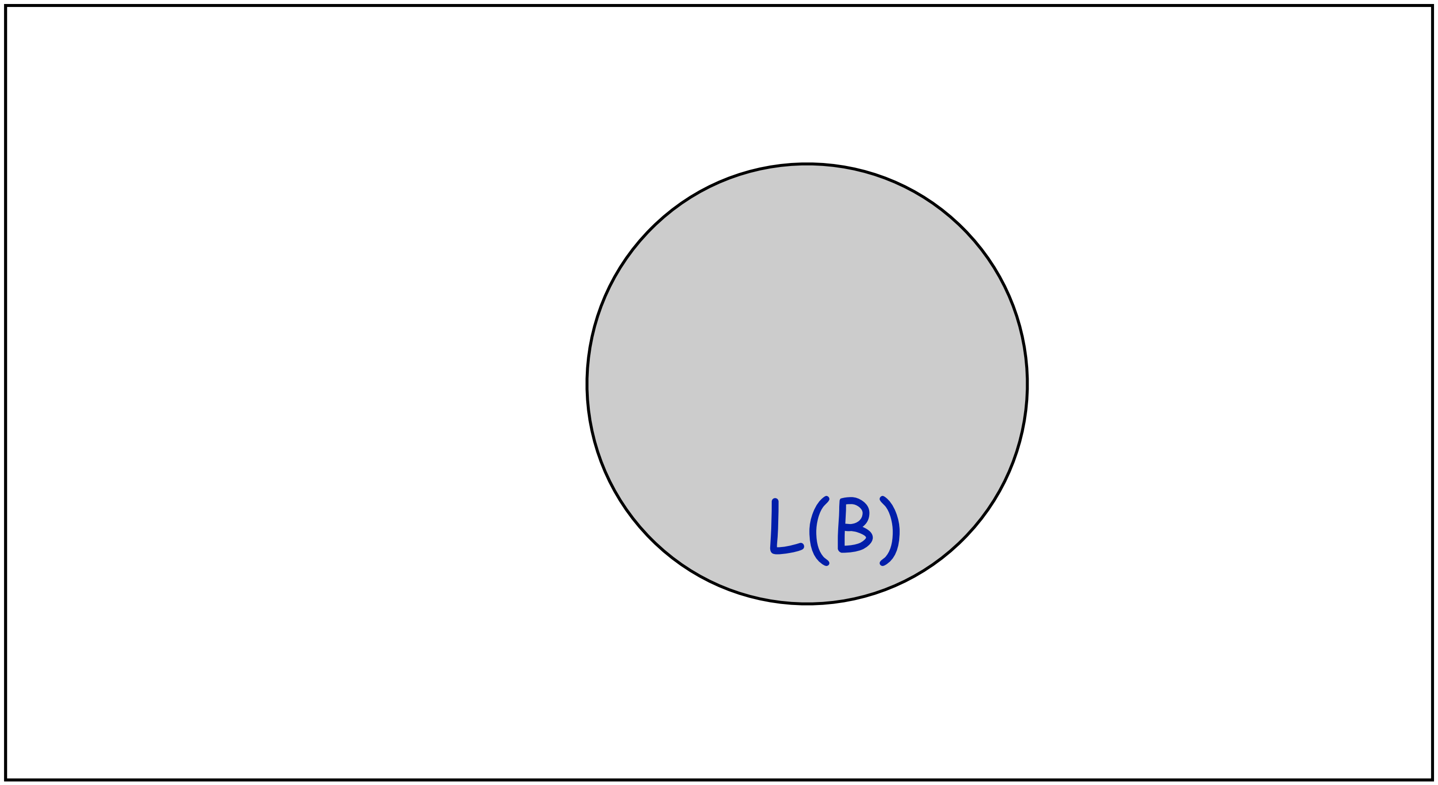
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



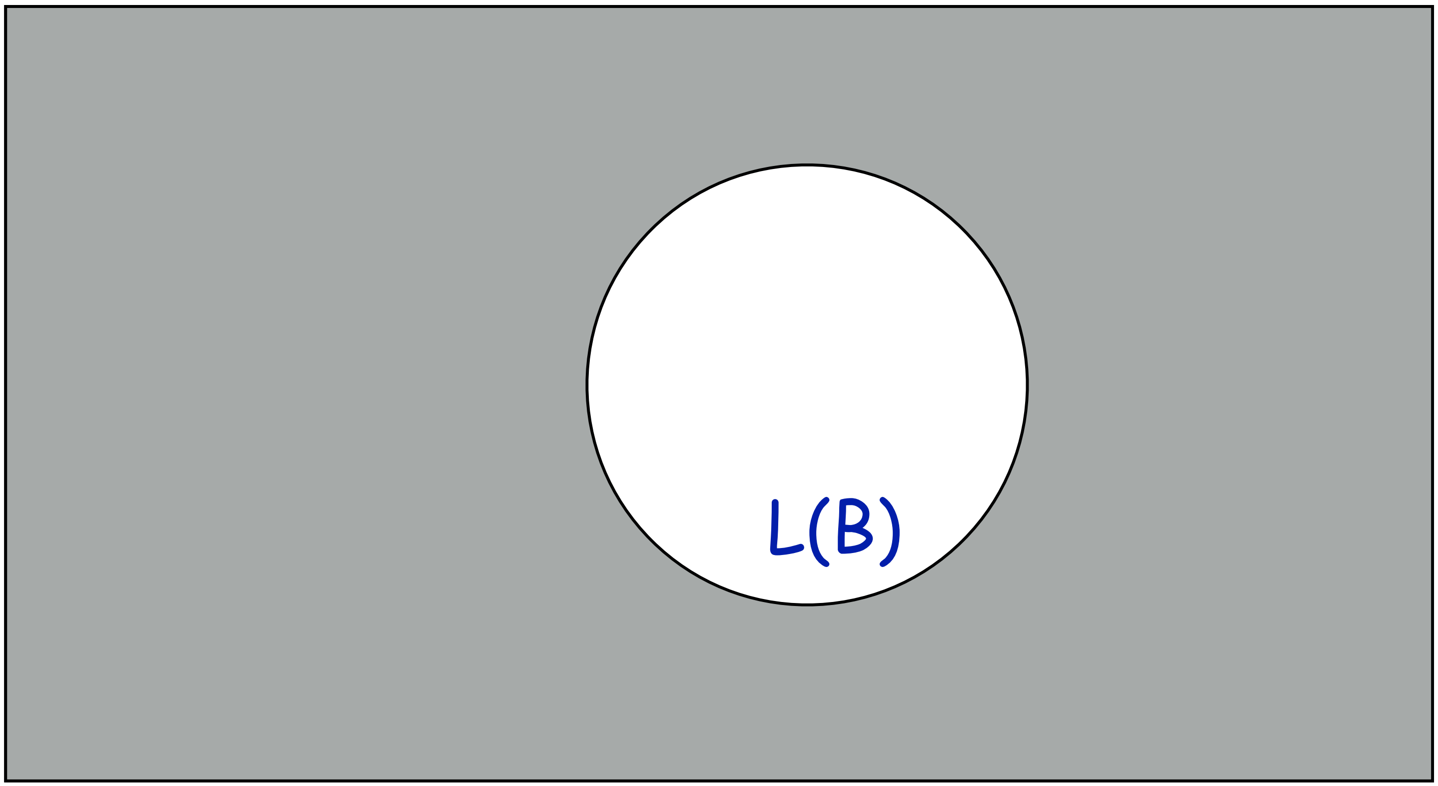
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



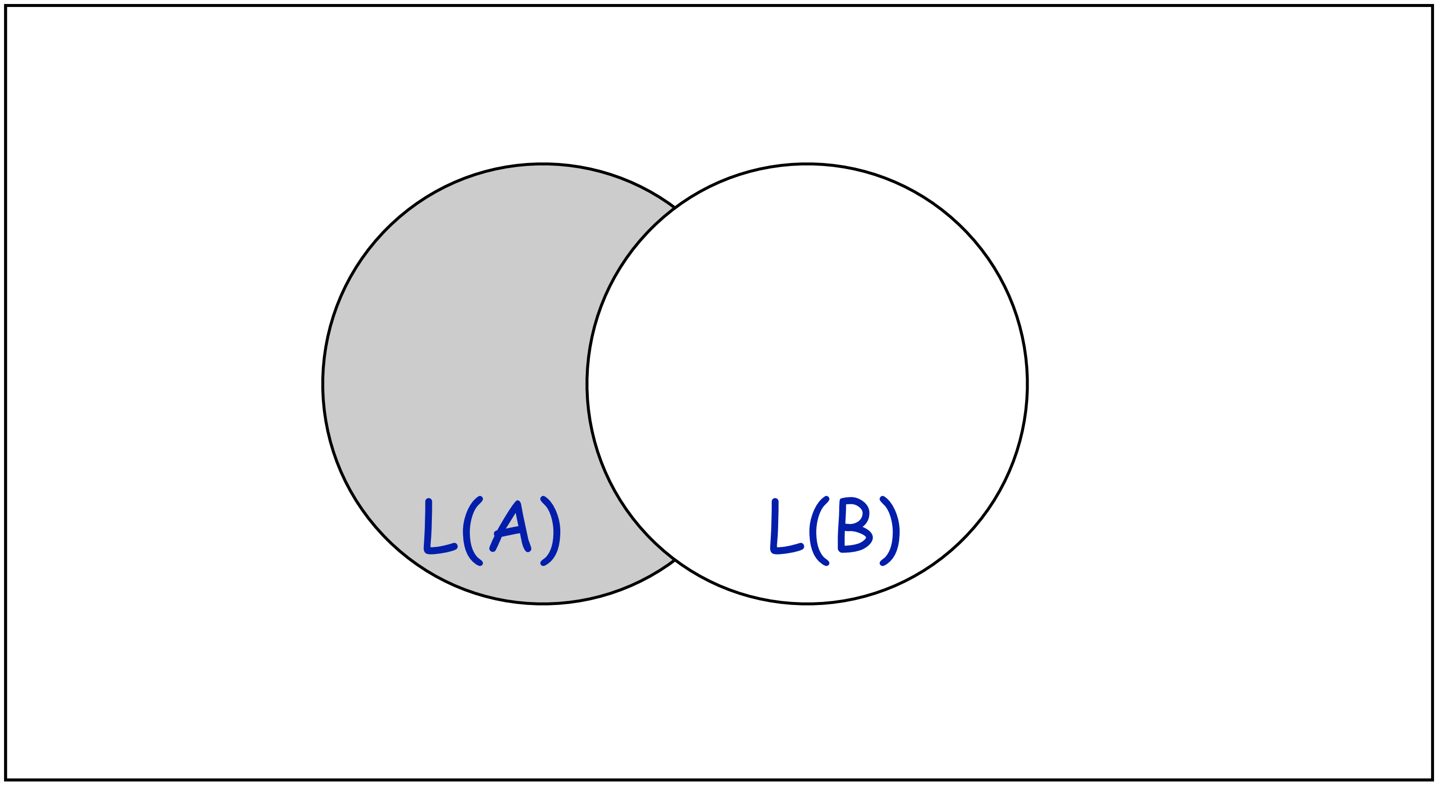
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



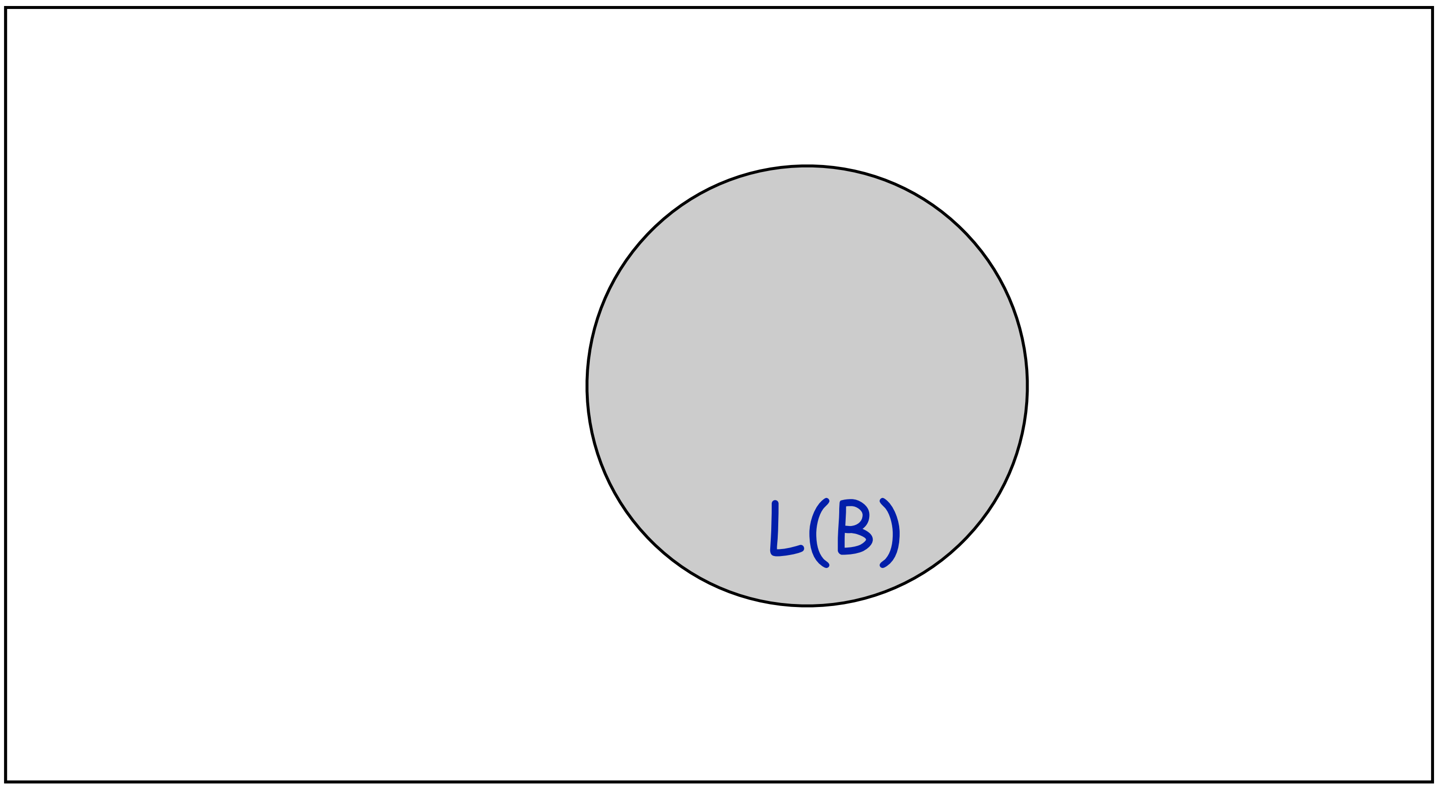
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



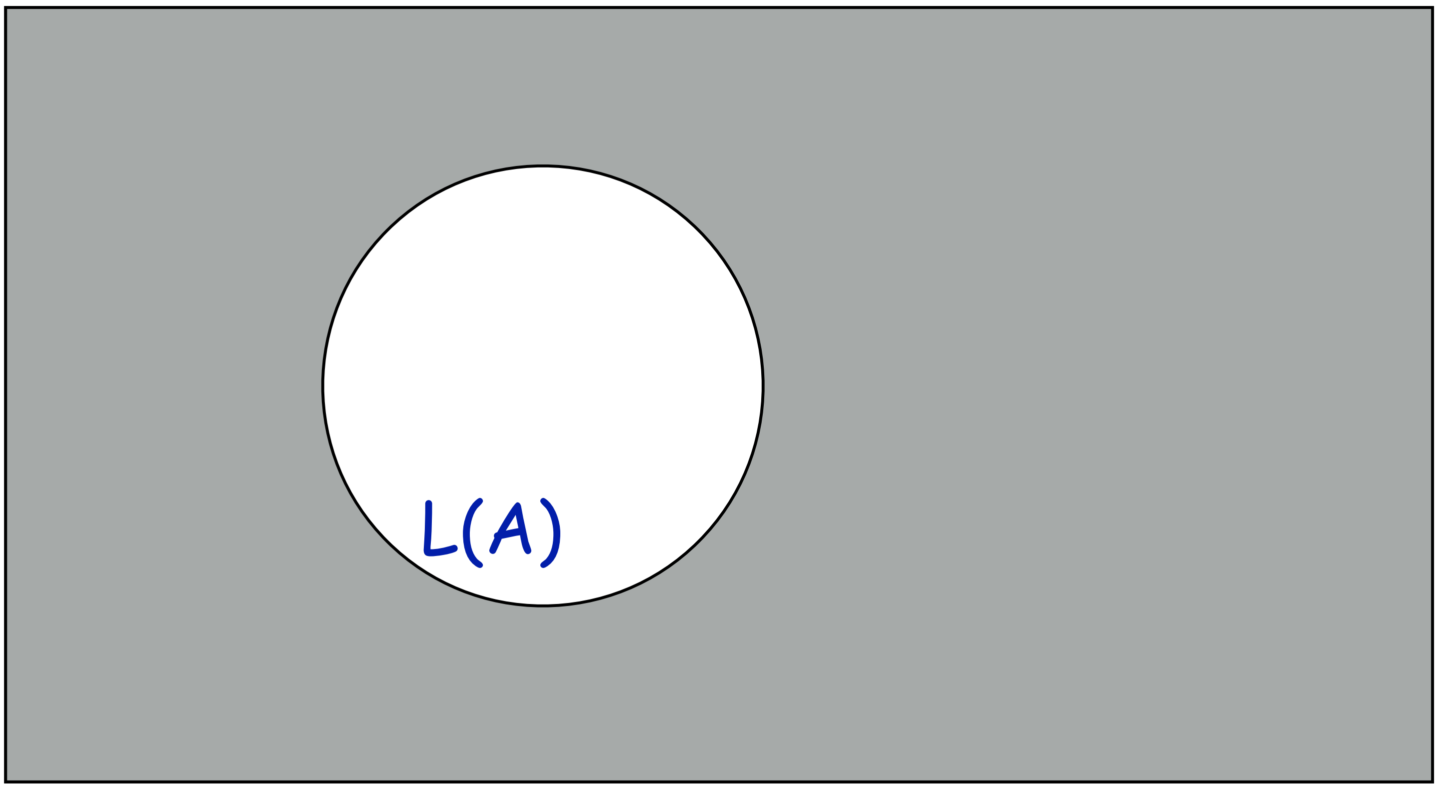
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



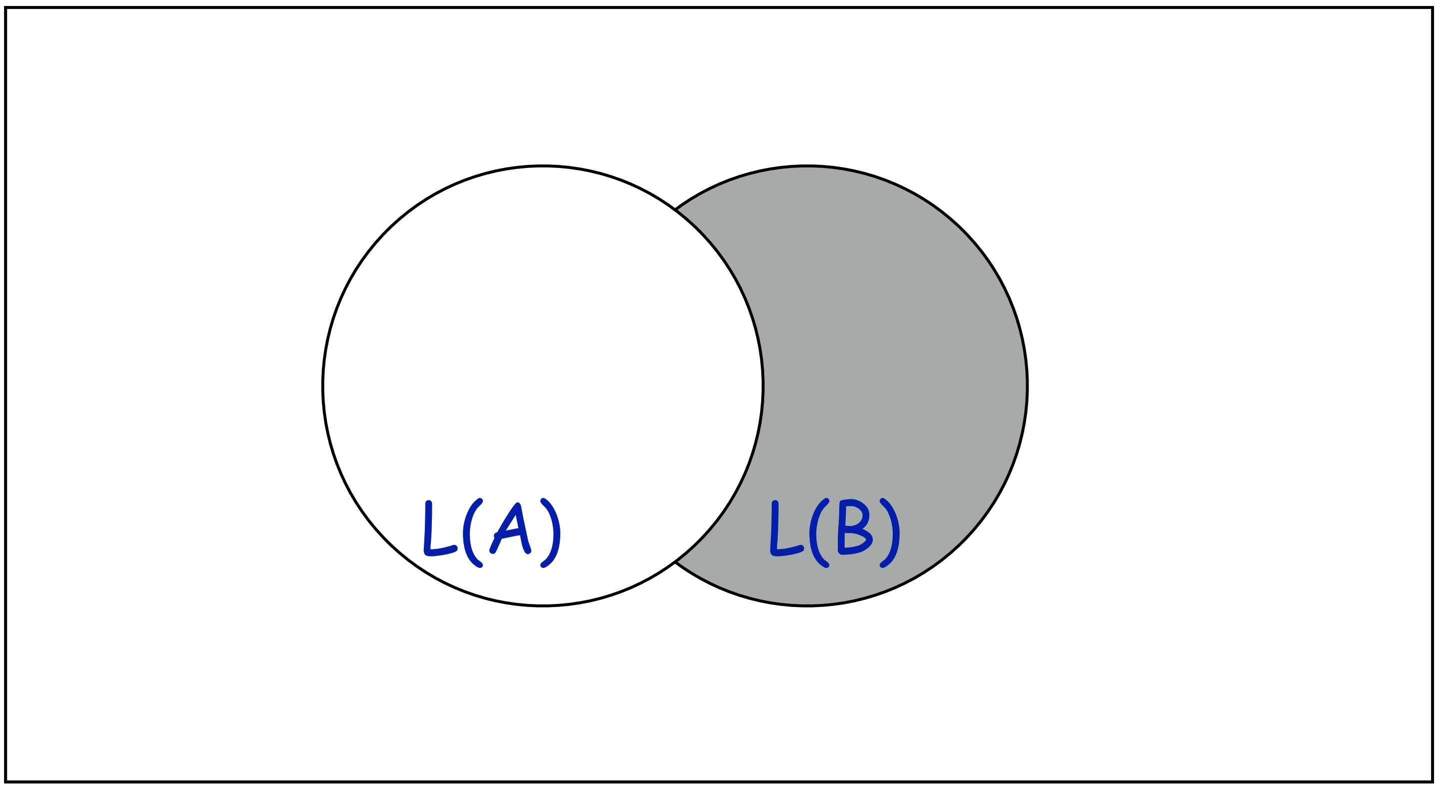
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



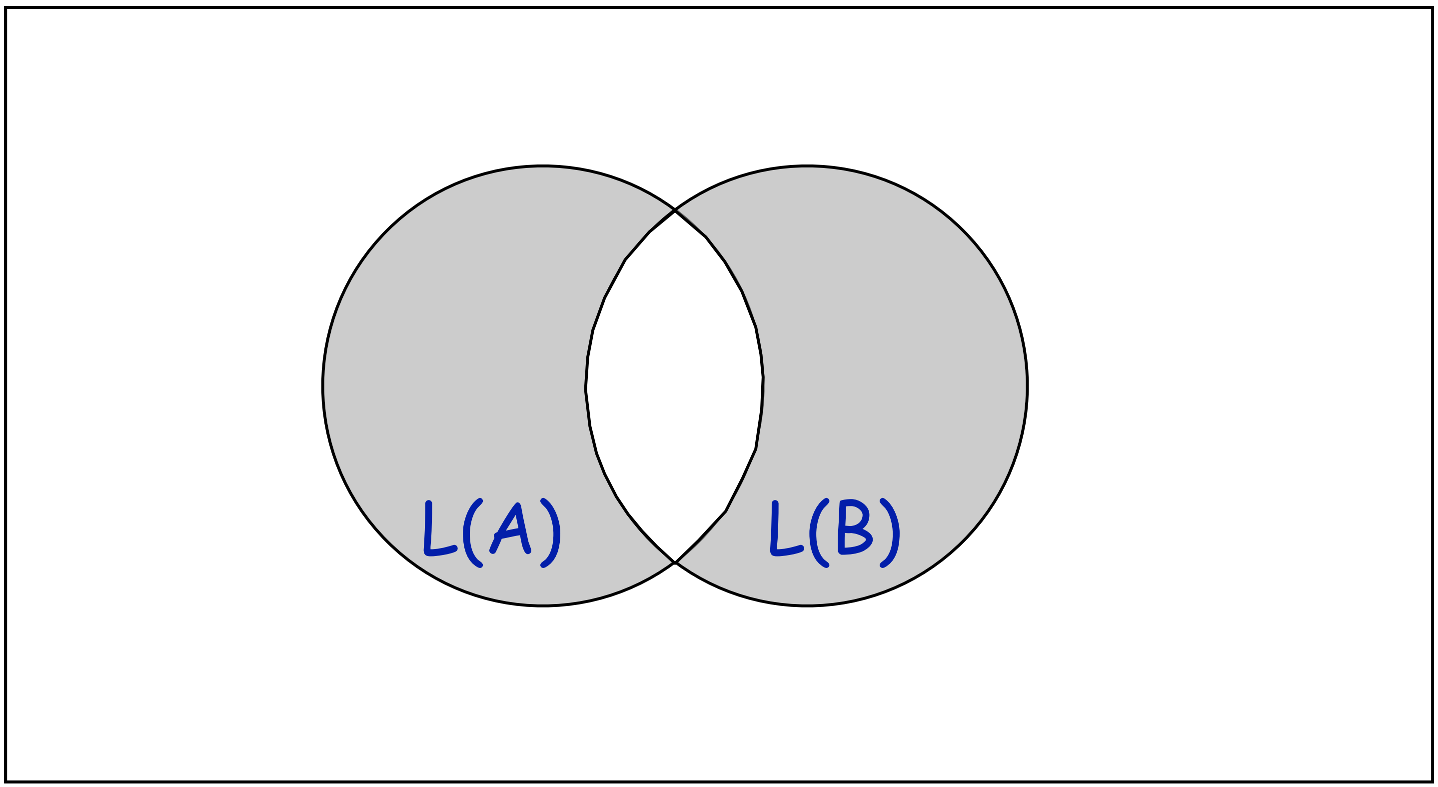
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



DFA Equivalence Problem

EQ_{DFA} = {<A,B> | A and B are DFA's and $L(A) = L(B)$}

Theorem: EQ_{DFA} is a decidable language

Proof: Consider DFA C that accepts

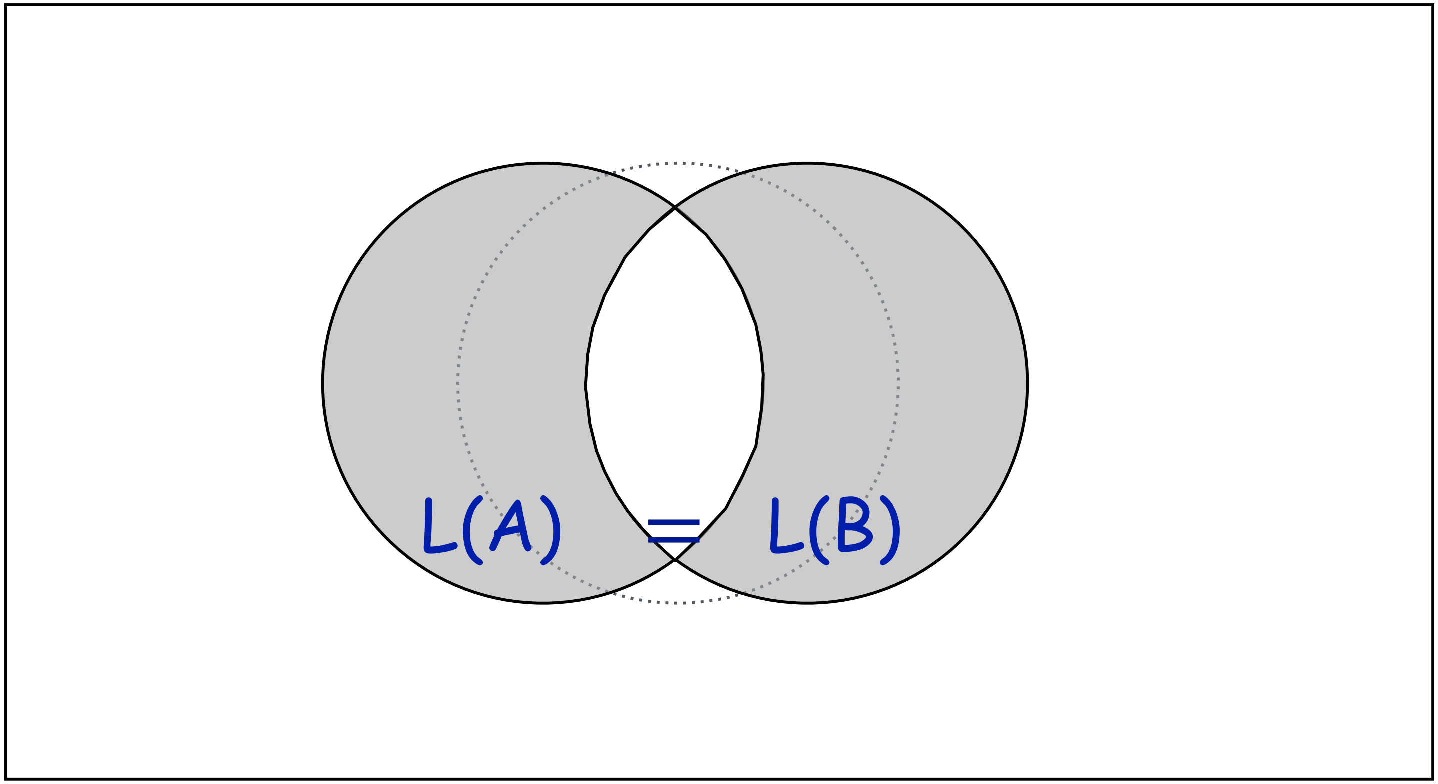
$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

How do we know such a DFA exists?

If $L(C) = \emptyset$, then $L(A) = L(B)$

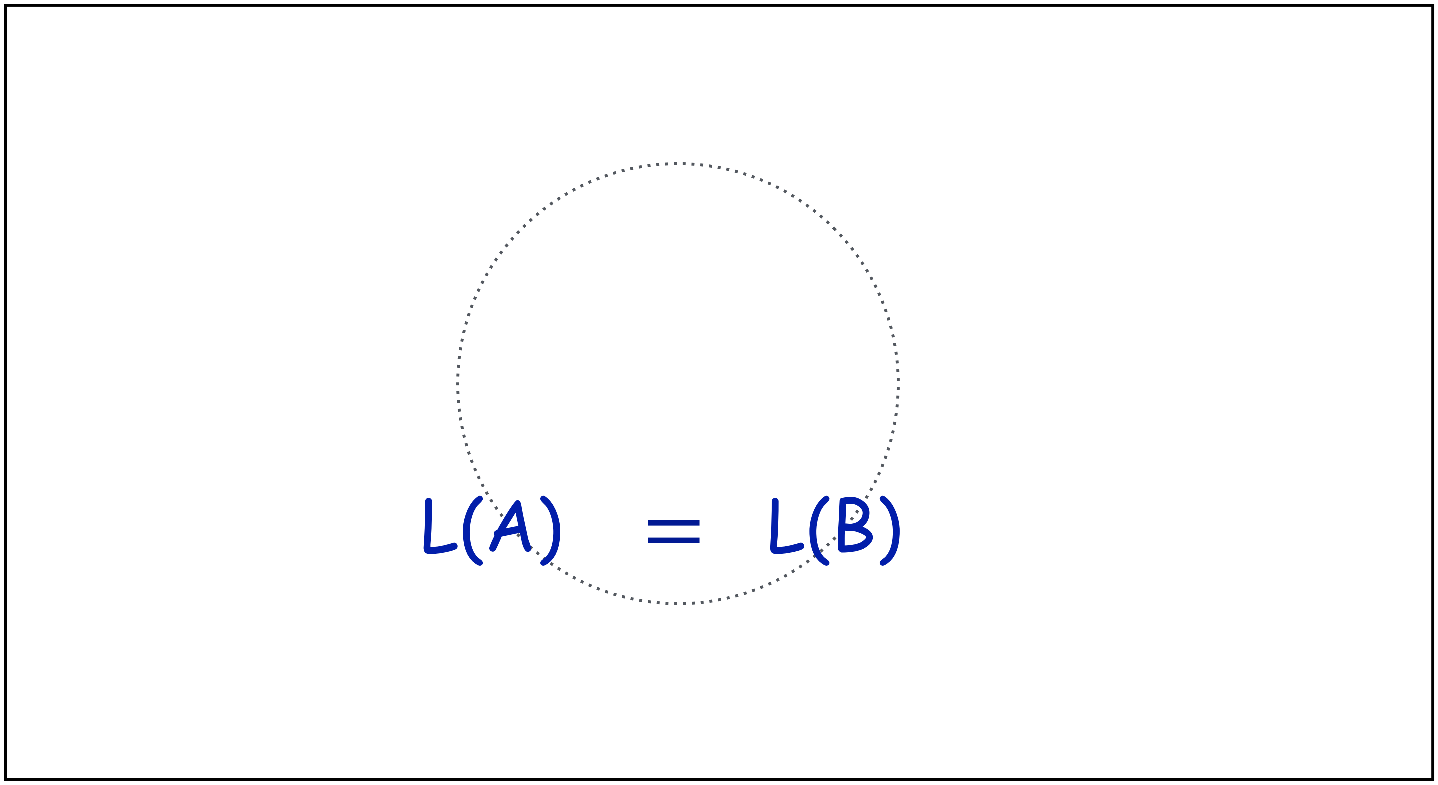
DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



DFA Equivalence Problem

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$



TM That Decides EQ_{DFA}

Q = "On input string <A,B>, where A and B are DFAs

1. Create DFA C such that

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

2. Submit C to Turing machine T that decides E_{DFA}

3. If T accepts C, **accept**. Otherwise, **reject**."

Some Decidable Languages

A_{DFA} = {<B,w> | B is a DFA that accepts input string w}

A_{NFA} = {<B,w> | B is an NFA that accepts input string w}

A_{REX} = {<R,w> | R is a regular expression that generates string w}

E_{DFA} = {<A> | A is a DFA and L(A) = ∅}

EQ_{DFA} = {<A,B> | A and B are DFA's and L(A) = L(B)}

Question

How would we show that the following language is decidable?

$$ALL_{DFA} = \{<A> \mid A \text{ is a DFA that recognizes } \Sigma^* \}$$

Another Question

Let L be any regular language

How would we show L is decidable?

- Assume L is described using a DFA

Deciders and CFG's

Consider the following language

A_{CFG} = {<G,w> | G is a CFG that generates string w}

Is A_{CFG} decidable?

Problem:

How can we get a TM to simulate a CFG?

Must be certain CFG tries a finite number of steps!

Solution: Use Chomsky Normal Form

Chomsky Normal Form Review

All rules are of the form

$A \to BC$

$A \to a$

where A, B, and C are any variables;
B and C cannot be the start variable

$S \to \varepsilon$

is the only ε rule;
S is the start variable

How Many Steps to Generate w?

If |w| = 0

1 step

If |w| = n > 0?

2n – 1 steps

TM Simulating A_{CFG}

M = "On input <G>, where G is a CFG

1. Convert G into Chomsky Normal Form
2. If |w| = 0
   - If there is an $S \rightarrow \varepsilon$ rule, accept
   - Otherwise, reject
3. List all derivations with 2|w|–1 steps
   - If any generate w, accept
   - Otherwise, reject"

Empty CFG's

Consider the following language

E_{CFG} = {<G> | G is a CFG and L(G) = ∅}

Theorem: E_{CFG} is decidable

Can we use the TM in A_{CFG} to prove this?

No.

There are infinitely many possible strings in Σ^*

Instead, we need to check if there is any way to get from the start variable to some string of terminals

Work Backwards

B = “On input <G>, where G is a CFG

1. Mark all terminals
2. Repeat until no new variables are marked

   Mark any variable A if G has a rule $A \rightarrow U_1U_2 \ldots U_k$ where $U_1, U_2, \ldots, U_k$ are all marked

✘ If S is marked, reject

✓ Otherwise, accept”

What About EQ_{CFG}?

Recall for EQ_{DFA}, we considered

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Will this work for CFG's?

- **No. CFG's are not closed under complementation or intersection**

EQ_{CFG} is *not* a decidable language!

- **We will see this later**

Decidability of CFL's

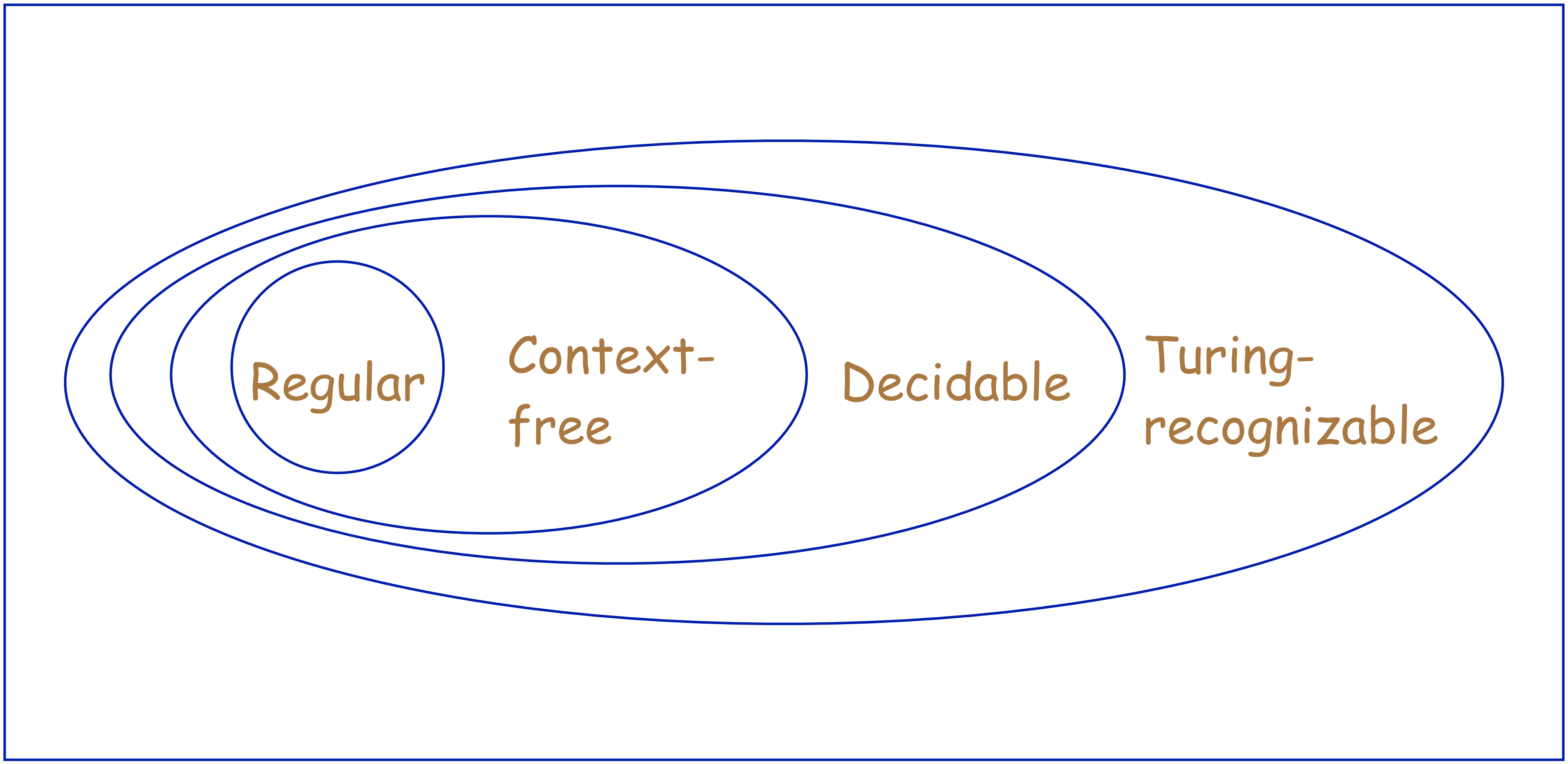
Theorem:

Every context-free language L is decidable

Proof:

For each w, we need to decide whether or not w is in L. Let G be a CFG for L. This problem boils down to A_{CFG}, which we showed is decidable.

Relationship of Classes of Languages



Languages We Know Are Decidable

Language	Input
A_{DFA}	$\langle D,w \rangle$, D is a DFA, w is a string
A_{NFA}	$\langle N,w \rangle$, N is an NFA, w is a string
A_{REX}	$\langle R,w \rangle$, R is an RE, w is a string
E_{DFA}	$\langle D \rangle$, D is a DFA and $L(D) = \emptyset$
EQ_{DFA}	$\langle C,D \rangle$, C and D are DFA's and $L(C) = L(D)$
L(R)	R is a regular language
A_{CFG}	$\langle G,w \rangle$, G is a CFG, w is a string
E_{CFG}	$\langle G \rangle$, G is a CFG and $L(G) = \emptyset$
L(C)	C is a context free language

Collaborative Exercise — 1

$F_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) \text{ is finite}\}$

Collaborative Exercise — 2

PRIME = { n | n is a prime number}

Collaborative Exercise — 3

CONN = {<G> | G is a connected graph}

Collaborative Exercise — 4

$L10_{DFA}$ = {D | D is a DFA that accepts every string w with |w| = 10}

Collaborative Exercise — 5

INT_{CFG} = {<G_1, G_2, w> | G_1 and G_2 are CFGs and w is accepted by both}

Collaborative Exercise — 6

$INTL_{CFG} = L(G_1 \cap G_2)$, where G_1 and G_2 are CFGs

Decidable Languages

A language is decidable if some Turing machine decides it

- Every string in Σ^* is either accepted or rejected

Not all languages can be decided by a Turing machine

Turing Machine Acceptance Problem

Consider the following language

A_{TM} = {<M,w> | M is a TM that accepts w}

Theorem: A_{TM} is Turing-recognizable

Theorem: A_{TM} is undecidable

Proof: The Universal Turing Machine recognizes, but does not decide, A_{TM}

The Universal Turing Machine

U = "On input <M, w>, 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
3. If M ever enters its reject state, reject"

Why Can't U Decide A_{TM}?

Intuitively, if M never halts on w,
then U never halts on <M,w>

This is also known as the *halting problem*

- **Given a TM M and a string w,**
  does M halt on input w?

- **Undecidable**

- **We may prove this more rigorously later**
  - **Need some additional tools for proving properties of languages**

Comparing the Size of Infinite Sets

Given two infinite sets A and B, is there any way of determining if |A|=|B| or if |A|>|B|?

Yes!

Functional correspondence can show two infinite sets have the same number of elements

Diagonalization can show one infinite set has more elements than another

Functional Correspondence

Let f be a function from A to B

f is called one-to-one if …

- $f(a_1) \neq f(a_2)$ whenever $a_1 \neq a_2$

f is called onto if …

- For every $b \in B$, there is some $a \in A$ such that $f(a) = b$

f is called a correspondence if it is both one-to-one *and* onto

- A correspondence is a way to pair elements of the two sets

Example — Correspondence

Consider f: $\mathbb{Z}^{\geq 0} \rightarrow P$, where

$\mathbb{Z}^{\geq 0}$ = {0,1,2,…} and P = {positive squares}

P = {1, 4, 9, 16, 25, …}

$f(x) = (x+1)^2$

Is f one-to-one?

Yes

Is f onto?

Yes

Therefore $|\mathbb{Z}^{\geq 0}| = |P|$

Comparing the Size of Infinite Sets

Given two infinite sets A and B, is there any way of determining if |A|=|B| or if |A|>|B|?

Yes!

Functional correspondence can show two infinite sets have the same number of elements

Diagonalization can show one infinite set has more elements than another

Countable Sets

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ the set of natural numbers

The set A is countable if …

- A is finite, or
- $|A| = |\mathbb{N}|$

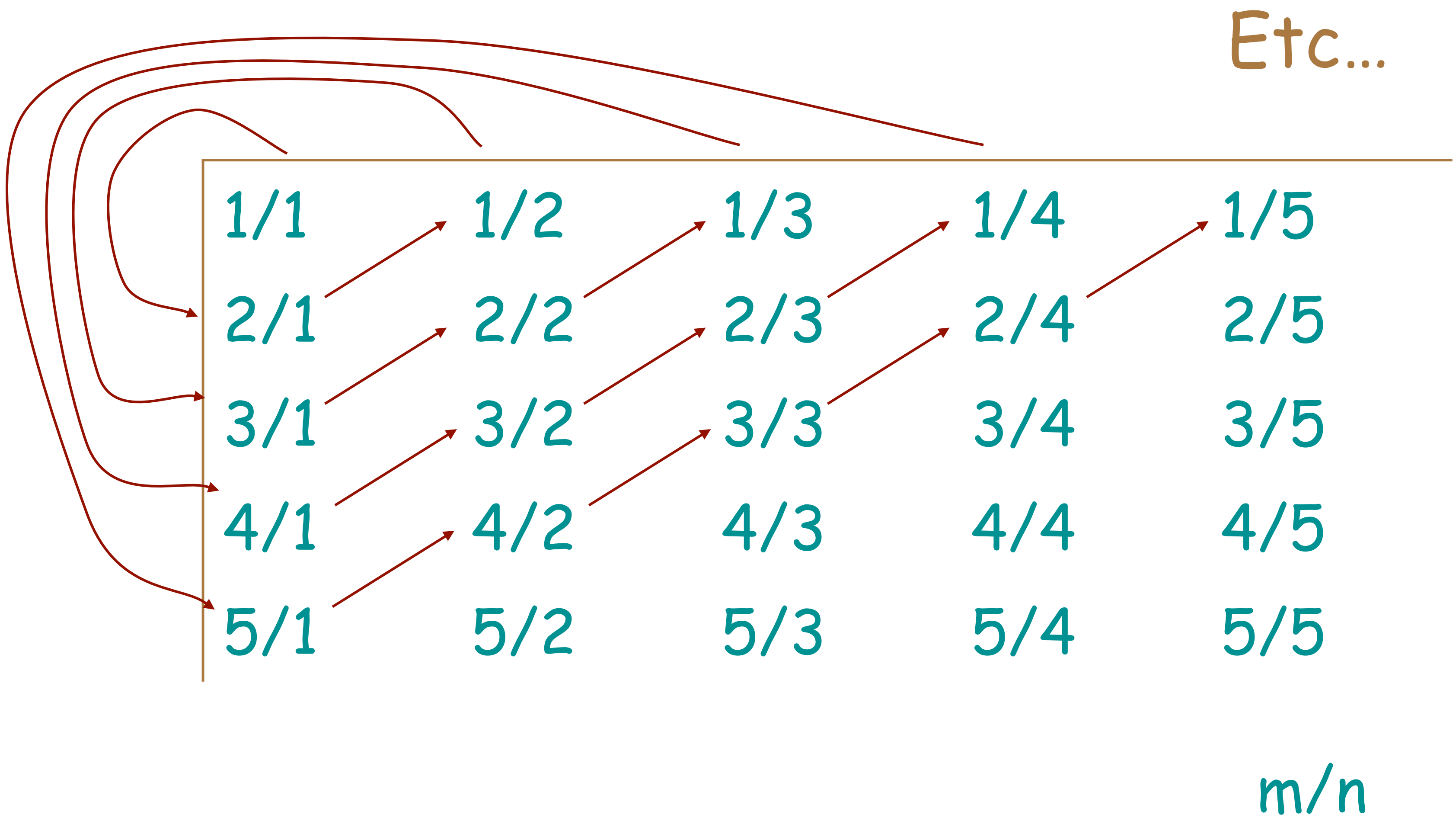
Some example of countable sets

- Integers `{0,-1,1,-2,2,-3,3,…}`
- $\{x \mid x \in \mathbb{N} \text{ and } (x \bmod 3) = 1\}$ `{1,4,7,10,…}`
- All positive primes `{2,3,5,7,11,…}`

The Positive Rational Numbers

Is Q = {m/n | m,n ∈ ℕ} countable?

Yes



The Real Numbers

Is $\mathbb{R}^+$ (the set of positive real numbers) countable?

No!

n	f(n)
1	<u>1</u>.56439...
2	3.<u>2</u>3891...
3	7.4<u>2</u>210...
4	2.22<u>2</u>66...
5	0.169<u>8</u>2...

X = 4.1337...

Diagonalization

The Real Numbers

The set of real numbers $\mathbb{R}$ is uncountable.

Proof by contradiction using diagonalization.

Assume that a correspondence f exists between $\mathbb{N}$ and $\mathbb{R}$.

Find an x in $\mathbb{R}$ that is not paired with anything in $\mathbb{N}$.

Construct such an x by choosing each digit of x to make x different from one of the real numbers that is paired with an element of $\mathbb{N}$, to ensure that $x \neq f(n)\ \forall n$.

We will construct x to be between 0 and 1, so all significant digits are part of the fractional part following the decimal point.

The Real Numbers

The set of real numbers $\mathbb{R}$ is uncountable.

To ensure that $x \neq f(1)$ we choose the first digit of x to be anything other than the first fractional digit of $f(1)$. Note that we have a choice of 9 other digits.

To ensure that $x \neq f(k)$ we choose the kth digit of x to be anything other than the kth digit of $f(k)$.

We continue down the diagonal of a table of $f(n)$ values.

We have constructed x so that if is not $f(n)$ for any n, because it differs from $f(n)$ in the nth fractional digit.

Thus we have a contradiction, since x is not paired with a number in $\mathbb{N}$.

The Real Numbers

Is $\mathbb{R}^+$ (the set of positive real numbers) countable?

No!

n	f(n)
1	0.<u>1</u>56439...
2	0.3<u>2</u>3891...
3	0.74<u>2</u>210...
4	0.222<u>2</u>66...
5	0.0169<u>8</u>2...

X = 0.41337...

Diagonalization

The Set of All Infinite Binary Strings

Is the set of all (infinite) binary strings countable?

- No
- Diagonalization also works to prove this is not countable

n	f(n)					
1	**1**	0	0	1	0	…
2	0	**1**	1	0	1	…
3	1	1	**0**	1	1	…
4	1	0	0	**1**	1	…
5	0	1	1	1	**0**	…

X = 0 0 1 0 1 …

The Set of All Infinite Binary Strings

Is the set of all (infinite) binary strings countable?

- No
- Diagonalization also works to prove this is not countable

On the other hand, the set of finite length binary strings is countable!

- Let x_b be the binary representation of x
- $f(x) = x_b$ is a 1-to-1 and onto function from $\mathbb{N}$ to the set of finite binary strings

$9_b = 1001$

The Set of All Binary Strings

Is the set of all binary strings countable?

- No
- Diagonalization works to prove this is not countable

The set of finite length binary strings is countable!

- Let x_b be the binary representation of x
- $f(x) = x_b$ is a 1-to-1 and onto function from $\mathbb{N}$ to the set of finite binary strings

Is the Set of All Languages in Σ^* Countable?

No

This set has the same cardinality as the set of all infinite binary strings

$$\Sigma^* = \{\ \varepsilon,\ a,\ b,\ aa,\ ab,\ ba,\ bb,\ aaa,\ aab,\ \ldots\}$$

$$A = \{\ a,\ ab,\ aaa,\ \ldots\ \}$$

$$\chi_A = 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ \ldots$$

The set of all languages in Σ^* is *not* countable

Σ^* vs. Languages in Σ^*

The set Σ^* *is* countable

- Let $|\Sigma| = n$
- Every string in Σ^* can be associated with a **unique** number, y, in base-(n+1)
- E.g., if $\Sigma = \{a, b, c\}$, we can associate the string **acba** with the value $1\times4^3+3\times4^2+2\times4^1+1\times4^0 = 121$
- Let f(x) be the string associated with x

The set of all languages in Σ^* is *not* countable

- It is the *power set* of Σ^*

Is the Set of All TM's Countable?

Yes

Every Turing machine can be represented by a finite length string, so the set of all Turing machines is countable

Theorem: Some languages are not Turing-recognizable

Proof: There are more languages than there are Turing machines

Some Languages Not Turing-recognizable

Theorem: Some languages are not Turing-recognizable

Proof: There are more languages than there are Turing machines

The set of all Turing machines is countable

The set of all languages is *not* countable

Undecidability of A_{TM}

Theorem: A_{TM} is undecidable

Proof: *(By Contradiction)*

Assume A_{TM} is decidable and let H be a decider for A_{TM}

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

H is a decider for A_{TM}

Undecidability of A_{TM} (continued)

Consider the TM D that submits the string <M> as input to the TM M

D = “On input <M>, where M is a TM:

Run H on input <M,<M>>

If H accepts <M,<M>>, reject

If H rejects <M,<M>>, accept

- Since H is a decider, it must accept or reject
- Therefore, D is a decider as well

H is a decider for A_{TM}

Undecidability of A_{TM} (continued)

What happens if D's input is <D>?

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

D cannot exist!

Therefore, H cannot exist

which is a contradiction

Thus A_{TM} is undecidable

Undecidability of A_{TM} (Review)

Assume H decides A_{TM}

- H(<M,w>) = accept if TM M accepts w,
  reject otherwise

Define D using H

- D(<M>) returns *opposite* of H(<M,<M>>)

Consider D(<D>)

- D *accepts* <D> if and only if D *rejects* <D>

Undecidability of A_{TM} (Review)

Assume H decides A_{TM}

- H(<M,w>) = accept if M accepts w
- H(<M,w>) = reject if M rejects w
- H(<M,<M>>) = reject if M rejects <M>

Define D using H

- D(<M>) = accept if H(<M,<M>>) = reject
- D(<M>) = accept if M rejects <M>
- D(<M>) = reject if M accepts <M>

Consider D(<D>)

- D(<D>) = accept if D rejects <D>
- **D *accepts* <D>** if and only if **D *rejects* <D>**

Undecidability of A_{TM} (Review)

Assume H decides A_{TM}

- H(<M,w>) = accept if M accepts w
- H(<M,w>) = reject if M rejects w
- H(<M,<M>>) = reject if M rejects <M>

Define D

- D(<M>) = accept if H(<M,<M>>) = reject
- D(<M>) = accept if M rejects <M>
- D(<M>) = reject if M accepts <M>

Consider D(<D>)

- D(<D>) = accept if D rejects <D>
- D *accepts* <D> if and only if D *rejects* <D>

A_{TM} is not decidable

CONTRADICTION

What about $\overline{A_{TM}}$?

What can we know about the complement of A_{TM}?

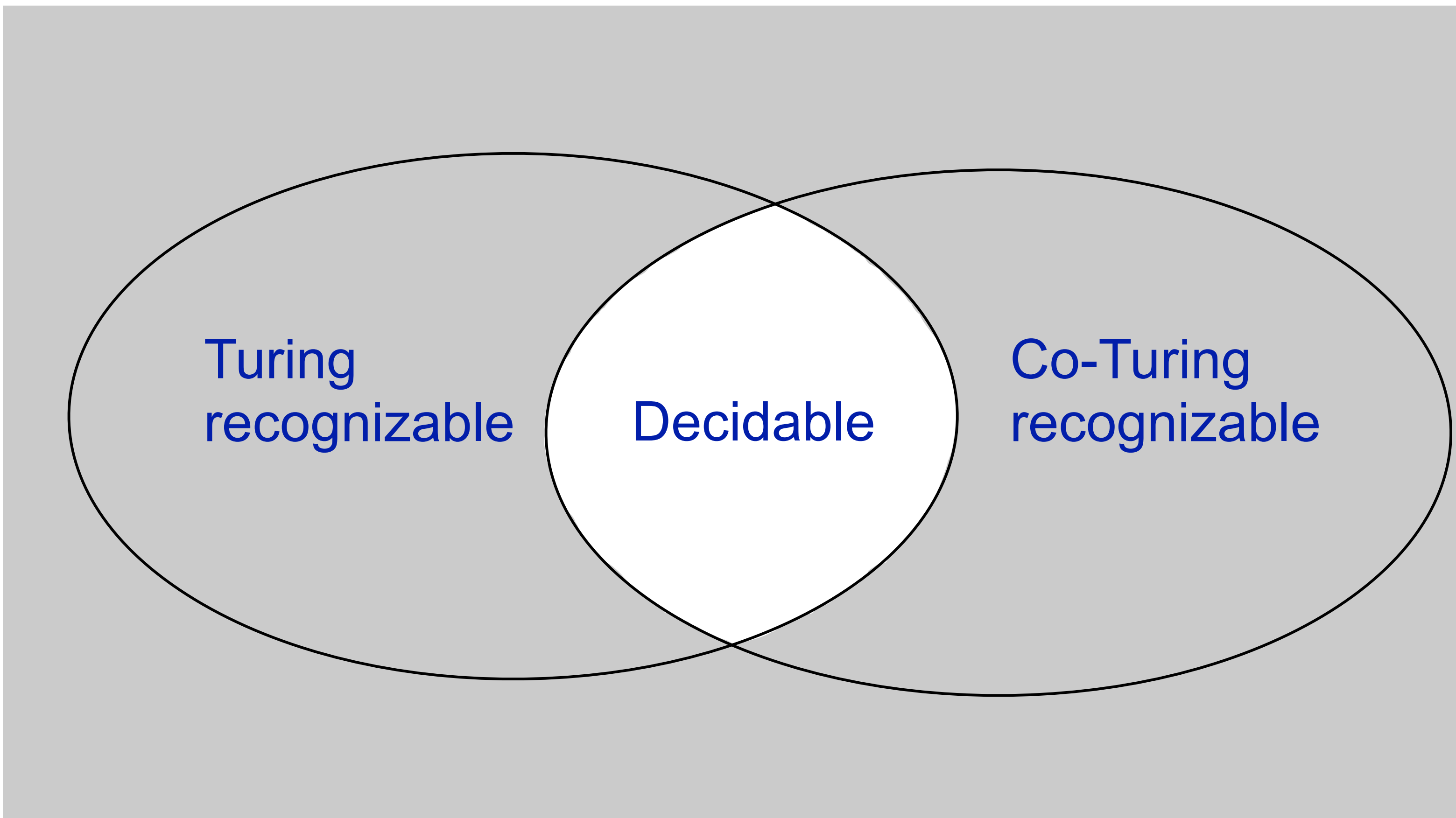
Can comp(A_{TM}) be decidable?

Can comp(A_{TM}) be recognizable?

We know that A_{TM} is Turing-recognizable.

What does it mean for both a language and its complement to both be Turing-recognizable?

Undecidable Languages



Coming Up

Proving a Language is Undecidable

- Use proof by contradiction
- Show that if a language L is decidable,
  it could be used to decide another language
  already known to be undecidable