# Decidability II

## Sides based on material by Kani, Erickson, Chekuri, et. al.

All mistakes are my own! - Ivan Abraham (Fall 2024)

Image by ChatGPT (probably collaborated with DALL-E)

# Reduction

**Meta definition:** Problem $X$ *reduces* to problem $Y$, if having a solution to $Y$, implies a solution to $X$. We denote this as $X \implies Y$.

**Definition:** Oracle $\mathrm{ORAC}_L$ for language $L$ is a function that maps a word $w$ to $TRUE \iff w \in L$

**Lemma:** A language $X$ reduces to a language $Y$, if one can construct a *TM* decider for $X$ using an oracle $\mathrm{ORAC}_Y$ for $Y$.

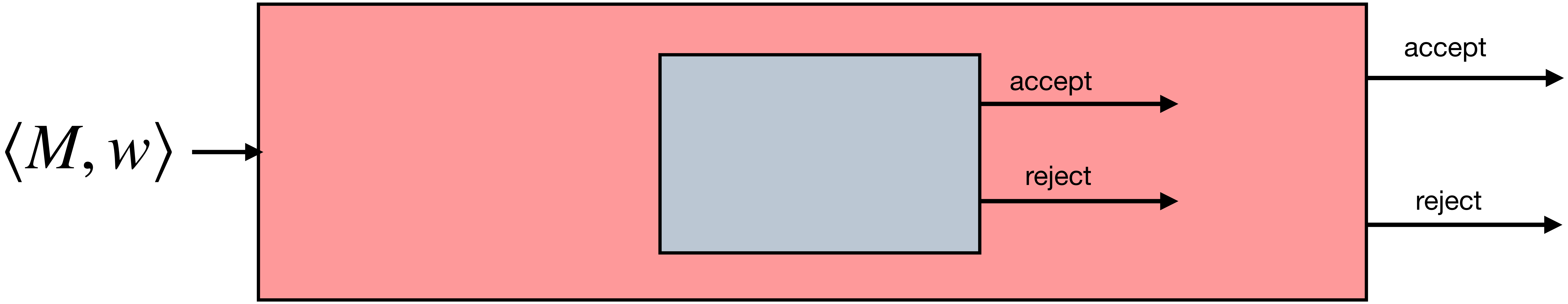
We will also denote this by $X \implies Y$.

# Reduction proof technique

- Let $Y$ be the problem/language for which we want to prove something (e.g. undecidability) and denote by $L$ the language of $Y$.
- Proof via reduction is essentially a proof by contradiction.
- Assume $L$ is decided by a *TM* $M$. Create a decider for ***known*** undecidable problem $X$ *using* $M$.
    - Results in decider for $X$ (i.e., $A_{\mathrm{TM}}$).
- Contradiction since $X$ is known not to be decidable. Thus, $L$ must be not decidable.

# Key diagram

## A picture is worth atleast a few slides of proofs.



# Reduction implies decidability

**Lemma:** Let $X$ and $Y$ be two languages, and assume that $X \implies Y$. Then if $Y$ is decidable then $X$ is decidable.

**Proof:** Since $X$ reduces to $Y$, there is a procedure $T_{X|Y}$ (i.e., decider) for $X$ that uses an oracle for $Y$ as a subroutine. Since $Y$ is decidable, let $T$ be a decider for $Y$ (i.e., a program or a *TM*). We replace the calls to the oracle in $T_{X|Y}$ with calls to $T$. The resulting program $T_X$ is a decider and its language is $X$. Thus $X$ is decidable (or more formally *TM* decidable).

# The contrapositive …

**Lemma:** Let $X$ and $Y$ be two languages, and assume that $X \implies Y$. Then if $X$ is undecidable then $Y$ is undecidable.

# Halting

## The halting problem

Define the language of all pairs $\langle M, w \rangle$ such that $M$ halts on $w$ as:

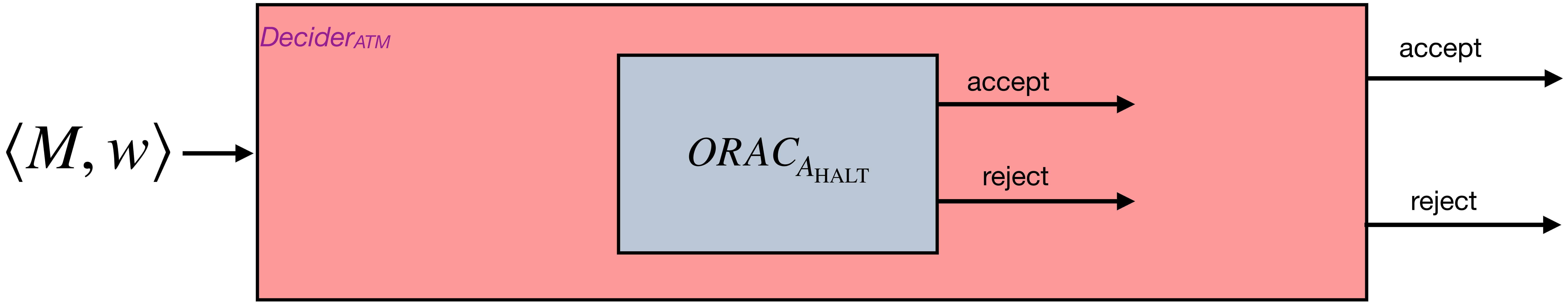
$$A_{\text{HALT}} = \{\langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ stops on } w\}$$

"Similar" to language we already know to be undecidable:

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

# One way to proving that Halting is undecidable...

**Lemma:** The language $A_{TM}$ reduces to $A_{\text{HALT}}$. Namely, given an oracle for $A_{\text{HALT}}$ one can build a decider (that uses this oracle) for $A_{TM}$.



# One way to proving that Halting is undecidable...

**Proof:** Let $ORAC_{A_{\mathrm{HALT}}}$ be the given oracle for $A_{\mathrm{HALT}}$. We build the following decider for $A_{TM}$.

```
AnotherDecider-A_TM(⟨M,w⟩):
  res ← ORAC_Halt(⟨M,w⟩)
  // if M does not halt on w then reject
  if res = reject then
       halt and reject
  // M halts on w since res = accept.
  // Simulating M on w terminates in finite time.
  res_2 ← Simulate M on w
  return res_2
```

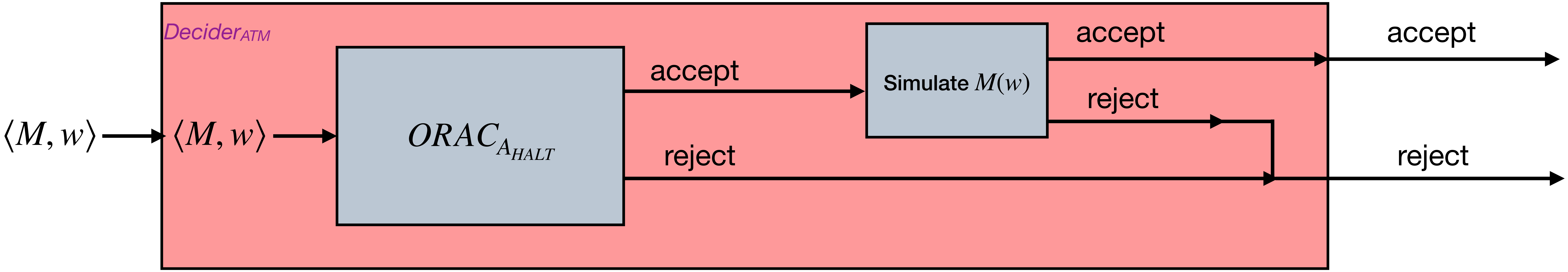
This procedure always returns and as such its a decider for $A_{TM}$.

# The Halting problem is not decidable

**Theorem:** The language $A_{\text{HALT}}$ is not decidable.

**Proof:** Assume, for the sake of contradiction, that $A_{\text{HALT}}$ is decidable. As such, there is a $TM$, denoted by $TM_{\text{HALT}}$, that is a decider for $A_{\text{HALT}}$. We can use $TM_{\text{HALT}}$ as an implementation of an oracle for $A_{\text{HALT}}$, which would imply that one can build a decider for $A_{TM}$. However, $A_{TM}$ is undecidable which is contradiction. Therefore it must be the case that $A_{\text{HALT}}$ is undecidable.

# The same proof by figure...

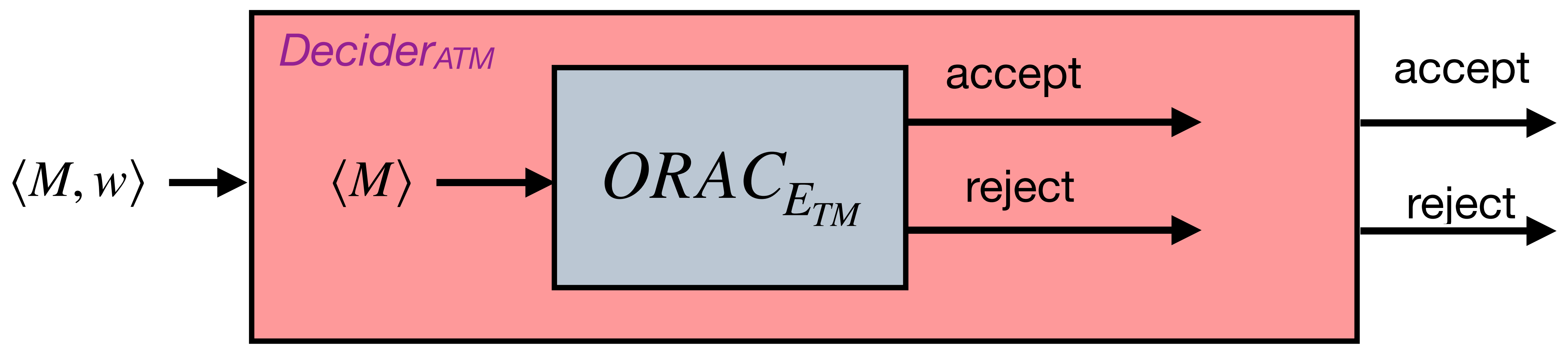



... if $A_{\mathrm{HALT}}$ is decidable, then $A_{TM}$ is decidable, which is impossible!

# Emptiness

## The language of empty languages

- Let $E_{TM} = \{\langle M \rangle \mid M \text{ is a } TM \text{ and } L(M) = \emptyset\}$ and let $TM_{ETM}$ be a decider for $E_{TM}$.
- Need to use $TM_{ETM}$ to build a decider for $A_{TM}$.
- Decider for $A_{TM}$: Given $M$ and $w$ decide whether $M$ accepts $w$.
- Need to somehow make the second input $(w)$ disappear



# Embedding strings

- Suppose given program $\langle M \rangle$ and input $w$ we can output a program $\langle M_w \rangle$.
  - The program $\langle M_w \rangle$ simulates $M$ on $w$. And accepts/rejects accordingly.
- Let EmbedString$\big(\langle M \rangle, w\big)$ take as input two strings $\langle M \rangle$ and $w$, and outputs a string encoding (TM) $\langle M_w \rangle$.

**Question:** What is $L(M_w)$?

Since $M_w$ ignores any input … language $M_w$ is either $\Sigma^*$ or $\emptyset$. It is $\Sigma^*$ if $M$ accepts $w$, and it is $\emptyset$ if $M$ does not accept $w$.

# Emptiness is …

**Theorem**: The language $E_{TM}$ is undecidable.

- Assume (for contradiction), that $E_{TM}$ is decidable and let $TM_{ETM}$ be its decider.
- Build decider AnotherDecider-A$_{TM}$ for $A_{TM}$:

```
AnotherDecider-ATM(⟨M, w⟩):
  ⟨Mw⟩ ← EmbedString(⟨M⟩, w)
  r ← TMETM(⟨Mw⟩)
  if r = accept then
       return reject
  // TMETM(⟨Mw⟩) rejected its input
  return accept
```

# … is undecidable.

Consider the possible behavior of AnotherDecider-$A_{TM}$ on the input $\langle M, w \rangle$.

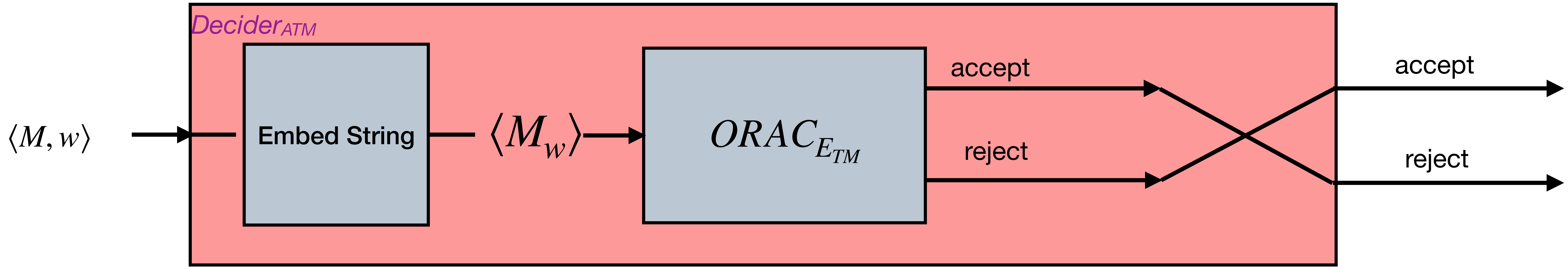
- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L\langle M_w \rangle$ is empty. This implies that $M$ does not accept $w$. As such, AnotherDecider-$A_{TM}$ rejects its input $\langle M, w \rangle$.

- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L\langle M_w \rangle$ is not empty. This implies that $M$ accepts $w$. So AnotherDecider-$A_{TM}$ accepts $\langle M, w \rangle$.

$\implies$ AnotherDecider-$A_{TM}$ is a decider for $A_{TM}$

But $A_{TM}$ is undecidable … so the assumption that $E_{TM}$ is decidable must be false.

# Emptiness is undecidable via diagram




NOTE: AnotherDecider-$A_{TM}$ never actually runs the code for $M_w$. It hands the code to a function $TM_{ETM}$ which analyzes what the code would do if run*. So it does not matter that $M_w$ might go into an infinite loop.

# Equality

## Equality is undecidable

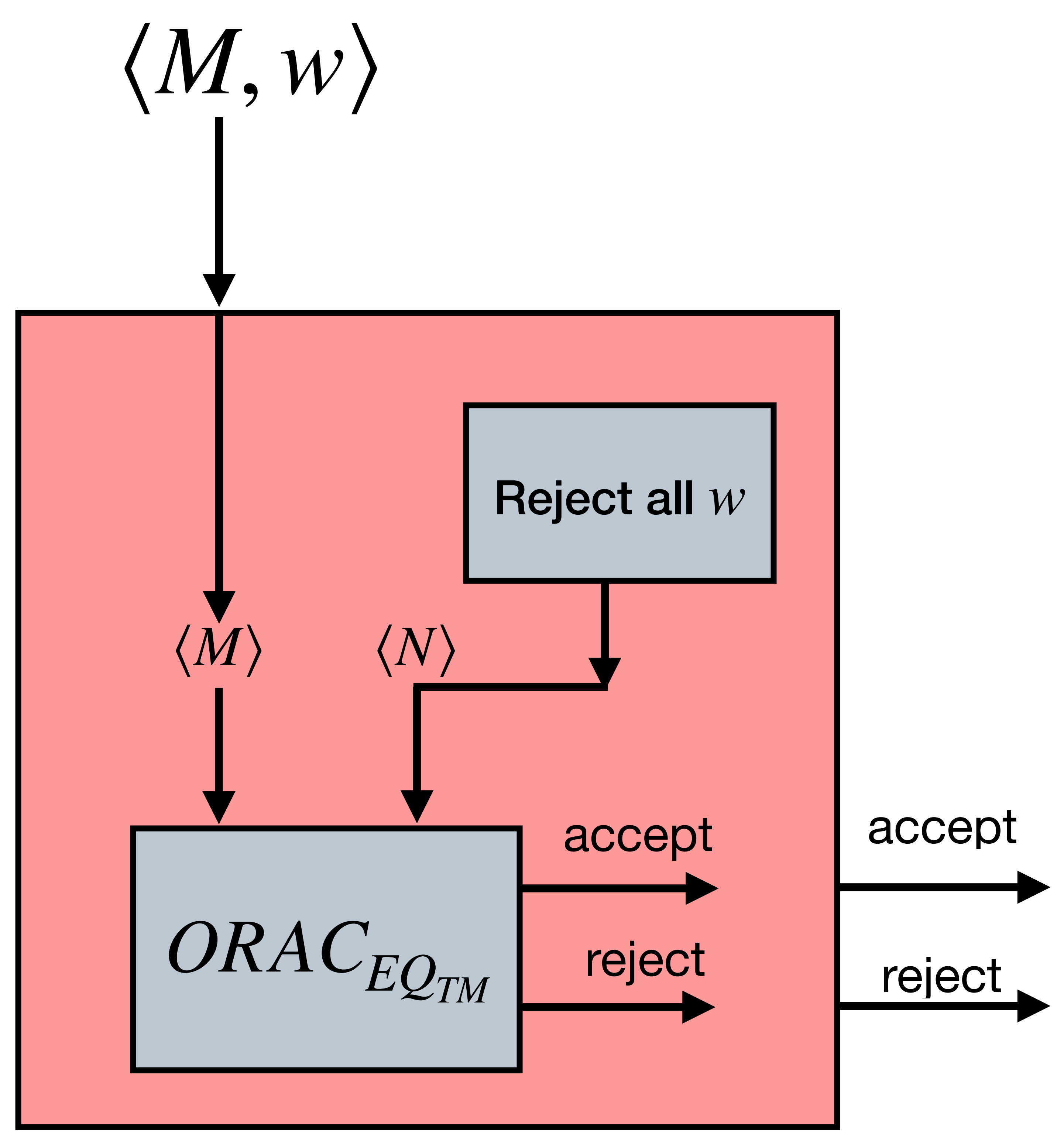
Let:

$$EQ_{TM} = \{\langle M, N \rangle \mid M \text{ and } N \text{ are TM's and } L(M) = L(N)\}$$

**Lemma:** The language $EQ_{TM}$ is undecidable

Let's try something different. We know $E_{TM}$ is undecidable.

Let's use that: $E_{TM} \implies EQ_{TM}$



# DFAs

## DFAs are empty?

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

What does the above language describe?

Is the language above decidable?

**Lemma**

The language $E_{DFA}$ is decidable.

# Proof

Unlike in the previous cases, we can directly build a decider (DeciderEmptyDFA) for $E_{DFA}$.

1. Input = $\langle A \rangle$
2. Mark start state of *A* as visited.
3. Repeat until no new states get marked:
   - Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, then accept.
5. Otherwise, then reject.

# Equal DFAs

## DFAs are equal?

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

What does the above language describe?

Is the language above decidable?

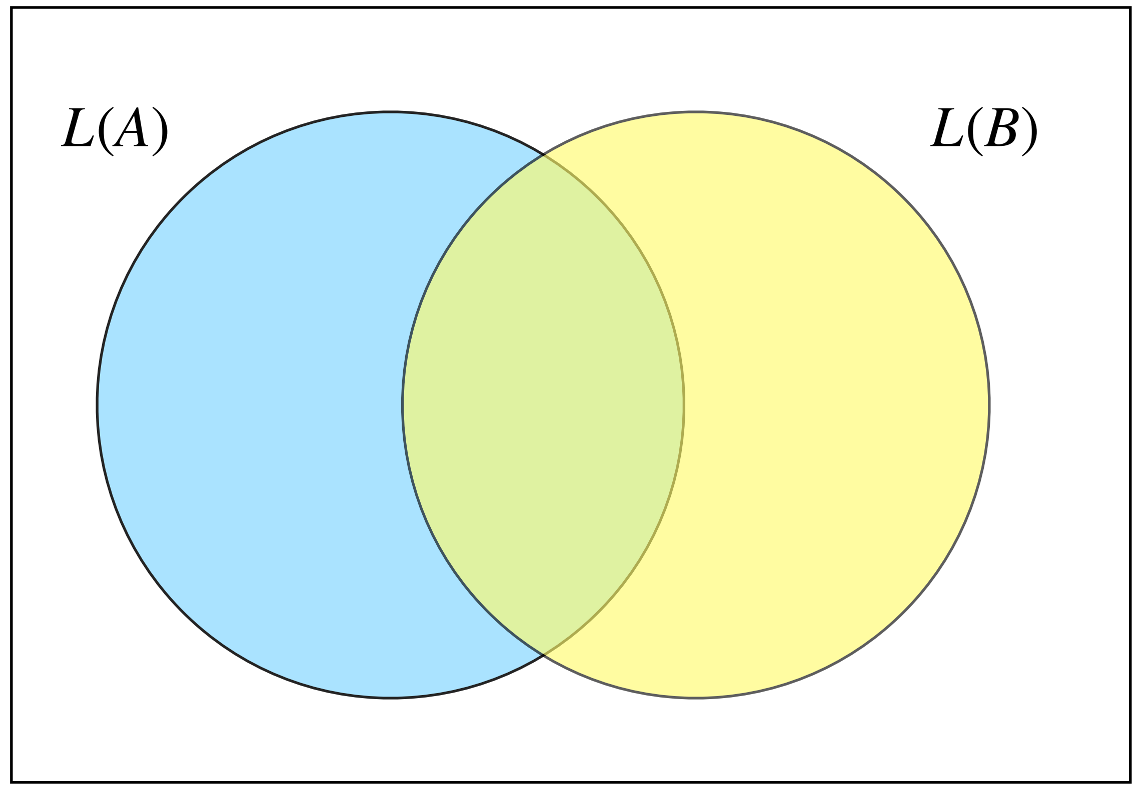
**Lemma**

The language $E_{DFA}$ is decidable.

Can we show this using reductions?

# Equal DFA trick I

Need a way to determine if there any strings in one language and not the other...




This is known as the symmetric difference. Given $A$, $B$, can create a new $DFA$ (call it $C$) which represents the symmetric difference of $L_A$ and $L_B$.

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

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

# Equal DFA trick II

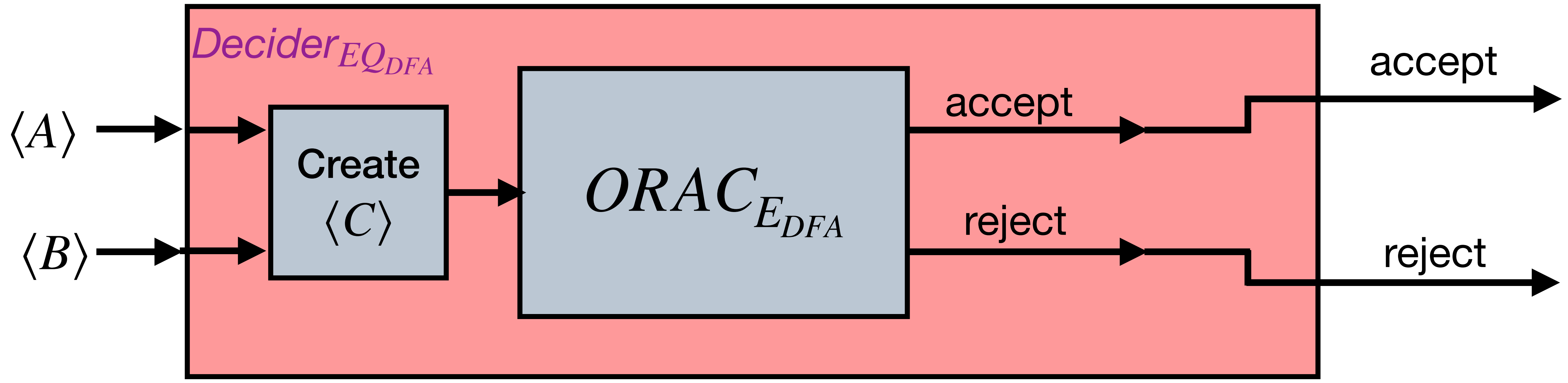
Notice with $L(C)$:

- If $L(A) = L(B)$ then $L(C) = \emptyset$
- If $L(A) \neq L(B)$ then $L(C)$ is not empty

Good time to use $E_{DFA}$ proof from before…

How do we show $EQ_{DFA}$ is decidable using a reduction?

Want to show $EQ_{DFA} \implies E_{DFA}$



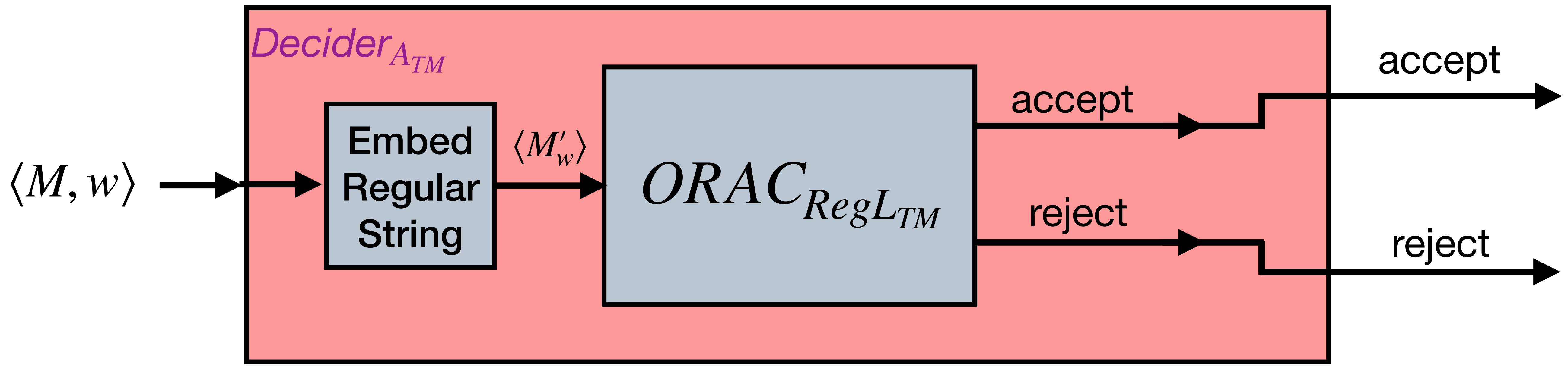
# Regularity

- Turns out, almost any property defining a *TM* language induces a language which is undecidable.
- The proofs all have the same basic pattern.
- E.g.— Regularity language: $\text{Regular}_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a } TM \text{ and } L(M) \text{ is regular}\}$
- $\text{DeciderRegL/OracRegL}_{\text{TM}}$: Assume *TM* decider for $\text{Regular}_{\text{TM}}$.
- Reduction from halting requires to turn problem about deciding whether a *TM* $M$ accepts $w$ (i.e., is $w \in A_{TM}$) into a problem about whether some *TM* accepts a regular set of strings.

# Proof idea

- Given $M$ and $w$, consider the following TM, $M'_w$:
    - Input = $x$, if $x$ has the form $a^n b^n$ , halt and accept.
    - Otherwise, simulate $M$ on $w$.
        - If the simulation accepts, then accept.
        - If the simulation rejects, then reject.
- Feed $\langle M'_w \rangle$ into OracRegLTM
    - Assume EmbedRegularString: program with input $\langle M \rangle$ and $w$, and outputs $\langle M'_w \rangle$, encoding the program $M'_w$.

# IsRegular reduction




- If $M$ accepts $w$, then any $x$ is accepted by $M'_w : L(M'_w) = \Sigma^*$ .
- If $M$ does not accept $w$, then $L(M'_w) = \{a^n b^n \,|\, n \geq 0\}$ .

- If OracRegLTM accepts $\implies L(M'_w)$ regular (its $\Sigma^*$ ) $\implies$ $M$ accepts $w$. So AnotherDecider-$A_{TM}$ should accept $\langle M, w \rangle$
- If OracRegLTM rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n$ $\implies$ $M$ does not accept $w \implies$ AnotherDecider-$A_{TM}$ should reject $\langle M, w \rangle$

# Rice's theorem

The above proofs were somewhat repetitious... ...they imply a more general result.

**Theorem (Rice's Theorem)**

Suppose that $L$ is a language of Turing machines; that is, each word in $L$ encodes a TM. Furthermore, assume that the following two properties hold.

(a) Membership in $L$ depends only on the Turing machine's language, i.e. if $L(M) = L(N)$ then $\langle M \rangle \in L \Leftrightarrow \langle N \rangle \in L$.

(b) The set $L$ is "non-trivial," i.e. $L \neq \emptyset$ and $L$ does not contain all Turing machines.

Then $L$ is a undecidable.

# T’is the …

- Happy Fall Break!
  - Stay warm …
  - … travel safe!