UMB CS420

Nondeterministic TMs

Tuesday, November 1, 2022
Announcements

• HW 6 in
  • Due Sun 10/30 11:59pm EST

• HW 7 out
  • Due Sun 11/6 11:59pm EST
Last Time: Turing Machines

- **Turing Machines** can read and write to arbitrary “tape” cells
  - Tape initially contains input string

- The tape is infinite
  - (to the right)

- On a transition, “head” can move left or right 1 step

Call a language **Turing-recognizable** if some Turing machine recognizes it.
Turing Machine: High-Level Description

- $M_1$ accepts if input is in language $B = \{w#w \mid w \in \{0,1\}^*\}$

$M_1 = \text{“On input string } w:\text{”}$

1. Zig-zag across the tape, crossing over symbol positions on either side of the # symbol. Instead of the # symbol, write the same symbol. Cross off symbols as you go, to keep track of which symbols correspond.

2. When all symbols to the left of the # symbol are crossed off, check for any remaining symbols on the tape. If no symbols remain, reject; otherwise, accept.

We will (mostly) stick to informal descriptions of Turing machines, like this one.

But it must always correspond to some precise formal description.

Analogy: High-level (e.g., Python) function definitions vs assembly language.
A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $Q$, $\Sigma$, $\Gamma$ are all finite sets and

1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet not containing the blank symbol $\_,$
3. $\Gamma$ is the tape alphabet, where $\_ \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$. 

(read, write, move)
Flashback: DFAs vs NFAs

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the states,
2. $\Sigma$ is a finite set called the alphabet,
3. $\delta: Q \times \Sigma \rightarrow Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set of states,
2. $\Sigma$ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Nondeterministic transition produces set of possible next states.
A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet not containing the blank symbol \(\_\),
3. \(\Gamma\) is the tape alphabet, where \(\_ \in \Gamma\) and \(\Sigma \subseteq \Gamma\),
4. \(\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}\) is the transition function,
5. \(q_0 \in Q\) is the start state,
6. \(q_{\text{accept}} \in Q\) is the accept state, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{reject}} \neq q_{\text{accept}}\).
A **nondeterministic Turing machine** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $Q, \Sigma, \Gamma$ are all finite sets and

1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet not containing the **blank symbol** $\sqcup$,
3. $\Gamma$ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$, where $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$. 
Thm: Deterministic TM $\iff$ Non-det. TM

$\Rightarrow$ If a deterministic TM recognizes a language, then a non-deterministic TM recognizes the language

- **Convert:** Deterministic TM $\rightarrow$ Non-deterministic TM ...
- ... change Deterministic TM $\delta$ fn output to a one-element set
  - (just like conversion of DFA to NFA --- HW 2, Problem 2)
- **DONE!**

$\Leftarrow$ If a non-deterministic TM recognizes a language, then a deterministic TM recognizes the language

- **Convert:** Non-deterministic TM $\rightarrow$ Deterministic TM ...
- ... ???
Review: Nondeterminism

Deterministic computation

- start
- ...
- accept or reject

Nondeterministic computation

- start
- ...
- reject
- ...

In nondeterministic computation, every step can branch into a set of “states”

What is a “state” for a TM?

$$\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$
Flashback: PDA Configurations (IDs)

• A configuration (or ID) is a “snapshot” of a PDA’s computation

• 3 components \((q, w, \gamma)\):
  - \(q\) = the current state
  - \(w\) = the remaining input string
  - \(\gamma\) = the stack contents

A sequence of configurations represents a PDA computation
TM Configuration (ID) = ???

1) states

control

3) read/write head

a b a b □ □ □...

2) Tape contents

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet not containing the blank symbol \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) and \(\Sigma \subseteq \Gamma\),
4. \(\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}\) is the transition function,
5. \(q_0 \in Q\) is the start state,
6. \(q_{\text{accept}} \in Q\) is the accept state, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{reject}} \neq q_{\text{accept}}\).
TM Configuration = State + Head + Tape

States

Starting configuration

Config after 1 step

Config after 2 steps

accept
TM Configuration = State + Head + Tape

Textual representation of “configuration” (use this in HW)

1st char after state is current head position
TM Computation, Formally

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}) \]

**Single-step**

(Right)

\[ \alpha q_1 a \beta \vdash \alpha x q_2 \beta \]

if \( q_1, q_2 \in Q \)
\[ \delta(q_1, a) = (q_2, x, R) \]
\( a, x \in \Gamma \) \( \alpha, \beta \in \Gamma^* \)

write

(Left)

\[ \alpha b q_1 a \beta \vdash \alpha q_2 b x \beta \]

if \( \delta(q_1, a) = (q_2, x, L) \)

**Extended**

- **Base Case**
  \[ I \vdash^* I \text{ for any ID } I \]

- **Recursive Case**
  \[ I \vdash^* J \text{ if there exists some ID } K \]
  such that \( I \vdash K \) and \( K \vdash^* J \)

**Edge cases:**

- Head stays at leftmost cell
  \[ q_1 a \beta \vdash q_2 x \beta \]
  if \( \delta(q_1, a) = (q_2, x, L) \)

- Add blank symbol to config
  \[ \alpha q_1 \vdash \alpha \beta q_2 \]
  if \( \delta(q_1, \cdot) = (q_2, \cdot, R) \)

(L move, when already at leftmost cell)

(R move, when at rightmost filled cell)
Nondeterminism in TMs

For TMs, each node is a configuration
Nondeterministic TM $\rightarrow$ Deterministic

1st way

- Simulate NTM with Det. TM:
  - Det. TM keeps multiple configs single tape
    - Like how single-tape TM simulates multi-tape
  - Then run all computations, **concurrently**
    - I.e., 1 step on one config, 1 step on the next, ...
- Accept if any accepting config is found
- **Important:**
  - Why must we step configs **concurrently**?
### Interlude: Running TMs inside other TMs

If TMs are function definitions, then they can be *called* like functions...

**Exercise:**
- Given: TMs $M_1$ and $M_2$
- Create: TM $M$ that accepts if either $M_1$ or $M_2$ accept

**Possible solution #1:**

$M$ = on input $x$,
1. Call $M_1$ with arg $x$; accept if $M_1$ accepts
2. Call $M_2$ with arg $x$; accept if $M_2$ accepts

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>reject</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>accept</td>
<td>reject</td>
<td>accept</td>
</tr>
</tbody>
</table>

**Note:** This solution would be ok if we knew $M_1$ and $M_2$ were *deciders* (which halt on all inputs)

"loop" means input string not accepted
Interlude: Running TMs inside other TMs

If TMs are function definitions, then they can be *called* like functions ...

... with concurrency!

Exercise:
- Given: TMs $M_1$ and $M_2$
- Create: TM $M$ that accepts if *either* $M_1$ or $M_2$ accept

Possible solution #1:
$M = \text{on input } x$,
1. Call $M_1$ with arg $x$; accept if $M_1$ accepts
2. Call $M_2$ with arg $x$; accept if $M_2$ accepts

Possible solution #2:
$M = \text{on input } x$,
1. Call $M_1$ and $M_2$ with $x$ *concurrently*, i.e.,
   a) Run $M_1$ with $x$ for 1 step; accept if $M_1$ accepts
   b) Run $M_2$ with $x$ for 1 step; accept if $M_2$ accepts
   c) Repeat

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>reject</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>accept</td>
<td>reject</td>
<td>accept</td>
</tr>
<tr>
<td>accept</td>
<td>loops</td>
<td>accept</td>
</tr>
<tr>
<td>loops</td>
<td>accept</td>
<td>loops</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>reject</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>accept</td>
<td>reject</td>
<td>accept</td>
</tr>
<tr>
<td>accept</td>
<td>loops</td>
<td>accept</td>
</tr>
<tr>
<td>loops</td>
<td>accept</td>
<td>accept</td>
</tr>
</tbody>
</table>
Nondeterministic TM $\rightarrow$ Deterministic

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Check all tree paths (in breadth-first order)
    - 1
    - 1-1

2nd way
(Sipser)
Nondeterministic TM → Deterministic

• Simulate NTM with Det. TM:
  • Number the nodes at each step
  • Check all tree paths (in breadth-first order)
    • 1
    • 1-1
    • 1-2

2nd way (Sipser)
Nondeterministic TM $\rightarrow$ Deterministic

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Check all tree paths (in breadth-first order)
    - 1
    - 1-1
    - 1-2
    - 1-1-1
Nondeterministic TM $\Rightarrow$ Deterministic

Always has input, never changes

“Work tape” when checking each path (re-copy input here each time)

Tracks which node we are on, e.g., 1-1-2, etc.

Needs 3 tapes

2nd way (Sipser)

$D$

input tape

simulation tape

address tape
Nondeterministic TM $\Leftrightarrow$ Deterministic TM

$\checkmark \Rightarrow$ If a deterministic TM recognizes a language, then a nondeterministic TM recognizes the language
  • Convert Deterministic TM $\rightarrow$ Non-deterministic TM

$\checkmark \Leftarrow$ If a nondeterministic TM recognizes a language, then a deterministic TM recognizes the language
  • Convert Nondeterministic TM $\rightarrow$ Deterministic TM
Conclusion: These are All Equivalent TMs!

• Single-tape Turing Machine

• Multi-tape Turing Machine

• Non-deterministic Turing Machine
Turing Machines and Algorithms

• **Turing Machines** can express any “computation”
  • i.e., a Turing Machine models (Python, Java) programs (functions)!

• 2 classes of Turing Machines
  • **Recognizers** may loop forever
  • **Deciders** always halt

• **Deciders = Algorithms**
  • i.e., an algorithm is any program that always halts

Remember: TMs = programs
Flashback: HW 1, Problem 1

1. Come up with a formal description for this DFA. Recall that a DFA’s formal description has five components: $M = (Q, \Sigma, \delta, q_{start}, F)$. You may assume that the alphabet contains only the symbols from the diagram.

2. Then for each of the following, say whether the computation represents an accepting computation or not (make sure to review the definition of an accepting computation). If the answer is no, explain why not:
   a. $\delta(q_1, UUMB)$
   b. $\delta(q_1, UMBM)$
   c. $\delta(q_2, UMBB)$
   d. $\delta(q_3, e)$
   e. $\delta(q_3, UMASSBOSTON)$

To “figure out” this computation ... you had to “do” (meta) computations (e.g., in your head)
Flashback: DFA Computations

Define the extended transition function: \( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \)

**Base case:** \( \hat{\delta}(q, \epsilon) = q \)

**Recursive case:** 
\[
\hat{\delta}(q, a_1 w_{rest}) = \hat{\delta}(\delta(q, a_1), w_{rest})
\]

Calculating this computation requires (meta) computation!

Could you implement this (meta) computation as an algorithm?

1) Define “current” state \( q_{current} = \) start state \( q_0 \)
2) For each input char \( a_i \)...
   a) Define \( q_{next} = \delta(q_{current}, a_i) \)
   b) Set \( q_{current} = q_{next} \)
3) Return TRUE if \( q_{current} \) is an accept state

Remember: TMs = programs
The language of **DFAaccepts**

\[ A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \} \]

But a language is a set of strings?
Interlude: Encoding Things into Strings

• Definition: A Turing machine’s input is always a string

• But: A TM (program)’s input could also be a list, graph, DFA, ...?
• Solution: anything used as TM input must be encoded as string

Notation: \(<\text{SOMETHING}>\) = string encoding for SOMETHING

• A tuple combines multiple encodings, e.g., \(<B, w>\) (from prev slide)

Example: Possible string encoding for a DFA?

But in this class, we don’t care about what the encoding is! (Just that there is one)

\((Q, \Sigma, \delta, q_0, F)\) (written as string)
Interlude: High-Level TMs and Encodings

A high-level TM description:

1. Doesn’t need to describe exactly how input string is encoded
2. Assumes input is a “valid” encoding
   • Invalid encodings are implicitly rejected
The language of $\text{DFAaccepts}$

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

- $\text{DFAaccepts}$ is a Turing machine
- But is it a decider or recognizer?
  - I.e., is it an algorithm?
- To show it’s an algo, need to prove:
  
  $A_{\text{DFA}}$ is a decidable language
How to prove that a language is decidable?

• Create a Turing machine that **decides** that language!

**Remember:**

• **A decider** is Turing Machine that always halts
  • I.e., for any input, it either accepts or rejects it.
  • It must never go into an infinite loop
How to Design Deciders

• If TMs = Programs ...
  … then Creating a TM = Programming

• E.g., if HW asks “Show that lang $L$ is decidable” ...
  • .. you must create a TM that decides $L$; to do this ...
  • … think of how to write a (halting) program that does what you want
Next Time: $A_{DFA}$ is a decidable language

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

Decider for $A_{DFA}$:
Check-in Quiz 11/1

On gradescope