UMB CS420

Nondeterministic TMs

Monday, March 7, 2022

This, for any nondeterministic Turing machine M that runs in some polynomial time, we can devise an algorithm that takes an input w of length n and produces M_w. The running time is O(n^k) on a multitape deterministic Turing machine and...
Announcements

• HW 5 in

• HW 6 out
  • Due Sun 3/20 11:59pm EST (2 weeks)

• Reminder: No class next week (Spring Break)
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: Informal Description

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

\( M_1 = \) “On input string \( w \):

1. Zig-zag across the tape to corresponding positions on either side of the \# symbol to check whether these positions contain the same symbol. If they do not, or if no \# is found, reject. Cross off symbols as they are checked to keep track of which symbols correspond.

2. When all symbols to the left of the \# have been crossed off, check for any remaining symbols to the right of the \#. If any symbols remain, reject; otherwise, accept.”
A *Turing machine* is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet not containing the *blank symbol* \(\Box\),
3. \(\Gamma\) is the tape alphabet, where \(\Box \in \Gamma\) 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 \(q_{\text{reject}} \neq q_{\text{accept}}\).
Non-Deterministic Turing Machines?
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.
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 \(q_{\text{reject}} \neq q_{\text{accept}}\).
Nondeterministic Turing Machine Formal Definition

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. \(\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}\)\(\xrightarrow{\delta} Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})\)
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
  • To 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 3)
  • DONE!

$\Leftarrow$ If a non-deterministic TM recognizes a language, then a deterministic TM recognizes the language
  • To convert Non-deterministic TM $\rightarrow$ Deterministic TM ...
  • ... ???
Review: Nondeterminism

Deterministic computation

- start
- ...
- accept or reject

Nondeterministic computation

- ...
- 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

• A configuration (or ID) \((q, w, \gamma)\) has three components:
  \(q\) = the current state
  \(w\) = the remaining input string
  \(\gamma\) = the stack contents
A *Turing machine* is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where 
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
TM Configuration = State + Head + Tape

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

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

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

**Single-step**

(Right)

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

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

(Left)

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

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

**Edge cases:**

- Head stays at leftmost cell
- Add blank symbol to config

**Extended**

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

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

- (L move, when already at leftmost cell)
- (R move, when at rightmost filled cell)
Nondeterminism in TMs

Deterministic computation

- start
- ...
- accept or reject

Nondeterministic computation

- For TMs, each node is a configuration
- 1011q_r01111
- 1011q_r01111
- 1011q_r01111
- reject
- accept
Nondeterministic TM $\rightarrow$ Deterministic

• Simulate NTM with Det. TM:
  • Det. TM keeps multiple configs single tape
    • Like how single-tape TM simulates multi-tape
  • Then run all computations, in parallel
    • 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 in parallel?
Interlude: Running TMs inside other TMs

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 $x$,
  • Run $M_1$ on $x$, accept if $M_1$ accepts
  • Run $M_2$ on $x$, 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

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 $x$,
  - Run $M_1$ on $x$, accept if $M_1$ accepts
  - Run $M_2$ on $x$, accept if $M_2$ accepts

Possible solution #2:
- $M$ = on input $x$,
  - Run $M_1$ and $M_2$ on $x$ in parallel, i.e.,
    - Run $M_1$ on $x$ for 1 step, accept if $M_1$ accepts
    - Run $M_2$ on $x$ for 1 step, accept if $M_2$ accepts
    - Repeat

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M$</th>
</tr>
</thead>
</table>
| reject| accept| accept| ✓  
| accept| reject| accept| ✓  
| accept| loops | accept|     
| loops | accept| loops|     

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M$</th>
</tr>
</thead>
</table>
| reject| accept| accept| ✓  
| accept| reject| accept| ✓  
| accept| loops | accept| ✓  
| loops | accept| accept|     

59
Nondeterministic TM $\rightarrow$ Deterministic

• Simulate NTM with Det. TM:
  • Number the nodes at each step
  • Deterministically check every tree path, in breadth-first order
    • 1
    • 1-1
Nondeterministic TM $\Rightarrow$ Deterministic

• Simulate NTM with Det. TM:
  • Number the nodes at each step
  • Deterministically check every tree path, in breadth-first order
    • 1
    • 1-1
    • 1-2

2nd way
(Sipser)
Nondeterministic TM → Deterministic

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - 1-1-1
Nondeterministic TM $\rightarrow$ Deterministic

$D$

Always has input, never changes

0 0 1 0 \(\square\) ... input tape

# Used to run each path (re-copy input here when checking a path)

x x # 0 1 x \(\square\) ... simulation tape

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

1 2 3 3 2 3 1 2 1 1 3 \(\square\) ... address tape

Needs 3 tapes

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

$\checkmark$ $\Rightarrow$ If a deterministic TM recognizes a language, then a nondeterministic TM recognizes the language
  • To 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)

$\checkmark$ $\Leftarrow$ 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!

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

• Deciders = Algorithms
  • I.e., an algorithm is any program that always halts
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, e.g. 
\( M = (Q, \Sigma, \delta, q_0, F) \).

You may assume that the alphabet contains only the symbols from the diagram.

2. Then do the following computations using extended transition function and say whether computation represents an accepting computation (some of these may be tricky so be careful here, you may want to review the definition of an accepting computation):

a. \( \hat{\delta}(q_0, \varepsilon) \)

b. \( \hat{\delta}(q_0, a) \)
Flashback: DFA Computations

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

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

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

Calculating this computation requires (meta) computation!

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

A function: \( \text{DFAaccepts}(B, w) \) returns \text{TRUE} if DFA B accepts string \( w \)

- Define “current” state \( q_{\text{current}} = \text{start state } q_0 \)
- For each input char \( a_i \)...
  - Define \( q_{\text{next}} = \delta(q_{\text{current}}, a_i) \)
  - Set \( q_{\text{current}} = q_{\text{next}} \)
- Return \text{TRUE} if \( q_{\text{current}} \) is an accept state
The language of **DFAaccepts**

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

But a language is a set of strings?

Function $DFAaccepts(B, w)$ returns **true** if DFA $B$ accepts string $w$. 
Interlude: Encoding Things into Strings

- A Turing machine’s input is always a string
- So anything we want to give to TM must be encoded as string

Notation: \(<SOMETHING>\) = string encoding for SOMETHING
  - A tuple combines multiple encodings, e.g., \(<B, w>\) (from prev slide)

Example: Possible string encoding for a DFA?

Or: \((Q, \Sigma, \delta, q_0, F)\)
(written as string)
Interlude: Informal TMs and Encodings

An informal TM description:

1. Doesn’t need to describe exactly how input string is encoded
2. Assumes input is a “valid” encoding
   - Invalid encodings are automatically 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: \( A_{\text{DFA}} \) is a decidable language

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

Decider for \( A_{\text{DFA}} \) :
Check-in Quiz 3/7

On gradescope