Theory of Formal Languages and Automata

Lecture 20

Mahdi Dolati

Sharif University of Technology

Fall 2023

December 15, 2023

Reducibility

- Reducibility: The primary **method** for proving that problems are computationally unsolvable.

- Reduction:
  - Convert one problem to another,
  - The solution to the 2nd is usable to the 1st.
  - If problem A reduces to problem B, then
  - We can use solution to B to solve A.

Reducibility

- Example:
  - A: Find your way in the city.
  - B: But a map.

- Example:
  - A: Find the area of a rectangle,
  - B: Find the length and width of the rectangle.

- Example:
  - A: Solve a system of linear equations,
  - B: Invert a matrix.

Reducibility

- We can classify problems with reduction:
  - A is reducible to B: Solving A is not harder than solving B.

- Decidability:
  - A is reducible to B:

    If B is decidable, then A is decidable.

  - A is reducible to B:

    If A is undecidable, then B is undecidable.

- Complexity.

Reducibility

- We can classify problems with reduction:
  - A is reducible to B: Solving A is not harder than solving B.

- Decidability:
  - A is reducible to B:

    If B is decidable, then A is decidable.

  - A is reducible to B:

    If A is undecidable, then B is undecidable.

- Complexity.

Reducibility

- The halting problem:

$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}.$$

Theorem

$HALT_{TM}$ is undecidable.

- Proof Idea: By contradiction.
  - Assume that $HALT_{TM}$ is decidable. I.e., there is TM R that decides $HALT_{TM}$.
  - Construct S, a TM that decides A_{TM}.
    - Given an input of the form $\langle M, w\rangle$,
    - Give $\langle M, w\rangle$ to R. Since R is a decider it always halts.
      - If R rejects the input, M rejects w by looping. So, S rejects $\langle M, w\rangle$.
      - If R accepts the input, M halts on w. So, simulate M on w and report its result.
  - A contradiction, as A_{TM} can not have a decider.

Reducibility

- The halting problem:

$$HALT_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}.$$

Theorem

$HALT_{TM}$ is undecidable.

- Proof: By contradiction.

```
S = "On input ⟨M, w⟩, an encoding of a TM M and a string w:
        1. Run TM R on input ⟨M, w⟩.
        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."
```

Reducibility

- The emptiness problem:

$$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.$$

Theorem
E_{TM} is undecidable.

- Proof Idea: By contradiction.
  - Let R be a TM that decides E_{TM}.
  - We want to build a TM S that decides A_{TM}.
  - On input $\langle M, w\rangle$, modify $\langle M\rangle$ to only accept w.
  - Call R on the modified machine. If its language is empty, then M rejects w. Otherwise, M accepts w.

Reducibility

- The emptiness problem:

$$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.$$

Theorem
E_{TM} is undecidable.

- Proof: By contradiction.
  - Let R be a TM that decides E_{TM}.
  - We want to build a TM S that decides A_{TM}.
  - On input $\langle M, w\rangle$, modify $\langle M\rangle$ to only accept w. Call it M1:
    - Need some new states and some checks to test x=w.

```
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."
```

Reducibility

- The emptiness problem:

$$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.$$

Theorem
E_{TM} is undecidable.

- Proof: By contradiction.
  - Let R be a TM that decides E_{TM}.
  - We want to build a TM S that decides A_{TM}.
  - On input $\langle M, w\rangle$, modify $\langle M\rangle$ to only accept w. Call it M1.
  - Put things together:

```
S = "On input ⟨M, w⟩, 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 ⟨M_1⟩.
    3. If R accepts, reject; if R rejects, accept."
```

Reducibility

- Is the language of a TM regular?

$REGULAR_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}.$

Theorem
$REGULAR_{TM}$ is undecidable.

- Proof: By contradiction.
  - Let R be a TM that decides $REGULAR_{TM}$.
  - We want to build a TM S that decides A_{TM}.
  - On input $\langle M, w\rangle$, modify $\langle M\rangle$ to recognize a regular language if M accepts w. Call it M2. Otherwise, M2 recognize a nonregular language:
    - Regular language: Σ^*
    - Nonregular language: $\{0^n1^n \mid n \geq 0\}$

Reducibility

- Is the language of a TM regular?

$REGULAR_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}.$

Theorem
$REGULAR_{TM}$ is undecidable.

- Proof: By contradiction.
  - Let R be a TM that decides $REGULAR_{TM}$.
  - We want to build a TM S that decides A_{TM}.
  - On input $\langle M, w\rangle$, modify $\langle M\rangle$ to recognize a regular language if M accepts w. Call it M2. Otherwise, M2 recognize a nonregular language:
    - Regular language: Σ^*
    - Nonregular language: $\{0^n1^n \mid n \geq 0\}$

Accept all strings in $\{0^n1^n \mid n \geq 0\}$. If w was accepted, accept all other strings.

Reducibility

- Is the language of a TM regular?

$REGULAR_{\text{TM}} = \{\langle M\rangle |\ M$ is a TM and $L(M)$ is a regular language$\}$.

Theorem

$REGULAR_{TM}$ is undecidable.

- Proof: By contradiction.

```
S = "On input ⟨M, w⟩, 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 ⟨M_2⟩.
    3. If R accepts, accept; if R rejects, reject."
```

Reducibility

- Similar to $REGULAR_{TM}$, the problems of testing whether the language of a Turing machine is:
  - a context-free language,
  - a decidable language,
  - a finite language
- are undecidable.

Rice's Theorem

Determining any property of the languages recognized by Turing machines is undecidable.

Reducibility

- The equivalency problem:

$$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}.$$

Theorem
EQ_{TM} is undecidable.

- Proof: By contradiction.
  - Let R be a TM that decides EQ_{TM}.
  - We want to build a TM S that decides $\boldsymbol{E_{TM}}$.
  - On input $\langle M\rangle$, build M1 that rejects all strings and check their equivalency.

```
S = "On input ⟨M⟩, where M is a TM:
    1. Run R on input ⟨M, M_1⟩, where M_1 is a TM that rejects all
       inputs.
    2. If R accepts, accept; if R rejects, reject."
```

Reducibility

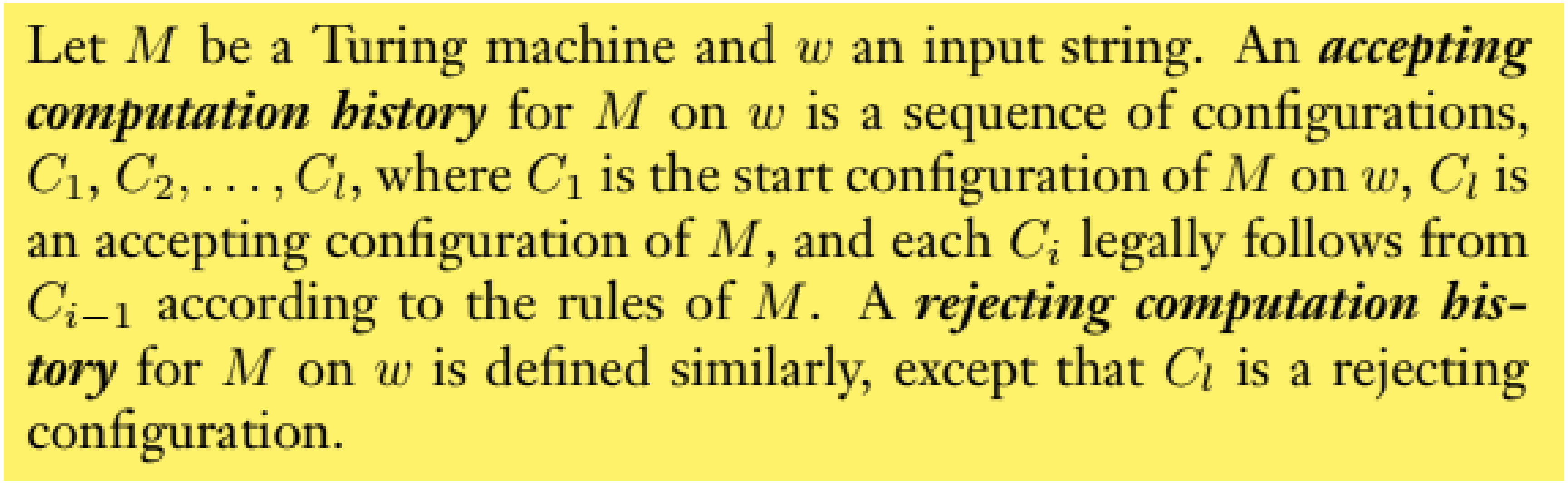
Computation Histories

- An important technique to reduce languages,
  - Often useful when the language involves testing for the existence of something:
    - The existence of integral roots in a polynomial.

- The computation history for a Turing machine on an input is simply the sequence of configurations that the machine goes through as it processes the input.

Reducibility

Computation Histories



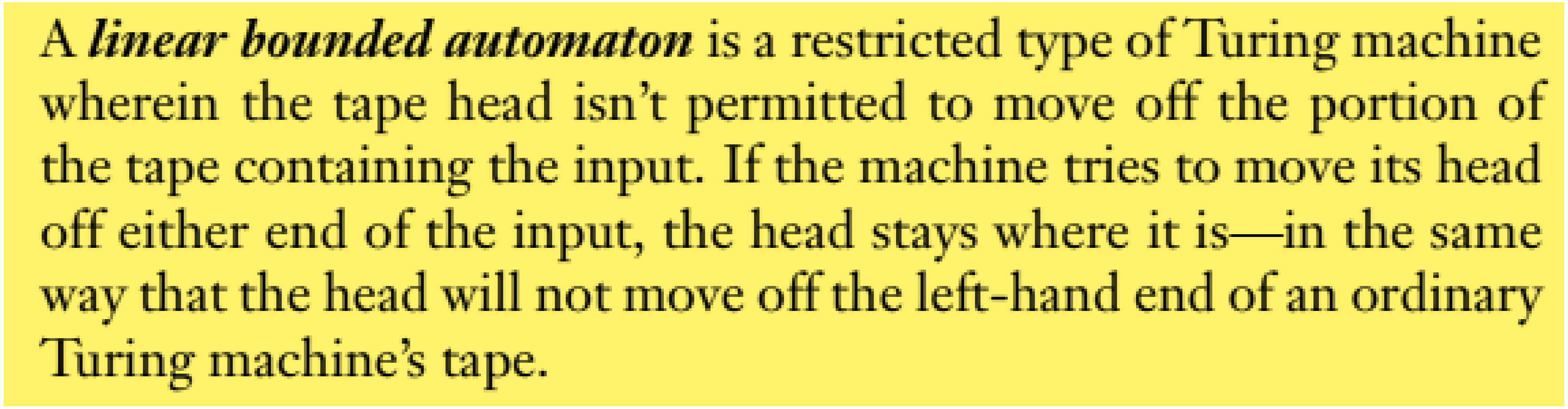
Let M be a Turing machine and w an input string. An ***accepting computation history*** for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_l$, where C_1 is the start configuration of M on w, C_l is an accepting configuration of M, and each C_i legally follows from C_{i-1} according to the rules of M. A ***rejecting computation history*** for M on w is defined similarly, except that C_l is a rejecting configuration.

- Computation histories are **finite** sequences.
  - If M doesn't halt on w, no accepting or rejecting computation history exists for M on w.
- Deterministic machines have at most **one computation history** on any given input. Nondeterministic machines may have **many computation histories** on a single input, corresponding to the various computation branches.

Reducibility

Computation Histories

- A Turing machine with a limited amount of memory:



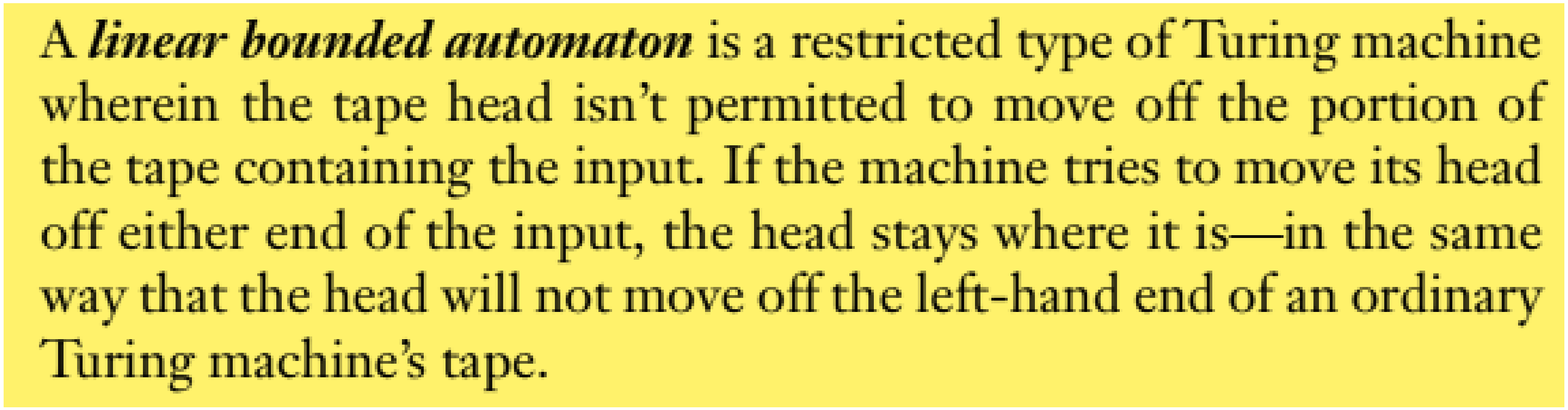
A ***linear bounded automaton*** is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not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

- Tape has sufficient squares to store the input.
- With the tape alphabet increase the memory up to a constant factor.
- We say: For an input of length n, the amount of memory available is linear in n.

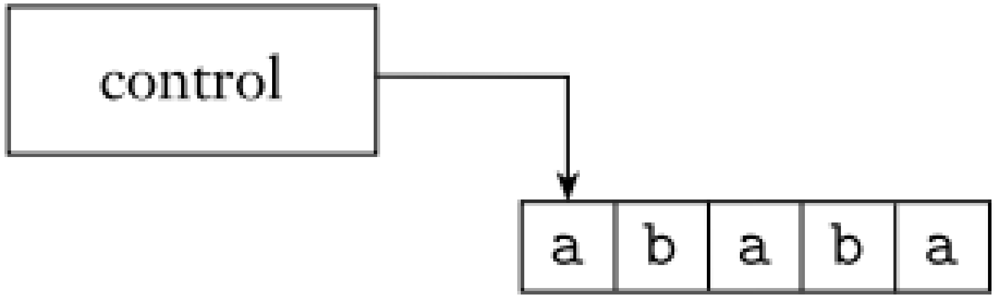
Reducibility

Computation Histories

- A Turing machine with a limited amount of memory:



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.



Reducibility
Computation Histories

- Linear Bounded Automata (LBA):
  - Can decide:
    - A_{DFA}
    - A_{CFG}
    - E_{DFA}
    - E_{CFG}
    - CFLs

Reducibility

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.

- Proof:
  - A configuration = A snapshot in the middle of its computation.
  - A configuration consists of the state of the control, position of the head, and contents of the tape.
    - M has q states.
    - The length of its tape is n, so the head can be in one of n positions, and
    - g^n possible strings of tape symbols appear on the tape.
  - The product of these three quantities is the total number of different configurations of M with a tape of length n.

Reducibility

- The acceptance problem for LBAs:

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

Theorem

A_{LBA} is **decidable**.

- Proof Idea: To decide whether LBA M accepts input w:
  - If M ever repeats a configuration, it would go on a loop.
  - M can be in only a limited number of configurations.
  - Simulate M for the maximum number of steps qng^n.
    - If M has not halted, reject.

Reducibility

- The acceptance problem for LBAs:

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

Theorem
A_{LBA} is **decidable**.

- Proof: To decide whether LBA M accepts input w:

L = "On input $\langle M, w\rangle$, where M is an LBA and w is a string:

```
    1. Simulate $M$ on $w$ for $qng^n$ steps or until it halts.
    2. If $M$ has halted, *accept* if it has accepted and *reject* if it has
       rejected. If it has not halted, *reject*."
```

Reducibility

- The emptiness problem for LBAs:

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

Theorem
E_{LBA} is undecidable.

- Proof Idea: By reduction from A_{TM}.
  - On input $\langle M, w\rangle$, construct LBA B:
    - B recognizes the language comprises all accepting computation histories for M on w.
    - If M accepts w $\rightarrow$ The language contains one string,
    - If M does not accept w $\rightarrow$ The language is empty.
    - Check whether the language of B is empty.

Reducibility

- The emptiness problem for LBAs:

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

Theorem
E_{LBA} is undecidable.

- Proof Idea: By reduction from A_{TM}.
  - How to construct B from M and w?
  - B should accept a string x if x is an accepting computation history for M on w.
  - This string can be of the following form:

$$\#\underbrace{\qquad\qquad}_{C_1}\#\underbrace{\qquad\qquad}_{C_2}\#\underbrace{\qquad\qquad}_{C_3}\#\ \cdots\ \#\underbrace{\qquad\qquad}_{C_l}\#$$

Reducibility

- The emptiness problem for LBAs:

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

Theorem
E_{LBA} is undecidable.

- Proof Idea: By reduction from A_{TM}.
  - How to construct B from M and w? Check the following:

$$\# \underbrace{\qquad\qquad}_{C_1} \# \underbrace{\qquad\qquad}_{C_2} \# \underbrace{\qquad\qquad}_{C_3} \# \quad \cdots \quad \# \underbrace{\qquad\qquad}_{C_l} \#$$

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.

Reducibility

- The emptiness problem for LBAs:

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

Theorem
ALL_{CFG} is undecidable.

- Proof: By contradiction:
  - Assume that ALL_{CFG} is decidable,
  - Show that A_{TM} is decidable.
  - On input $\langle M, w\rangle$, construct a CFG G:
    - If M does not accept w, G generates all strings,
    - If M accepts w, G does not generate the accepting computation history for M on w.

Reducibility

- CFG G:
  - If M does not accept w, G generates all strings,
  - If M accepts w, G does not generate the accepting computation history for M on w.
- An accepting computation history:

$$\#C_1\#C_2\#\cdots\#C_l\#,$$

- Ci : The config. on the ith step of computation on w.
- Generate all strings:

1. that *do not* start with C_1,
2. that *do not* end with an accepting configuration, or
3. in which some C_i *does not* properly yield C_{i+1} under the rules of M.

Reducibility

- CFG G:
  - If M does not accept w, G generates all strings,
  - If M accepts w, G does not generate the accepting computation history for M on w.
- An accepting computation history:

$$\#C_1\#C_2\#\cdots\#C_l\#,$$

- Ci : The config. on the ith step of computation on w.
- Generate all strings:

If M does not accept w, all strings satisfy one of these conditions and are generated by G.

1. that *do not* start with C_1,
2. that *do not* end with an accepting configuration, or
3. in which some C_i *does not* properly yield C_{i+1} under the rules of M.

Reducibility

- CFG G:
  - If M does not accept w, G generates all strings,
  - If M accepts w, G does not generate the accepting computation history for M on w.
- Build a PDA D instead and convert it to a CFG G.
  - Nondeterministically check the conditions.
- 1- Check the first configuration and accept if C1 is not a correct initial configuration.
- 2- Check the last configuration and accept if it is not a correct accepting configuration.
- 3- Select Ci Nondeterministically and compare with Ci+1. Accept if found a mismatch or improper update.

Reducibility

- CFG G:
  - If M does not accept w, G generates all strings,
  - If M accepts w, G does not generate the accepting computation history for M on w.
- Build a PDA D instead and convert it to a CFG G.
  - Nondeterministically check the conditions.
- 3- Select Ci Nondeterministically and compare with Ci+1. Accept if found a mismatch or improper update.
  - Comparison needs Ci+1 to be in reverse order:

$$\# \underbrace{\quad\longrightarrow\quad}_{C_1} \# \underbrace{\quad\longleftarrow\quad}_{C_2^{\mathcal{R}}} \# \underbrace{\quad\longrightarrow\quad}_{C_3} \# \underbrace{\quad\longleftarrow\quad}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\qquad}_{C_l} \#$$