Foundations of Computing

Lecture 14

Arkady Yerukhimovich

March 7, 2023

Outline

Lecture 13 Review

- More Turing Machines
- Turing Machine Variants
  - Multi-tape Turing Machines
  - Non-deterministic Turing Machines

Outline

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ
- Can use multiple tapes if it's useful

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ
- Can use multiple tapes if it's useful
- Can give a machine as an input to another machine
  - All machines we have seen can be written as finite tuples, e.g. $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
  - So, we can write this as a string and pass it to a TM
  - TM can then run the machine from this description

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ
- Can use multiple tapes if it's useful
- Can give a machine as an input to another machine
  - All machines we have seen can be written as finite tuples, e.g. $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
  - So, we can write this as a string and pass it to a TM
  - TM can then run the machine from this description
  - A TM that accepts any TM and runs it is called a *universal TM*

Specification of a Turing Machine

There are several levels of detail for specifying a TM

1. Full specification
    - Give full detail of transition function δ
    - This is very tedious

Specification of a Turing Machine

There are several levels of detail for specifying a TM

1. Full specification
    - Give full detail of transition function δ
    - This is very tedious
2. Turing Machine Algorithm specification
    - Explain algorithmically what happens on the tape
    - For example, scan the tape until you find a $\#$, zig-zag on the tape, etc.
    - Don't bother specifying a DFA for the control state

Specification of a Turing Machine

There are several levels of detail for specifying a TM

1. Full specification
   - Give full detail of transition function δ
   - This is very tedious
2. Turing Machine Algorithm specification
   - Explain algorithmically what happens on the tape
   - For example, scan the tape until you find a $\#$, zig-zag on the tape, etc.
   - Don't bother specifying a DFA for the control state
3. Algorithm specification
   - Give algorithm in pseudocode
   - Don't explicitly spell out what happens on the tape

Outline

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L
- That is, given a string w, an algorithm (aka. a Turing Machine) should tell us whether $w \in L$ or not.

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L
- That is, given a string w, an algorithm (aka. a Turing Machine) should tell us whether $w \in L$ or not.

Question(s)

- Can all languages be computed in this way?

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L
- That is, given a string w, an algorithm (aka. a Turing Machine) should tell us whether $w \in L$ or not.

Question(s)

- Can all languages be computed in this way?
- Are there some problems that inherently do not have any algorithmic solution?

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepting those in L and rejecting those not in L

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepting those in L and rejecting those not in L
- Seems to match informal definition we wanted before

Definition: Turing-recognizable languages

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepting those in L and rejecting those not in L
- Seems to match informal definition we wanted before

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepting those in L and rejecting those not in L
- Seems to match informal definition we wanted before

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepting those in L and rejecting those not in L
- Seems to match informal definition we wanted before

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepting those in L and rejecting those not in L
- Seems to match informal definition we wanted before

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Observation

Every Decidable language is also Turing-recognizable, but the reverse direction may not be true

The Big Question

The Question

Are there problems that are undecidable?

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm
- If you believe Church-Turing thesis, it cannot be solved by any computer

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm
- If you believe Church-Turing thesis, it cannot be solved by any computer
- We will see that even relatively natural problems can be undecidable

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm
- If you believe Church-Turing thesis, it cannot be solved by any computer
- We will see that even relatively natural problems can be undecidable

A Second Question

What about Turing-unrecognizable languages?

Outline

Taking Machines as Input

- Recall that we have defined machines as tuples:

Taking Machines as Input

- Recall that we have defined machines as tuples:
  1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
  2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  3. TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

Taking Machines as Input

- Recall that we have defined machines as tuples:
  1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
  2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  3. TM $M = (Q, \Sigma\Gamma, \delta, q_0, q_{accept}, q_{reject})$
- This means that any such machine can be written down as a finite length string

Taking Machines as Input

- Recall that we have defined machines as tuples:
  1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
  2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  3. TM $M = (Q, \Sigma\Gamma, \delta, q_0, q_{accept}, q_{reject})$
- This means that any such machine can be written down as a finite length string
- So, can give a description of a machine M to another machine M'

Taking Machines as Input

- Recall that we have defined machines as tuples:
  1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
  2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  3. TM $M = (Q, \Sigma\Gamma, \delta, q_0, q_{accept}, q_{reject})$
- This means that any such machine can be written down as a finite length string
- So, can give a description of a machine M to another machine M'
- Today, we will talk about TM's that run another machine M'

Problems About Regular Languages

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

Problems About Regular Languages

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

Algorithm to decide L_{DFA}:
On input $\langle B, w\rangle$

Problems About Regular Languages

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

Algorithm to decide L_{DFA}:
On input $\langle B, w\rangle$

1. Simulate B on input w

Problems About Regular Languages

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

Algorithm to decide L_{DFA}:
On input $\langle B, w\rangle$

1. Simulate B on input w
2. If simulation ends in an accept, then accept. If it ends in a non-accepting state, then reject

Problems About Regular Languages

$$L_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Problems About Regular Languages

$$L_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Algorithm to decide L_{NFA}:
On input $\langle B, w\rangle$

Problems About Regular Languages

$$L_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Algorithm to decide L_{NFA}:
On input $\langle B, w\rangle$

1. Convert NFA B to equivalent DFA C

Problems About Regular Languages

$$L_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Algorithm to decide L_{NFA}:
On input $\langle B, w\rangle$

1. Convert NFA B to equivalent DFA C
2. Run TM from previous slide on input $\langle C, w\rangle$
3. Output what this TM outputs

Problems About Regular Languages

$L_{REX} = \{\langle R, w \rangle \mid R \text{ is a reg. exp. that generates the string } w\}$

Problems About Regular Languages

$$L_{REX} = \{\langle R, w\rangle \mid R \text{ is a reg. exp. that generates the string } w\}$$

Algorithm to decide L_{REX}:
On input $\langle R, w\rangle$

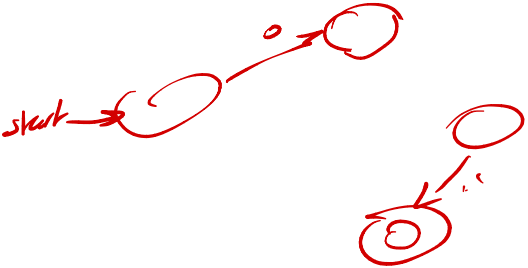
1. Convert R to DFA A

2. Run Algorithm to decide $L(A)$

Problems About Regular Languages

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

A :



Problems About Regular Languages

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

Intuition:

- Remember, for a DFA to accept some string there must be a path from start state to accept state

Problems About Regular Languages

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

Intuition:

- Remember, for a DFA to accept some string there must be a path from start state to accept state
- We need to figure out if such a path exists

Problems About Regular Languages

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

Intuition:

- Remember, for a DFA to accept some string there must be a path from start state to accept state
- We need to figure out if such a path exists

Algorithm to decide $L_{\emptyset-DFA}$:
On input $\langle A \rangle$

Problems About Regular Languages

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

Intuition:

- Remember, for a DFA to accept some string there must be a path from start state to accept state
- We need to figure out if such a path exists

Algorithm to decide $L_{\emptyset-DFA}$:
On input $\langle A \rangle$

1. Mark the start state of A

Problems About Regular Languages

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

Intuition:

- Remember, for a DFA to accept some string there must be a path from start state to accept state
- We need to figure out if such a path exists

Algorithm to decide $L_{\emptyset-DFA}$:
On input $\langle A\rangle$

1. Mark the start state of A
2. Repeat until no new states get marked:
    - Mark any state that has an incoming transition from any state already marked

Problems About Regular Languages

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

Intuition:

- Remember, for a DFA to accept some string there must be a path from start state to accept state
- We need to figure out if such a path exists

Algorithm to decide $L_{\emptyset-DFA}$:
On input $\langle A \rangle$

1. Mark the start state of A
2. Repeat until no new states get marked:
   - Mark any state that has an incoming transition from any state already marked
3. If no accept state is marked, accept, else, reject

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

$$L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C \rangle$ to decide this language

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in L(A)$ s.t. $x \notin L(B)$

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in \underline{L(A)}$ s.t. $x \notin L(B)$
  $L(A) \setminus L(B) = L(A) \cap \overline{L(B)}$ contains all such items

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C \rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in \underline{L(A)}$ s.t. $x \notin L(B)$
  $L(A) \setminus L(B) = L(A) \cap \overline{L(B)}$ contains all such items
- Need to also consider items in $L(B)$ that are not in $L(A)$

Problems About Regular Languages

$$L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in L(A)$ s.t. $x \notin L(B)$
  $L(A) \setminus L(B) = L(A) \cap \overline{L(B)}$ contains all such items
- Need to also consider items in $L(B)$ that are not in $L(A)$

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