Last time

- Three basic ways we model computations
  - Functions (λ functions)
  - Formal Languages
  - Abstract Automata
- The three ways are equivalent in power
- Church-Turing Thesis (all effective = mechanical computations can be done via λ functions or TMs)

Last time: Formal Languages

- Use grammar rules to define them
- Different allowed kinds of grammar rules leads to different families of languages
  - Type 3 = regular languages = regular expressions
  - Type 2 = context-free languages (practical for us)
  - Type 1 = context-sensitive languages (would be nice to use but it's not practical)
  - Type 0 = unrestricted languages
Last Time: Overview of Abstract Automata

- Abstract, not physical; state-based, perhaps memory.
- **Finite State Machines = Finite State Automata**
  - States, Transitions (symbol and state leads to new state); design of machine is fixed (no changing as you go).
- **Push-down Automata** (PDA) = FSA + stack
- **Turing Machine (TM)** = TM + memory tape

Today: FSA (FSM) Finite-State Automata / Machines

- Basic kind of abstract machine
  - Reads a string of symbols, decides at the end to **recognize/accept or reject** the string.
- The language of the automaton is the set of strings it accepts.

Also variations - produce output as you go

Transition Functions are represented using tables or diagrams

- Example: \( S = \{s_0, s_1, s_2, s_3\}, I = \{0, 1\}, \text{start} = s_0, F = \{s_0, s_2\} \)
- Transition function \( f \):

\[
\begin{array}{c|cc}
    & 0 & 1 \\
\hline
s_0 & s_1 & s_2 \\
s_1 & s_2 & s_0 \\
s_2 & s_3 & s_1 \\
s_3 & s_3 & s_2 \\
\end{array}
\]
• To execute this DFA on an input string $a_0 \ a_1 \ ... \ a_n$ we pass through a sequence of states $t_0, t_1, ..., t_{n+1}$ where $t_0$ is the start state and $t_{k+1} = f(t_k, a_k)$, for $k = 1, ..., n$.
• If the ending state $t_{n+1} \in F$, we accept the string else we reject it.
• The language of $M$ is $L(M) = \{ s \in I^* \mid M$ accepts $s\}$

E.g., does our DFA accept or reject each of the strings below?

- 01 $\quad$ A
- 0001 $\quad$ A
- 11101 $\quad$ A
- 0000000 $\quad$ R
- 1000001 $\quad$ A
• Another Example:

\[ L(M) = \text{bitstrings w/ even # of 1s} \]

Flip accept/reject states? Accept complement language!

\[ L(\text{this } M) = \text{complement of } L(\text{above } M) \]

odd # 1s

CS Dept., IIT

© J. Sasaki 2021

• A DFA that accepts bitstrings that begin with 00 and end with 1

\[ S_0 \text{ looking for initial 0} \]
\[ S_1 \text{ see 2nd 0} \]
\[ S_2 \text{ want ending 1} \]
\[ S_3 \text{ found ending 1} \]
\[ S_4 \text{ error/reject} \]
But there are easy-to-describe languages that can't be accepted by a DFA

- \( L = \{0^n1^n \mid n \geq 0\} \), e.g.
- I.e., \( L = \{\varepsilon, 01, 0011, 000111, \ldots\} \)
- The problem is that with a fixed number of states, you can only count so high.

Take a DFA with \( n \) states \( s_0, s_1, s_2, \ldots \)
- No matter where the state transition arrows go, if you want to follow \( > 2 \) arrows, you have to repeat a state. E.g., \( s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \)?

Let our DFA has \( n \) states, and let's assume its state transition diagram has a circuit.

- Let \( u \) be an input string that gets us to the circuit
- Let \( v \) be an input string that takes us around the circuit
- Let \( w \) be a string that takes us from the circuit to an accepting state
- By following the circuit an arbitrary number of times, we get strings \( uw, uvw, uvvw, uvvww, \ldots \in L(M) \).
- This is called the Pumping Lemma (we can pump up as many copies of \( v \) as we like).

Note: If a DFA with \( n \) states can accept a string of length \( > n \), then it must use a circuit of states.
(Diagram: DFA accepting a long string)

Say \( n = 4 \) states
\( w \) with 3 arrows

4th arrow - skip to reject?
jump back to earlier state? cycle!

- **Pumping Lemma**: For every DFA \( M \), there exists a number \( p \) such that if \( t \in L(M) \) with \( |t| \geq p \), then there are strings \( u, v, w \) such that \( |uw| \leq p \), \( |v| \geq 1 \), and \( uv^kw \in L(M) \) for all \( k \).

- (Note some DFAs only accept a finite set of inputs; those can't be pumped.)
Pumping Lemma example

- We can show that \( L = \{0^n 1^n \mid n \geq 0 \} \) is not accepted by any DFA.
- Say \( \exists \) such a DFA \( M \).
  - Let \( p \) be the integer from the pumping lemma.
  - The string \( 0^p 1^p \) has length \( = 2p \), so the lemma says that \( 0^p 1^p = u v w \) for some \( u, v, w \), and \( |v| \geq 1 \).
  - So \( u v v w \) is supposed to be \( \in L \), but \( v \) has form \( 0^+ \) or \( 1^+ \) or \( 0^+ 1^+ \), and all those make \( u v v w \notin L \).
  - Assuming \( M \) exists leads to a contradiction (\( 0^p 1^p \) can be pumped and it can't be pumped), so there's no such \( M \).

Advanced Machines — Push-Down Automata

- **Push-Down Automata (PDA):** an FSA plus a **stack**.
- A transition \( f(state, symbol, top-of-stack symbol) \) gives you a **new state** and a **top-of-stack replacement** string (pop stack and push list of stack symbols).
- PDAs are stronger than FSAs
  - A PDA can use the stack to count things
  - To recognize \( 0^n 1^n \), push each 0 onto the stack, then pop one off for each 1, check for too many/few 1s.
Advanced Machines — Turing Machine

- Turing Machines (TM): An FSA with a memory tape (infinite array of tape symbols)
- Current tape symbol = symbol under read/write head
- Transition $f(\text{state}, \text{input symbol}, \text{current tape symbol})$ gives the new state, a replacement tape symbol, and a R/W head command (move L/R 1 entry, or don't move).
- Not surprising: TMs more powerful than PDAs
- Surprising? Adding a second stack to a PDA gives you a TM equivalent.
Linear-Bounded Automaton

- There's a restricted form of TM called a **Linear-Bounded Automaton**
- Instead of infinite tape, it's restricted to tape length proportional to the input size
- Somewhat more realistic than fully infinite tape of TM

---

**Automata and Languages**

- There's a correspondence between language families and these automata:
  - **Type 3** (regular languages) and **FSAs**
  - **Type 2** (context-free languages) and **PDAs**
  - **Type 1** (context-sensitive languages) and **LBAs**
  - **Type 0** (unrestricted languages) and **TMs**
  - Church-Turing Thesis: TMs and $\lambda$ calculus can model any computing machinery & programming language.
  - As far as we know, this is true