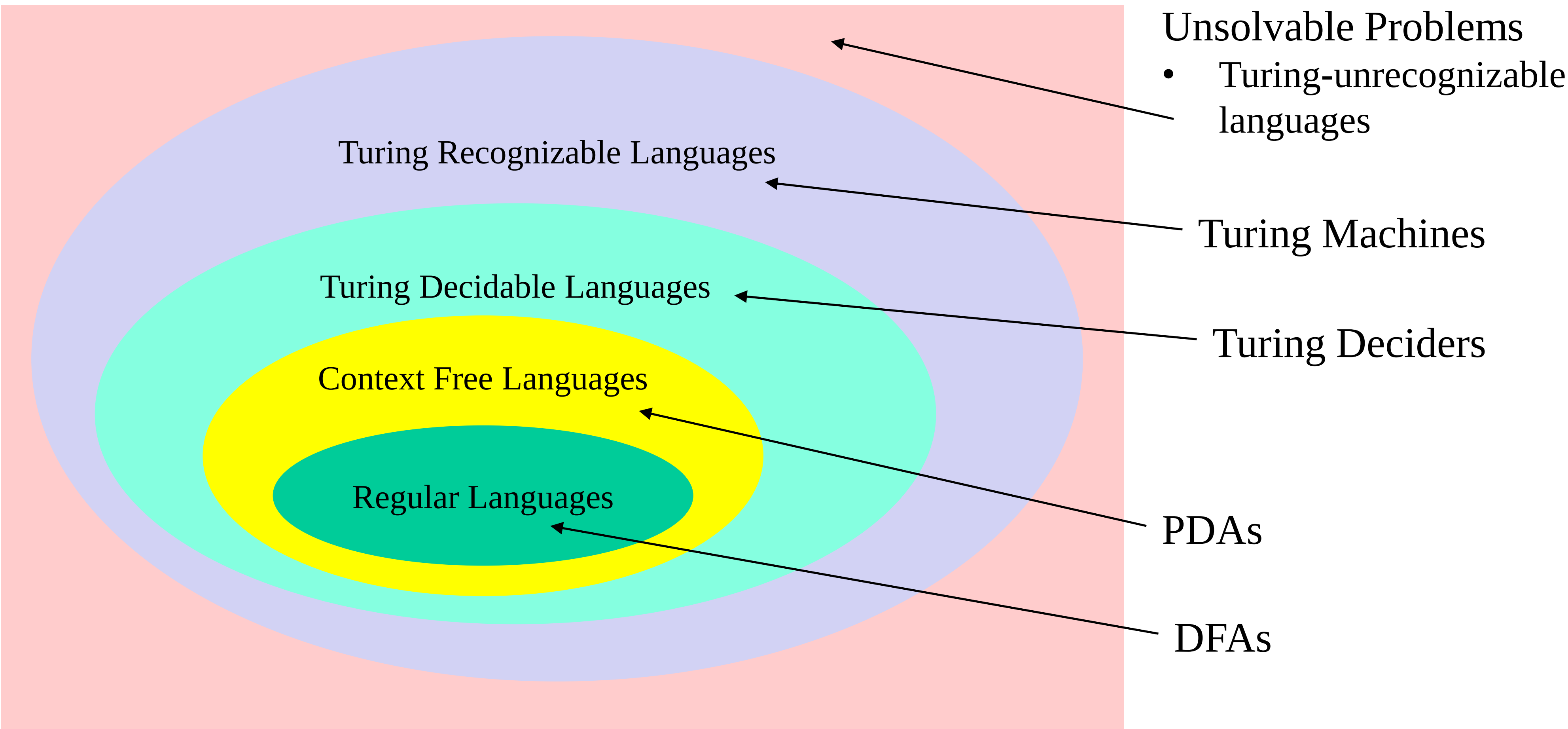
Decidability

Decidability



Turing Machines

- The most general model of computation
- Computations of a TM are described by a sequence of configurations. (Accepting Configuration, Rejecting Configuration)
- Turing-recognizable languages
  - TM halts in an accepting configuration if w is in the language.
  - TM may halt in a rejecting configuration or go on indefinitely if w is not in the language.
- Turing-decidable languages
  - TM halts in an accepting configuration if w is in the language.
  - TM halts in a rejecting configuration if w is not in the language.
- Nondeterministic TMs are equivalent to Deterministic TMs.
- Multitape TMs are equivalent to Deterministic TMs.

Decidability

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be solved.

- Why discuss **unsolvability**?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.

Decidable Languages

Encoding Finite Automata As Strings

- The inputs to TMs have to be strings.
- Every object O that enters a computation will be represented with a string <O>, encoding the object.

- **To represent a DFA as a string:**
- Encode Q using unary encoding:
  - For $Q = \{q_0,\ldots,q_n\}$ encode q_i using i+1 0's, i.e., using the string 0^{i+1}.
  - We assume that q_0 is always the start state.
- Encode Σ using unary encoding:
  - For $\Sigma = \{a_1,\ldots,a_m\}$, encode a_i using i 0's, i.e., using the string 0^i .
- With these conventions, all we need to encode is δ and F.
- Each entry of δ, e.g., $\delta(q_i,a_j) = q_k$ is encoded as

$$\underbrace{0^{i+1}}_{q_i}1\underbrace{0^{j}}_{a_j}1\underbrace{0^{k+1}}_{q_k}$$

Decidable Languages

Encoding Finite Automata As Strings

- The whole δ can now be encoded as

$$\underbrace{00100001000}_{transition_1}1\underbrace{000001001000000}_{transition_2}\cdots1\underbrace{000000100000010}_{transition_t}$$

- F can be encoded just as a list of the encodings of all the final states. For example, if states 2 and 4 are the final states, F could be encoded as

$$\underbrace{000}_{q_2}1\underbrace{00000}_{q_4}$$

- The whole DFA would be encoded by

$$11\underbrace{00100010000100000\cdots0}_{encoding\ of\ the\ transitions}11\underbrace{0000000010000000}_{encoding\ of\ the\ final\ states}11$$

Decidable Languages

Encoding Finite Automata As Strings

- <B> representing the encoding of the description of an automaton (DFA/NFA) would be something like

$$\langle B \rangle = 11\,\underbrace{001000100001000000\cdots0}_{\text{encoding of the transitions}}\,11\,\underbrace{0000000010000000}_{\text{encoding of the final states}}\,11$$

- In fact, the description of all DFAs could be described by a regular expression like

$$11(0^+10^+10^+1)^*1(0^+1)^+1$$

- Similarly strings over Σ can be encoded with (the same convention)

$$\langle w \rangle = \underbrace{0000}_{a_4}\,1\,\underbrace{000000}_{a_6}\,1\cdots\underbrace{0}_{a_1}$$

Decidable Problems on Regular Languages

- <B,w> represents the encoding of a machine followed by an input string, (with a suitable separator between <B> and <w>).
- Now we can describe our problems over languages and automata as problems over strings (representing automata and languages).

Decidable Problems on Regular Languages

- **Does B accept w?**
- **Is L(B) empty?**
- **Is L(A) = L(B)?**

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}.
  - This language contains the encodings of all DFAs together with strings that DFAs accept.

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

- The problem of testing whether a DFA B accepts an input w is the same as the problem of testing whether $\langle B,w \rangle$ is a member of the language A_{DFA}.
- Similarly, we can formulate other computational problems in terms of testing membership in a language.
- Showing that the language is decidable is the same as showing that the computational problem is decidable.

Acceptance Problem for DFAs

THEOREM: A_{DFA} = {<B,w> | B is a DFA that accepts input string w} is a decidable language.

PROOF IDEA:

- We simply need to present a TM M that decides A_{DFA}.
- 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, **accept** . If it ends in a non-accepting state, **reject** .**"**

Emptiness Problem for DFAs

THEOREM: E_{DFA} **= {<A> | A is a DFA and L(A) = ϕ} is a decidable language.**

PROOF:

- A DFA accepts some string iff reaching an accept state from the start state by traveling along the arrows of the DFA is possible.
- To test this condition, we can design a TM T that uses a marking algorithm

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** ."

Equivalence Problem for DFAs

THEOREM: **EQ_{DFA} = {<A,B> | A and B are DFAs and L(A) = L(B)} is a decidable language.**

PROOF:

- Construct the machine for

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

T = "On input <A,B> where A and B are DFAs.

1. Construct the DFA for L(C) as described above.
2. Run TM T of Emptiness Theorem on input <C>.
3. If T accepts, **accept**; otherwise **reject**."

Decidable Problems on Context-Free Languages

Decidable Problems on CFLs

- **Does grammar G generate w?**
- **Is L(G) empty?**

Undecidable Problems on CFLs

- **Is L(G) = L(H) for grammars G and H?**

Generation Problem for CFGs

THEOREM: A_{CFG} **= {<G,w> | G is a CFG that generates input string w} is a decidable language.**

PROOF :

- The TM S for A_{CFG} is as follows.

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 2n−1 steps, where n is the length of w; except if n = 0, then instead list all derivations with one step.
   - This works because every derivation using a CFG in CNF either increase the length of the sentential form by 1 (using a rule like $A \rightarrow BC$ or leaves it the same using a rule like $A \rightarrow a$)
   - Obviously this is not very efficient as there may be exponentially many strings of length up to 2n-1.
3. If any of these derivations generate w, **accept** ; if not, **reject** ."

Emptiness Problem for CFGs

THEOREM: **E_{CFG} = {<G> | G is a CFG and L(G) = ϕ} is a decidable language.**

PROOF:

- The TM R for E_{CFG} is as follows.

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 $A \rightarrow U_1 \ldots U_k$ and each symbol $U_1, \ldots, U_k$ has already been marked.
4. If the start variable is not marked, **accept** ; otherwise, **reject** ."

Equivalence Problem for CFGs

EQ_{CFG} = {<G,H> | G and H are CFGs and L(G) = L(H)} is NOT a decidable language.

- It turns out that EQ_{CFG} is **NOT a decidable language**.
- The construction does not work because CFLs are NOT closed under intersection and complementation.

Decidability of CFLs

THEOREM: Every context free language is decidable.

PROOF:

- Let G be a CFG for a CFL A and design a TM M_G that decides A. We build a copy of G into M_G.

M_G = "On input w:

1. Run TM S of Generation Theorem for CFGs on input <G,w>.
2. If this machine accepts, **accept** ; if it rejects, **reject** ."

Undecidability

- **What sorts of problems are unsolvable by computer?**
  - In one type of unsolvable problem, you are given a computer program and a precise specification of what that program is supposed to do. You need to verify that the program performs as specified or not.
  - *The general problem of software verification is* ***not*** *solvable by computer*.

- The problem of determining whether a Turing machine accepts a given input string is an undecidable problem

Acceptance Problem for TMs

- Remember that acceptance problems for DFAs and CFGs are **decidable** (i.e. A_{DFA} and A_{CFG} are decidable languages).

THEOREM: A_{TM} = {<M,w> | M is a TM and accepts string w} is UNDECIDABLE.

- Note that $\mathbf{A_{TM}}$ **is Turing-recognizable**.
- When this theorem is proved, it shows that **recognizers are more powerful than deciders.**
  - **Requiring a TM to halt on all inputs restricts the kinds of languages that it can recognize.**
- We can encode TMs with strings just like we did for DFAs

Acceptance Problem for TMs

THEOREM: A_{TM} = {<M,w> | M is a TM and accepts string w} is UNDECIDABLE.

- The following Turing machine U recognizes A_{TM}.

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

- Note that if M loops on w, then U loops on <M,w>, i.e. **U is NOT a decider**!
- **U can not detect that M halts on w.**
- **A_{TM} is also known as the Halting Problem**
- **U is known as the Universal Turing Machine because it can simulate every TM (including itself!)**

Diagonalization Method

- **The proof of the undecidability of A_{TM} uses a technique called diagonalization.**

Some Basic Definitions :

- Let A and B be any two sets (not necessarily finite) and f be a function from A to B.
- f is one-to-one if $f(a) \neq f(b)$ whenever $a \neq b$.
- f is onto if for every $b \in B$ there is an $a \in A$ such that $f(a)=b$.
- We say A and B are the **same size** if there is a one-to-one and onto function $f : A \rightarrow B$:
- Such a function is called a **correspondence** for pairing A and B.
  - Every element of A maps to a unique element of B
  - Each element of B has a unique element of A mapping to it.

Diagonalization Method

Countable Set

- Let N be the set of natural numbers {1,2,3,…}and let E be the set of even numbers {2,4,6,…}.
- f(n)=2n is a **correspondence** mapping N to E.
- **Hence, N and E have the same size** (even though E⊂N).

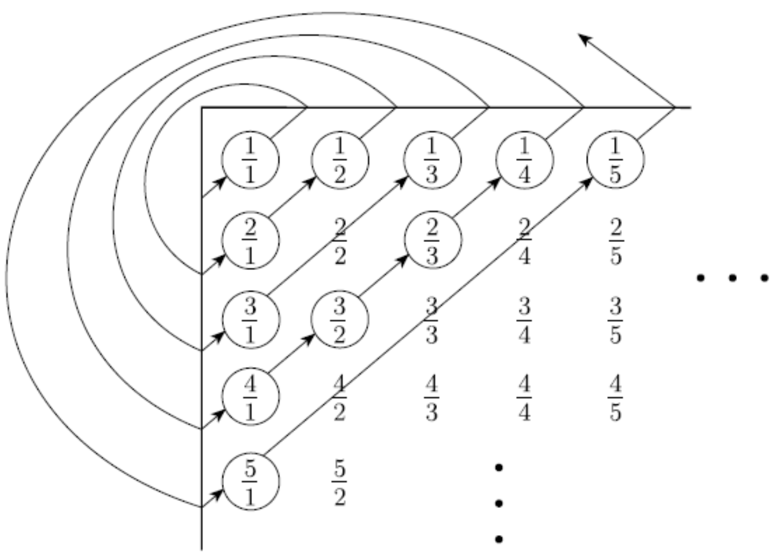
Definition: Countable Set

A set S is countable if it is either finite or has the same size as N (natural numbers).

Diagonalization Method

Countable Set

- **Positive rational numbers Q = { m/n | m, n ∈ N } is countable.**
  - **Correspondence:**
    - list all the elements of Q.
    - Then we pair the first element on the list with the number 1 from N, the second element on the list with the number 2 from N, and so on.
    - We must ensure that every member of Q appears only once on the list.



Diagonalization Method

Uncountable Set

- Are there infinite sets that are uncountable (i.e. No correspondence with N)? **YES**

THEOREM: The set of positive real numbers R are uncountable.

PROOF:

- In order to show that R is uncountable, we show that no correspondence exists between N and R.
- The proof is by contradiction.
  - Suppose that a correspondence f existed between N and R.
  - Our job is to show that f fails to work as it should.
  - For it to be a correspondence, 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, which will be our contradiction.

Diagonalization Method

Uncountable Set

THEOREM: The set of positive real numbers R are uncountable.

PROOF:

- Assume f exists and every number in R is listed.
- Assume x∈R is a real number such that x differs from the j^{th} number in the j^{th} decimal digit.
- If x is listed at some position k, then it differs from itself at k^{th} position; otherwise the premise does not hold.
- f does not exist.

n	$f(n)$
1	3.<u>1</u>4159...
2	55.5<u>5</u>555...
3	0.12<u>3</u>45...
4	0.500<u>0</u>0...
⋮	⋮

x = .4627… defined as such, can not be on this list.

Diagonalization over Languages

- **How many languages are there?** ➔ **uncountably many languages**
- **How many TMs are there?** ➔ **countably many TMs**

- This means that there are some languages that
  - They are not decidable and even they are not Turing recognizable.

COROLLARY: Some languages are not Turing-recognizable.

- In order to prove this corollary, we have to show that there are countably many TMs and there are uncountably many languages.

Diagonalization over Languages

COROLLARY: Some languages are not Turing-recognizable.

PROOF: To show that the set of all TMS is countable:

- **For any alphabet Σ, Σ^* is countable.**
  - Order strings in Σ^* by length and then alphanumerically, so $\Sigma^* = \{s_1, s_2, \ldots\}$
  - Σ^* is countable.
- **The set of all TMs is a countable language.**
  - Each TM M corresponds to a string <M>.
  - Generate a list of strings and remove any strings that do not represent a TM to get a list of TMs.
  - Since Σ^* is countable, the set of all TMs is a countable language.

Diagonalization over Languages

COROLLARY: Some languages are not Turing-recognizable.

PROOF: To show that the set of all languages is uncountable:

- **The set of infinite binary sequences B is uncountable.**
  - The same proof we gave for uncountability of R.
- **The set of all languages L is uncountable.**
  - Let L be the set of all languages over Σ.
  - For each language $A \in L$ there is unique infinite binary sequence X_A
    - The i^{th} bit in X_A is 1 if $s_i \in A$, 0 otherwise.

$\Sigma^* = \{$	ε,	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	$\cdots$

  - The function $f : L \rightarrow B$ is a correspondence. Thus L is uncountable.

- **So, there are languages that can not be recognized by some TM.**
- **There are not enough TMs to go around.**

Acceptance Problem for TMs
Halting Problem is Undecidable

THEOREM: A_{TM} = {<M,w> | M is a TM and accepts string w} is UNDECIDABLE.

PROOF:

- We assume that A_{TM} is decidable and obtain a contradiction.
- Suppose that H is a decider for A_{TM}, i.e. H is a TM where

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

  - H produces *reject* if M rejects w or M runs forever.

Acceptance Problem for TMs
Halting Problem is Undecidable

PROOF (cont.):

- Now, construct a new TM D

  $D =$ "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*."

- So,

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

- Run D with its own description <D> as input:

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

- No matter what D does, it is forced to do the opposite, ➔ a contradiction.
- Thus, neither TM D nor TM H can exist.

Diagonalization in Halting Problem

- **Where is the diagonalization in the proof of Halting Problem?**

- List all TMs down the rows, $M_1, M_2, \ldots$, and all their descriptions across the columns, $<M_1>$, $<M_2>$, …
- The entries tell whether the machine in a given row accepts the input in a given column.
  - The entry is **accept** if the machine accepts the input but is **blank** if it rejects or loops on that input.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$		$\vdots$			

Diagonalization in Halting Problem

- The results of running H on inputs:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$		$\vdots$			

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$		$\vdots$			

Diagonalization in Halting Problem

- Consider the behavior of all possible deciders:
  - D computes the opposite of the diagonal entries!

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$			$\vdots$				$\ddots$

a contradiction occurs at "?"

A Turing-Unrecognizable Language

- **A_{TM} is an undecidable language (but it is a Turing-recognizable language).**

- **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.

- In other words, a language is decidable exactly when both it and its complement are Turing-recognizable.

A Turing-Unrecognizable Language

COROLLARY: $\overline{\mathbf{ATM}}$ **is not Turing-recognizable.**

PROOF:

- We know $\mathbf{A_{TM}}$ is Turing-recognizable.
- If $\overline{\mathbf{ATM}}$ were also Turing-recognizable, $\mathbf{A_{TM}}$ would have to be decidable.
- We know $\mathbf{A_{TM}}$ is not decidable.
- $\overline{\mathbf{ATM}}$ must not be Turing-recognizable.