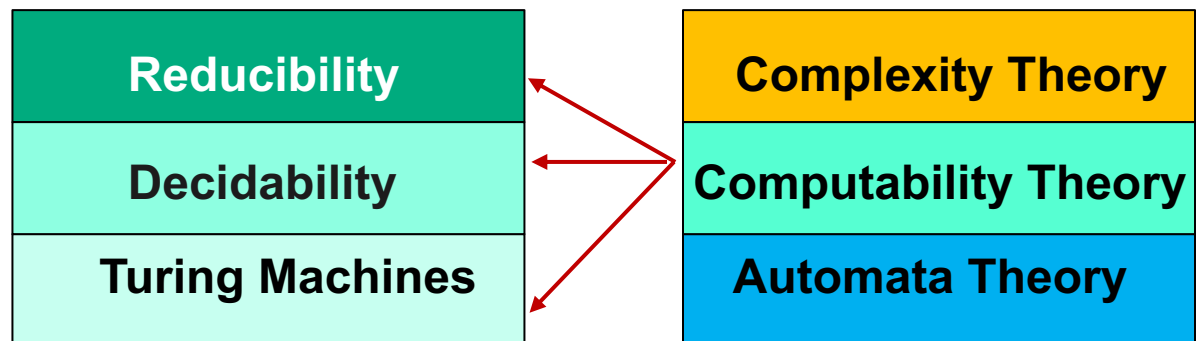




Reducibility, part 3





Agenda

- Finish discussion on Linearly Bounded Automaton (LBA)
- Mapping Reducibility



Warm-up question (4/22/22)

What is the best book you read this year or last year?

Send me your response by Canvas email.



Recap: so far on “reductions”...

Decidable:

A_{DFA} (acceptance)
 A_{NFA}
 A_{REG}
 E_{DFA} (emptiness)
 EQ_{DFA} (equivalence)
 A_{CFG}
 E_{CFG}
Every CFG

Undecidable:

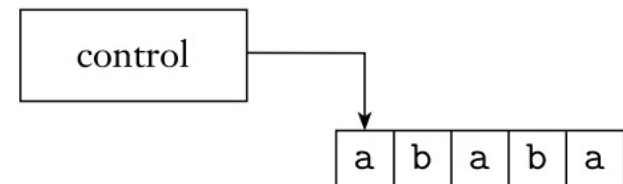
A_{TM} ← diagonalization
 HALT_{TM}
 E_{TM}
 EQ_{TM}
 $\text{REGULAR}_{\text{TM}}$
 CFL_{TM}
 $\text{DECIDABLE}_{\text{TM}}$ } Rice's theorem

reduction



Recap: Definition – linear bounded automaton (LBA)

A *linear bounded automaton* is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.



An LBA is a TM with a limited amount of memory



Recap: A_{LBA}

Let:

$A_{\text{LBA}} = \{\langle M, w \rangle \mid M \text{ is an LBA that accepts string } w\}.$

Lemma:

Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n .

Theorem:

A_{LBA} is decidable.



New case: E_{LBA}

Let:

$$E_{\text{LBA}} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$$

Theorem:

E_{LBA} is undecidable.



Proof idea

- The proof is reduction from A_{TM}
 - We show that if E_{LBA} were decidable, A_{TM} would also be decidable
- Suppose that E_{LBA} is decidable
 - How can we use this assumption to decide A_{TM} ?
- For a TM M and an input w , we can determine whether M accepts w by constructing a certain LBA B and then testing whether $L(B)$ is empty
- The language that B recognizes comprises all accepting computation histories for M on w
- If M accepts w , this language contains one string and so is nonempty
- If M does not accept w , this language is empty
- If we can determine whether B 's language is empty, clearly we can determine whether M accepts w



Proof idea

- Now we describe how to construct **B** from **M** and **w**
- Note that we need to show more than the mere existence of **B**
- We have to show how a TM can obtain a description of **B**, given descriptions of **M** and **w**
- Note that we construct **B** only to feed its description into the presumed E_{LBA} decider, but not to run **B** on some input



Proof idea

- We construct **B** to accept its input **x** if **x** is an accepting computation history for **M** on **w**
- We assume that the accepting computation history is presented as a single string with the configurations separated from each other by the **#** symbol

$$\# \underbrace{\hspace{1.5cm}}_{C_1} \# \underbrace{\hspace{1.5cm}}_{C_2} \# \underbrace{\hspace{1.5cm}}_{C_3} \# \dots \# \underbrace{\hspace{1.5cm}}_{C_l} \#$$



Proof idea

- The LBA B works as follows
 - When it receives an input x , B is supposed to accept if x is an accepting computation history for M on w
 - First, B breaks up x according to the delimiters into strings C_1, C_2, \dots, C_l
 - Then B determines whether the C_i 's satisfy the three conditions of an accepting computation history.
 1. C_1 is the start configuration for M on w
 2. Each C_{i+1} legally follows from C_i
 3. C_l is an accepting configuration for M
- By inverting the decider's answer, we obtain the answer to whether M accepts w
- Thus we can decide A_{TM} , a contradiction

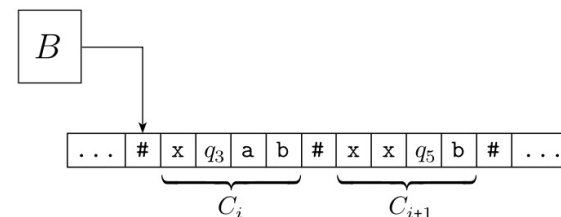
Putting it together: the proof

PROOF Now we are ready to state the reduction of A_{TM} to E_{LBA} . Suppose that TM R decides E_{LBA} . Construct TM S to decide A_{TM} as follows.

$S =$ “On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct LBA B from M and w as described in the proof idea.
2. Run R on input $\langle B \rangle$.
3. If R rejects, *accept*; if R accepts, *reject*.”

If R accepts $\langle B \rangle$, then $L(B) = \emptyset$. Thus, M has no accepting computation history on w and M doesn't accept w . Consequently, S rejects $\langle M, w \rangle$. Similarly, if R rejects $\langle B \rangle$, the language of B is nonempty. The only string that B can accept is an accepting computation history for M on w . Thus, M must accept w . Consequently, S accepts $\langle M, w \rangle$. Figure 5.12 illustrates LBA B .





One last result, ALL_{CFG}

Let:

$$ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}.$$

Theorem:

ALL_{CFG} is undecidable.

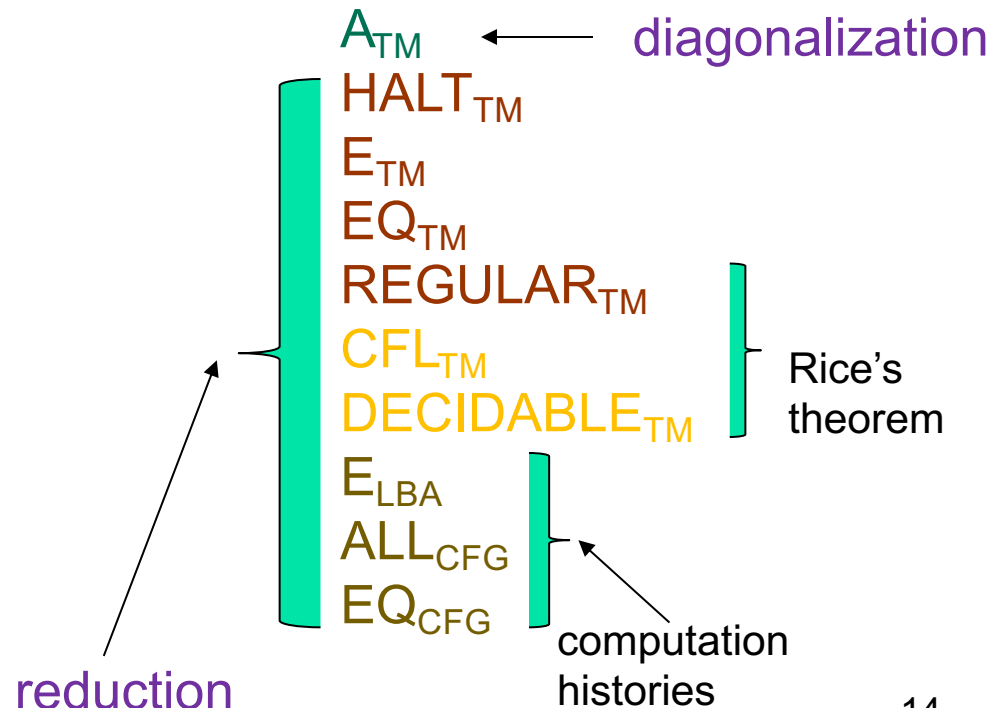
We state this result without proof, and it is enough for our purposes.
Also this result can be used to show that **EQ_{CFG} is undecidable**.

Another look at our growing list...

Decidable:

A_{DFA} (acceptance)
 A_{NFA}
 A_{REG}
 E_{DFA} (emptiness)
 EQ_{DFA} (equivalence)
 A_{CFG}
 E_{CFG}
 A_{LBA}

Undecidable:





Mapping Reducibility



Mapping reducibility

- In the last few lectures, we have been seeing how to use the **reducibility** technique to prove that various problems are **undecidable**
- Now we **formalize the notion of reducibility**
- This enables us to use reducibility in more refined ways, for example
 - To show that certain languages are **not Turing-recognizable**
 - For applications in **complexity theory**
- We use a simple formalization called ***mapping reducibility***



Mapping reducibility: the idea

- Being able to reduce problem A to problem B using a **mapping reducibility** means that a **computable function** exists that **converts instances of problem A to instances of problem B**.
- If we have such a conversion, called a **reduction**, we can solve A with a solver for B.



Computable functions

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a *computable function* if some Turing machine M , on every input w , halts with just $f(w)$ on its tape.

Examples of computable functions:

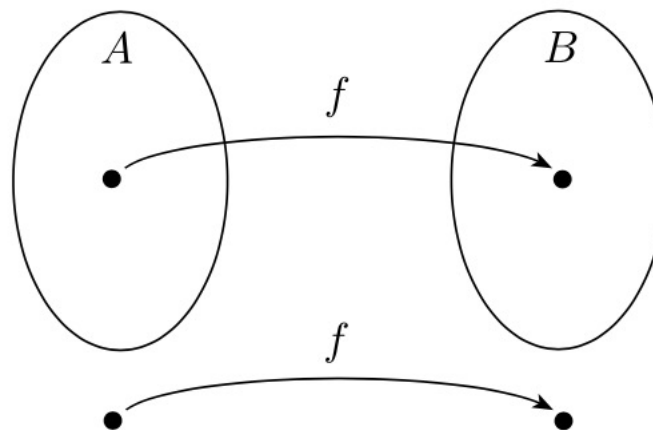
- **Arithmetic operations** (e.g. sum, product)
 - A machine that takes input $\langle m, n \rangle$ and returns $m + n$
- **Transformation of machine descriptions**
 - A function f that takes input w and returns a description of a TM $\langle M' \rangle$ if $w = \langle M \rangle$ is an encoding of a TM M .
The machine M' recognizes the same language as M , but never attempts to move its head off the left-hand end of its tape.

Mapping reducibility: formal definition

Language A is **mapping reducible** to language B , written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every w ,

$$w \in A \iff f(w) \in B.$$

The function f is called the **reduction** from A to B .





Mapping reducibility uses: preview

- Decidable problems
- Undecidable problems
- Recognizability of certain languages
- Nonrecognizability of certain languages



Mapping reducibility for decidable problems

If $A \leq_m B$ and B is decidable, then A is decidable.

PROOF We let M be the decider for B and f be the reduction from A to B . We describe a decider N for A as follows.

$N =$ “On input w :

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.”

Clearly, if $w \in A$, then $f(w) \in B$ because f is a reduction from A to B . Thus, M accepts $f(w)$ whenever $w \in A$. Therefore, N works as desired.



Mapping reducibility for undecidable problems

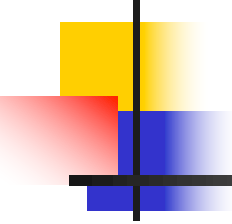
If $A \leq_m B$ and A is undecidable, then B is undecidable.

Example: Reduction from A_{TM} to prove that $HALT_{TM}$ is undecidable.
We present a computable function f that takes input of the form $\langle M, w \rangle$ and returns output of the form $\langle M', w' \rangle$, where
 $\langle M, w \rangle \in A_{TM}$ if and only if $\langle M', w' \rangle \in HALT_{TM}$

The following machine F computes a reduction f .

$F =$ “On input $\langle M, w \rangle$:

1. Construct the following machine M' .
 $M' =$ “On input x :
 1. Run M on x .
 2. If M accepts, *accept*.
 3. If M rejects, enter a loop.”
2. Output $\langle M', w \rangle$.”



Proof E_{TM} is undecidable using reducibility (old slide from lecture last week)

$M_1 =$ “On input x :

1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does.”

- This machine has the string w as part of its description. It conducts the test of whether $x = w$ in the obvious way.
- Putting all this together, we assume that TM R decides E_{TM} and construct TM S That decides A_{TM} as follows.

$S =$ “On input $\langle M, w \rangle$, an encoding of a TM M and a string w :

1. Use the description of M and w to construct the TM M_1 just described.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*.”

- If R were a decider for E_{TM} , S would be a decider for A_{TM} .
- A decider for A_{TM} cannot exist, so we know that E_{TM} must be undecidable.



Comment on the proof via reducibility

- From the original reduction, we may easily construct a function f that takes input $\langle M, w \rangle$ and produces output $\langle M_1 \rangle$
- But M accepts w iff $L(M_1)$ is not empty so f is a mapping reduction from A_{TM} to E_{TM} complement.
- It still shows that E_{TM} is undecidable because decidability is not affected by complementation, but it doesn't give a mapping reduction from A_{TM} to E_{TM}
- In fact, no such reduction exists

- The sensitivity of mapping reducibility to complementation is important in the use of reducibility to prove nonrecognizability of certain languages.



Mapping reducibility for Turing recognizable problems

Theorem:

If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

recognizer

PROOF We let M be the ~~decider~~ for B and f be the reduction from A to B . We describe a ~~decider~~ N for A as follows.

$N =$ “On input w :

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.”

Clearly, if $w \in A$, then $f(w) \in B$ because f is a reduction from A to B . Thus, M accepts $f(w)$ whenever $w \in A$. Therefore, N works as desired.



Mapping reducibility for Turing nonrecognizable problems

Corollary to theorem from previous slide:

If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

New theorem:

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.



EQ_{TM} is neither Turing-recognizable nor co-Turing recognizable

$F =$ “On input $\langle M, w \rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2 .

$M_1 =$ “On any input:

1. *Reject*.”

$M_2 =$ “On any input:

1. Run M on w . If it accepts, *accept*.”

2. Output $\langle M_1, M_2 \rangle$.”

EQ_{TM} is not
Turing recognizable

EQ_{TM} complement
is not
Turing recognizable

$G =$ “On input $\langle M, w \rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2 .

$M_1 =$ “On any input:

1. *Accept*.”

$M_2 =$ “On any input:

1. Run M on w .
2. If it accepts, *accept*.”

2. Output $\langle M_1, M_2 \rangle$.”