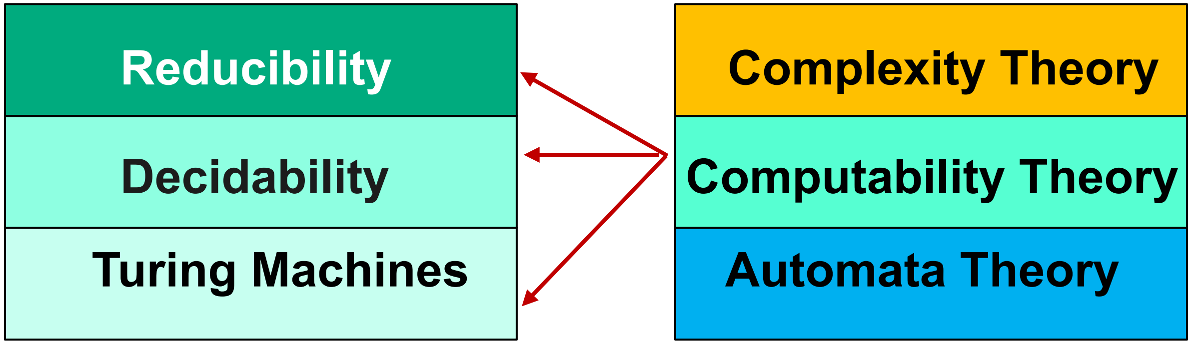
Reducibility



Warm-up question (4/18/22)

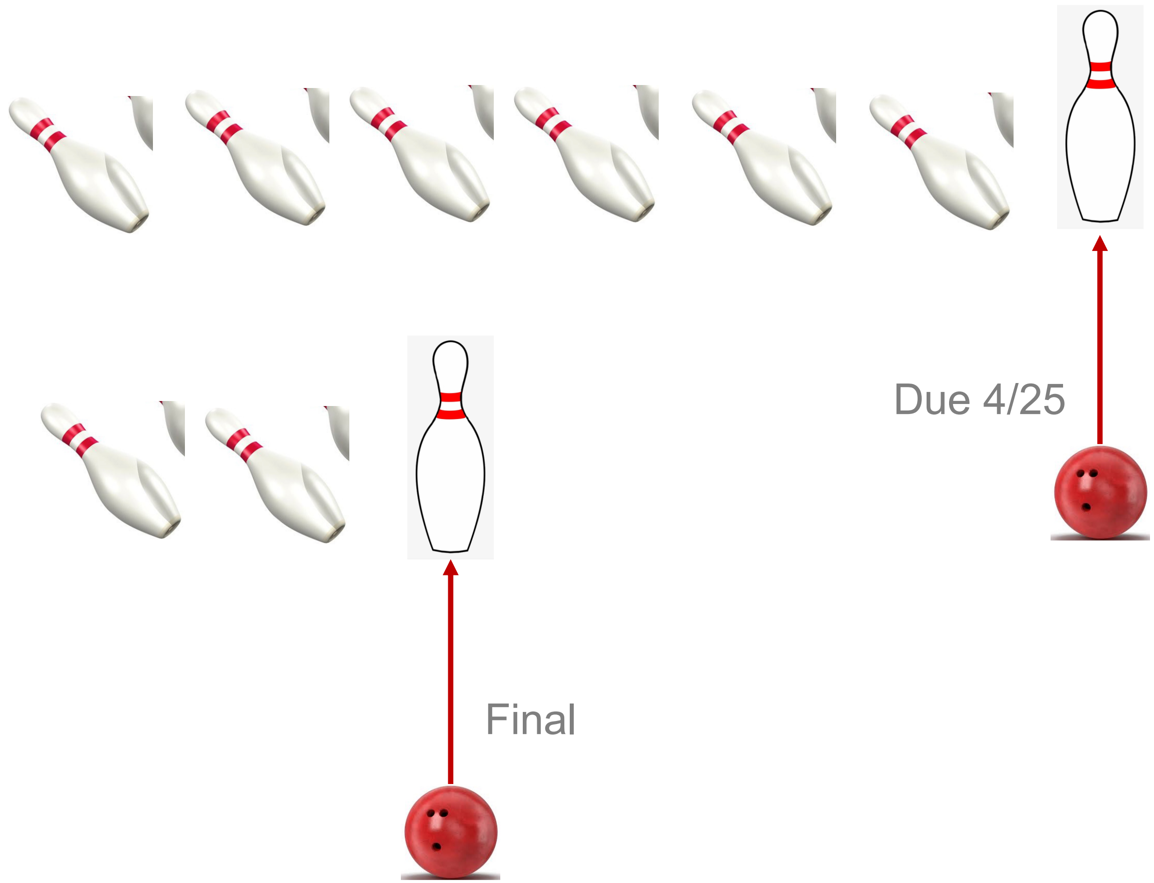
Write Happy Monday in a language other than English.

Send me your response by Canvas email.

Status

HWs

Exams



Participation: stay engaged to maintain your current good score

In the last few lectures, we…

- Established the TM as our model of a general purpose computer
- Presented several examples of problems that are solvable on a TM
- Gave one example of a problem that is computationally unsolvable (A_{TM})

In the last few lectures, we saw…

Decidable:

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

Undecidable:

A_{TM}

Today and in the next few lectures, we will…

- Examine several additional (besides A_{TM}) unsolvable problems

- Learn about the primary method for proving that problems are computationally unsolvable – that method is called reducibility

Reduction

Problem A → Problem B

A way of converting problem A to problem B
such that solution to
Problem B can be used to solve Problem A

Examples:

- Everyday life
  - Traveling, cooking
- Mathematical problems
  - Area calculation, solving systems of equations
- Classifying problems by decidability, complexity

Reducibility in computability theory

$\boxed{\text{Problem A}} \longrightarrow \boxed{\text{Problem B}}$

If A is reducible to B, and

- If B is **decidable**, then A also is **decidable**
- If A is **undecidable**, then B is **undecidable**

Undecidable problems from language theory: Example 1 (Halting Problem)

Recall:

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

A_{TM} is undecidable

Let:

$HALT_{TM}$ = {<M,w> | M is a TM and M halts on input w}

Theorem:

$HALT_{TM}$ is undecidable

Proof idea

- The proof is by contradiction
- The key idea is to show that A_{TM} is reducible to $HALT_{TM}$
- Assume we have a TM R that decides $HALT_{TM}$
- Then use R to construct S, a TM that decides A_{TM}
  - If R indicates that M doesn't halt on w, reject because <M,w> isn't in A_{TM}
  - However, if R indicates that M does halt on w, you can simulate R without any danger of looping
- Thus, if TM R exists, we can decide A_{TM}, but we know A_{TM} is undecidable. By virtue of this contradiction, we can conclude that R does not exist.
- Therefore, $HALT_{TM}$ is undecidable

Proof

- Assume for the purpose of obtaining a contradiction that TM R decides $HALT_{TM}$.
- Construct TM S to decide A_{TM}, with S operating as follows.

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

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

- Clearly, if R decides $HALT_{TM}$, then S decides A_{TM}
- Because A_{TM} is undecidable, $HALT_{TM}$ also must be undecidable

Example 2: E_{TM}

Let:

$E_{TM} = \{<M> \mid M \text{ is a TM and } L(M) = \emptyset\}$

Theorem:

E_{TM} is undecidable

Proof Idea

- Assume that E_{TM} is decidable and then show that A_{TM} is decidable – a contradiction
- Let R be a TM that decides E_{TM}
- We use R to construct TM S that decides A_{TM}
- How will S work when it receives input <M,w>?
  - Idea 1: S runs R on input <M> and see whether it accepts.
  - If it does, we know that L(M) is empty and therefore that M does not accept w
  - But if R rejects <M>, all we know is that L(M) is nonempty and therefore that M accepts some string – but we still do not know whether M accepts the particular string w
  - So we need a different idea

Proof idea 2

- Instead of running R on <M>, we run R on a modification of <M>
- We modify <M> to guarantee that M rejects all strings except w, but on input w it works as usual
- Then we use R to determine whether the modified machine recognizes the empty language
- The only string the machine can now accept is w, so its language will be nonempty iff it accepts w
- If R accepts when it is fed a description of the modified language, we know that the modified machine doesn't accept anything and that M doesn't accept w

Proof

- Let's write the modified machine described in proof idea 2 using our standard notation. We call it M_1.

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.

Example 3: $REGULAR_{TM}$

Let:

$REGULAR_{TM}$ = {<M> | M is a TM and L(M) is a regular language}

Theorem:

$REGULAR_{TM}$ is undecidable

Proof idea

- We assume that $REGULAR_{TM}$ is decidable by a TM R and use this assumption to construct s TM S that decides A_{TM}
- The idea is for S to take its input <M,w> and modify M so that the resulting TM recognizes a regular language iff M accepts w.
- We call the modified language M_2
- We design M_2 to recognize the nonregular language

$\{0^n1^n \mid n \geq 0\}$ if M does not accept w, and to recognize the regular language Σ^* if M accepts w

- We must specify how S can construct such an M_2 from M and w
- Here, M_2 works by automatically accepting all strings in

$\{0^n1^n \mid n \geq 0\}$

- In addition, if M accepts w, M_2 accepts all other strings

Proof

- We let R be a TM that decides $REGULAR_{TM,}$ and Construct TM S to decide A_{TM}.
- Then S works in the following manner.

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

1. Construct the following TM M_2.

$M_2 =$ "On input x:

1. If x has the form $0^n 1^n$, *accept*.

2. If x does not have this form, run M on input w and *accept* if M accepts w."

2. Run R on input $\langle M_2 \rangle$.

3. If R accepts, *accept*; if R rejects, *reject*."