Today we will learn...

- Recap exercises
- TM configuration and configuration history
- TM acceptance
- Variants of Turing Machines
  - Multi-tape
  - Nondeterministic

Section 3.1, 3.2, and 3.3

Supplementary material

- Professor Harry Porter's video
- Professor Dan Gusfield's video
- Turing Machines, Stanford Encyclopedia of Philosophy
Exercises
Exercise 1

Convert the following grammar into a PDA

\[
A \rightarrow 0A1 \mid B \\
B \rightarrow 1B \mid \epsilon
\]
Exercise 1

Convert the following grammar into a PDA

\[
\begin{align*}
A & \to 0A1 \mid B \\
B & \to 1B \mid \epsilon
\end{align*}
\]
Exercise 2

Show that if $L_1 \cup L_2$ is not context-free, then either $L_1$ is not context-free or $L_2$ is not context free.

Proof.
Exercise 2

Show that if $L_1 \cup L_2$ is not context-free, then either $L_1$ is not context-free or $L_2$ is not context free.

**Proof.**

1. We know that if $L_1$ is CF and $L_2$ is CF, then $L_1 \cup L_2$ is CF.
2. Apply the contrapositive to (1) and we conclude our proof.
Exercise 3

We know that $L_2 = \{ w \mid w = a^n b^n c^n \vee |w| \text{ is even} \}$ is not context free. Show that $L_3 = \{ a^n b^n c^n \mid n \geq 0 \}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

Proof.
Exercise 3

We know that $L_2 = \{w \mid w = a^n b^n c^n \lor |w| \text{ is even}\}$ is not context free.

Show that $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

Proof.

1. It is easy to see that $L_2 = L_3 \cup L_4$ where $L_4 = \{w \mid |w| \text{ is even}\}$. 
Exercise 3

We know that \( L_2 = \{ w \mid w = a^n b^n c^n \lor |w| \text{ is even} \} \) is not context free. Show that \( L_3 = \{ a^n b^n c^n \mid n \geq 0 \} \) is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

**Proof.**

1. It is easy to see that \( L_2 = L_3 \cup L_4 \) where \( L_4 = \{ w \mid |w| \text{ is even} \} \).
2. We apply the previous theorem of Exercise 1 and get that either \( L_3 \) is not context free, or \( L_4 \) is not-context free.
Exercise 3

We know that \( L_2 = \{ w \mid w = a^n b^n c^n \lor |w| \text{ is even} \} \) is not context free.

Show that \( L_3 = \{ a^n b^n c^n \mid n \geq 0 \} \) is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

**Proof.**

1. It is easy to see that \( L_2 = L_3 \cup L_4 \) where \( L_4 = \{ w \mid |w| \text{ is even} \} \).
2. We apply the previous theorem of Exercise 1 and get that either \( L_3 \) is not context free, or \( L_4 \) is not-context free.
3. But we know that \( L_4 \) is regular and therefore context-free.
Exercise 3

We know that $L_2 = \{ w \mid w = a^n b^n c^n \lor |w| \text{ is even} \}$ is not context free.

Show that $L_3 = \{ a^n b^n c^n \mid n \geq 0 \}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

**Proof.**

1. It is easy to see that $L_2 = L_3 \cup L_4$ where $L_4 = \{ w \mid |w| \text{ is even} \}$.

2. We apply the previous theorem of Exercise 1 and get that either $L_3$ is not context free, or $L_4$ is not-context free.

3. But we know that $L_4$ is regular and therefore context-free.

4. Thus, $L_2$ is not CF.
Turing Machine:
configuration & configuration history
Turing Machines

Definition 3.3

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

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

To ponder..

- What is the minimum number of states?
- Can the input and the tape alphabets be the same?
- Write a Turing machine with the minimum number of states that recognizes $\emptyset$
- Write a Turing machine with the minimum number of states that recognizes $\Sigma^*$
Configuration

A configuration is a snapshot of a computation. That is, it contains all information necessary to resume (or replay) a computation from any point in time.

A configuration consists of

- the tape
- the head of the tape
- the current state
Example 2

**State**  **Tape**

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>0011</td>
</tr>
<tr>
<td>$B$</td>
<td>X011</td>
</tr>
<tr>
<td>$B$</td>
<td>X011</td>
</tr>
<tr>
<td>$C$</td>
<td>X0Y1</td>
</tr>
</tbody>
</table>
Example 2

Example 2

Turing Machines
Lecture 19
Tiago Cogumbreiro

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0011</td>
</tr>
<tr>
<td>B</td>
<td>X011</td>
</tr>
<tr>
<td>B</td>
<td>X011</td>
</tr>
<tr>
<td>C</td>
<td>X0Y1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>X0Y1</td>
</tr>
<tr>
<td>C</td>
<td>X0Y1</td>
</tr>
<tr>
<td>B</td>
<td>XXY1</td>
</tr>
<tr>
<td>B</td>
<td>XXY1</td>
</tr>
</tbody>
</table>
Example 2

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>0011</td>
</tr>
<tr>
<td>$B$</td>
<td>X011</td>
</tr>
<tr>
<td>$B$</td>
<td>X011</td>
</tr>
<tr>
<td>$C$</td>
<td>X0Y1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>X0Y1</td>
</tr>
<tr>
<td>$S$</td>
<td>X0Y1</td>
</tr>
<tr>
<td>$B$</td>
<td>XXY1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>XXYY</td>
</tr>
<tr>
<td>$S$</td>
<td>XXYY</td>
</tr>
<tr>
<td>$D$</td>
<td>XXYY</td>
</tr>
</tbody>
</table>
Example 2

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0011</td>
</tr>
<tr>
<td>B</td>
<td>X011</td>
</tr>
<tr>
<td>B</td>
<td>X011</td>
</tr>
<tr>
<td>C</td>
<td>X0Y1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>X0Y1</td>
</tr>
<tr>
<td>C</td>
<td>X0Y1</td>
</tr>
<tr>
<td>B</td>
<td>X0Y1</td>
</tr>
<tr>
<td>B</td>
<td>XXY1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>XXYY</td>
</tr>
<tr>
<td>C</td>
<td>XXYY</td>
</tr>
<tr>
<td>S</td>
<td>XXYY</td>
</tr>
<tr>
<td>B</td>
<td>XXYY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>XXYY</td>
</tr>
</tbody>
</table>

Accept!
Simulate
Example 1 configuration

<table>
<thead>
<tr>
<th>State</th>
<th>Tape</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>01110</td>
<td>S 01110</td>
</tr>
<tr>
<td>$B$</td>
<td>_x1110</td>
<td>x B 1110</td>
</tr>
<tr>
<td>$B$</td>
<td>_xy110</td>
<td>xy B 110</td>
</tr>
<tr>
<td>$B$</td>
<td>_xyy10</td>
<td>xyy B 10</td>
</tr>
<tr>
<td>$B$</td>
<td>_xyyy0</td>
<td>xyyy B 0</td>
</tr>
<tr>
<td>$B$</td>
<td>_xyyyx</td>
<td>xyyyy B</td>
</tr>
</tbody>
</table>
The configuration history (sequence of configurations), describes all configurations from the initial state until a current state.

Definition

We say that $C_1$ yields $C_2$

### Example

<table>
<thead>
<tr>
<th>Configuration history</th>
</tr>
</thead>
<tbody>
<tr>
<td>S 01110</td>
</tr>
<tr>
<td>x B 1110</td>
</tr>
<tr>
<td>xy B 110</td>
</tr>
<tr>
<td>xyy B 10</td>
</tr>
<tr>
<td>xyyy B 0</td>
</tr>
<tr>
<td>xyyyyx B</td>
</tr>
</tbody>
</table>
Acceptance

A Turing machine

- **accepts** a string if there is a configuration history that reaches the accept state.
- **rejects** a string if there is a configuration history that reaches the reject state.
- **rejects** a string if it never reaches an accept or reject states
  This means that for any configuration of any length, there is no accept nor a reject state.

The acceptance algorithm

- **halts** when the machine is in an accept or reject state
  *This is different than NFAs/PDAs which can enter and leave the accept state.*
Exercise

Give a Turing Machine that recognizes words of an even length.

NFA

```
start → q1 → q2
```

Transitions:
- $a, b$ from $q_1$ to $q_2$
TM that recognizes words of an even length

(online)
Exercise

What language does this TM recognize? (online)
The Church-Turing thesis
Alan Turing and the Turing Machine

- No computers at the time (1936)
- Alan Turing was researching into the foundations of mathematics
- **Original intent:** capture all possible processes which can be carried out in computing a number†
- What about non-numerical problems?
- How do Turing machines capture all general and effective procedures which determine whether something is the case or not.

**Section 3.3**

†: Devise an algorithm that tests whether a polynomial has an integral root.
The Church-Turing thesis

- Any algorithm can be represented by an equivalent Turing machine
- A problem is computable if, and only if, there exists a Turing Machine that recognizes it.
- Turing Machines are equivalent to $\lambda$-terms
The Universal Turing Machine
Or, How do we study the limits of computability
The Universal Turing Machine

A Turing Machine that is capable of simulating any other Turing Machine

- Let $U$ be a TM.
- Given some TM $M$ and some input $w$, we can encoded as an input string, which we represent as $\langle M, w \rangle$

$U$ accepts $\langle M, w \rangle$ iff $M$ accepts $w$

Note that the Universal TM is a regular TM. This computability model is expressive enough to simulate itself.
Alan Turing's impact on modern computers

- Modern computers: von Neumann’s EDVAC design
- Fundamental idea of the EDVAC design: stored-programs
  Manipulation of programs as data
- **Universal Turing Machines pioneer the idea of stored programs**

TM are used to reflect on the limits and potentials of general-purpose computers by engineers, mathematicians and logicians (Module 3)

A single machine simulates all possible machine designs!

Without this idea, computers would have limited scope.
Multi-tape Turing Machine
The TM tape only grows to the right

- An important thing to note is that TMs have a tape that grows only to the right
- In turingmachine.io, the tape actually grows both ways

Are Turing machines that grow both sides more expressive?

Generalizing, are TM with multi-tapes more expressive?
Multi-tape Turing Machine

- A variation of the Turing Machine with multiple tapes
- The control may issue each head to move: forward, backward, or skip
Are Turing Machines less expressive than Multitape Turing Machines?
Turing Machines $\iff$ Multitape Turing Machines

- ($\Leftarrow$) Multitape Turing Machines trivially recognize the same language as Turing Machine (let the number of tapes be 1)
- ($\Rightarrow$) How can a single tape encode multitape?
Simulating a multitape

Tape encoding

- Concatenate the three tapes together
- Delimit each tape with a character that is not in the alphabet \#
- "Tag" the character to encode each tape head (virtual heads), eg \#·a
- The tape head always sits in the beginning of the tape
Simulating a multitape

Operation

- To move the $i$-th head, read the tape from the beginning until you read $\#$ a total of $i$ times and then seek until you find the marked character.
- If the virtual head $i$ hits the end of the tape $\#$, then shift the rest of the tape to the right and insert a blank character $\_$. 

Nondeterministic Turing Machines
Nondeterministic Turing Machines (NTM)

A machine can follow more than one transitions for the same input:

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

Consequence

Deterministic can only have one outgoing edge per character read, a nondeterministic machine can have multiple edges
Configurations in a TM

In a deterministic TM, a configuration history is *linear*

\[
\begin{align*}
abc & \rightarrow aac & \rightarrow abcx & \rightarrow abcxa & \rightarrow abcx & \rightarrow ay
\end{align*}
\]
In a nondeterministic TM, a configuration history is a tree!
Nondeterministic Turing Machines

- **Accept:** when *any* branch reaches $q_{start}$
- **Reject:** when *all* branches reach $q_{reject}$
- To find a single acceptance state we need to search the computation tree
Are Turing Machines less expressive than Nondeterministic Turing Machines?
Given an NTM, say $N$ we show how to construct a TM, say $D$

- If $N$ accepts on any branch, then $D$ halts and accepts
- If $N$ rejects on every branch, then $D$ halts and rejects

**Intuition**

Simulate all branches of the computation; search for any node with an accept state.

**Attention!**

**Question:** If we are searching a search tree, and there may exist infinite branches (due to loops), how should we search the tree: DFS or BFS?
Given an NTM, say \( N \) we show how to construct a TM, say \( D \)

- If \( N \) accepts on any branch, then \( D \) halts and accepts
- If \( N \) rejects on every branch, then \( D \) halts and rejects

**Intuition**

Simulate all branches of the computation; search for any node with an accept state.

**Attention!**

**Question:** If we are searching a search tree, and there may exist infinite branches (due to loops), how should we search the tree: DFS or BFS?

**Bread-First Search** will ensure our search is not caught in a never-ending branch.
Addressing configuration history

- We can use a sequence of numbers to uniquely identify each node of the configuration history

Unique paths
- 11
- 223
- 2221
- 221
- 21
- 2311
- 31
Using a TM to simulate a NTM

Use 3 tapes

1. **Initial input**: One tape for the input
2. **Simulation tape**: Where we will be executing an address
3. **Address tape**: An ever growing number that uniquely identifies where we are in the tree

How many choices at each step?

1. Copy tape 1 to tape 2
2. Simulate TM with address from the address tape; if it reaches an accepted state, then ACCEPT, otherwise continue
3. Increment address (next BFS-wise) and go to 1