CSE547T Class 2

Jeremy Buhler

January 23, 2017

1 From Languages to Machines

So far, we've talked abstractly about formal languages. Now, we need a connection to computers.

- We can formulate many computational problems as *decision* problems. "Does the input have some property P?"
- *Example*: does this string contain "bbb"?
- *Example*: do these two strings match?
- *Example*: are two strings within edit distance at most d?
- *Example*: does this graph have a minimum spanning tree of weight at most W? A clique of size at least k?
- *Example*: is this number a Mersenne prime?

Connection to strings?

- Devise some textual encoding of problem inputs as strings over some alphabet Σ.
- *Example*: graphs could be encoded as serialized adjacency lists or matrices over integers plus some special "delimiter" chars
- For any property P, we can define the language $L_P \subseteq \Sigma^*$ composed of all valid problem inputs with property P.
- Solving the decision problem for P is equivalent to asking, "Given a string x, is $x \in L_P$?"

Instead of studying algorithms in all their complexity, we will focus only on the ability to recognize strings in a given language L, since this is equivalent to solving many problems of interest. (Cf connection to optimization)

- **Defn**: an abstract machine M is a *recognizer* for a language $L \subseteq \Sigma^*$ if, given a string $x \in \Sigma^*$, it can determine whether or not $x \in L$.
- *Key question*: how powerful must a machine be to enable it to recognize a given language?

- Note that some languages may have arbitrarily long strings, so a recognizer must be able to answer correctly for arbitrarily large problem inputs.
- We'll build machines with some intuitively nice capabilities, then ask if they are powerful enough to recognize certain languages.

2 Our First Machines

Computers are many things, most of them unprintable.

- One thing they are is *finite*.
- In particular, they have only finite memory, and so can store only a limited number of states.
- (Of course, this number can be extremely large!)
- Another thing computers are is *deterministic*.
- Given the same program, starting state, and input, they produce the same output every time.
- We want an abstract machine that captures these essential properties!
- Appropriately, the standard model of this kind is called a **Deterministic Finite Automaton** (DFA).

Let's start with an example M over alphabet $\Sigma = \{0, 1\}$...

- Salient features of this machine?
- *states*: current state of machine is its (only) memory
- *start state* is marked by incoming arrow
- machine reads an input string one character at a time, causing successive state transitions.
- *transitions*: given current state and next character of string, what is new state? (must be specified for all chars in Σ)

- *accepting state(s)*: machine "accepts" a string iff it is left in accepting state after reading its last character. (Can have more than one)
- **Defn**: the *language* of a machine M, denoted $L(M)$, is the language of strings that it accepts.
- *Quick examples*: what does M do on "11010"? On "1001"?
- What is the language accepted by M?

Another example over alphabet $\Sigma = \{a, b, c\}$...

- For simplicity, we can label transitions with more than one char – that transition is taken on all chars
- What language does this machine accept?

3 Formal Definition of a DFA

OK, now that you have some intuition about what a DFA looks like, here is the formal definition (needed to prove things about DFAs).

- A DFA is a 5-tuple $M = (Q, \Sigma, q_0, A, \delta)$, where
- Q is finite set of states
- Σ is the alphabet of input chars (symbols)
- $q_0 \in Q$ is the unique starting state
- $A \subseteq Q$ is the set of accepting states
- $\delta : Q \times \Sigma \to Q$ is a *transition function.*
- If machine is in state q and sees char a, it moves to state $\delta(q, a)$.

Note that we can express δ in a standard tabular form, which may be easier to write or read in some cases. Example for first machine:

- δ allows us to express "where does M go from state q on char a"?
- A slightly extended notation lets us express where M goes when it reads an entire string.
- **Defn**: The extended transition function $\delta^* : Q \times \Sigma^* \to Q$ for a DFA M is defined as follows:
- $\delta^*(q, \varepsilon) = q$.
- For a string $x = ya$, $y \in \Sigma^*$, $a \in \Sigma$,

$$\delta^*(q, x) = \delta(\delta^*(q, y), a).$$

In this notation, we can compute for first machine, e.g., $\delta^*(q_{\text{even}}, 101)$.

- Now, we can properly define acceptance.
- A DFA M accepts a string x iff $\delta^*(q_0, x) \in A$.

4 Finger Exercise

Let's do one exercise that requires a (trivial) proof using the new DFA notation.

- Build me a machine M over $\Sigma = \{a, b\}$ for which

$$L(M) = \{x \in \{a, b\}^* \mid x \text{ contains } aba\}.$$

- Whew, that was fun. Are we sure this machine actually accepts the desired language? Let's prove that $L = L(M)$!
- Name machine's states q_ε, q_a, q_{ab}, q_{aba}.
- ($\rightarrow$) If x contains aba, then M accepts x.
- **Pf**: x is of the form $y \cdot aba \cdot z$ for some y, z.
- Reading y leaves M in some state q.
- Check that $\forall q \in Q$, $\delta^*(q, aba) = q_{aba}$.
- Hence, $\delta^*(q_\varepsilon, yaba) = q_{aba}$.
- Note that once in q_{aba}, M stays there for any z.
- Conclude that $\delta^*(q_\varepsilon, x) = q_{aba}$, and so M accepts x.

Oh dear, we're only half done.

- ($\leftarrow$) If M accepts x, then x contains aba.
- **Pf**: Let $x = yz$, where y is *shortest* prefix of x for which $\delta^*(q_\varepsilon, y) = q_{aba}$.
- (Such a y exists, since we get to q_{aba} and stay there!)
- I claim that y ends with aba.
- First, $|y| \geq 3$, since every path in M from q_ε to q_{aba} has length at least 3.
- Let $y = vw$, where $|w| = 3$.
- After reading v, M is in some state.
- Can $\delta^*(q_\varepsilon, v) = q_{ab}$ or q_{aba}? No! No way to get from these states to to q_{aba} *for first time* after *exactly* three moves (cannot get there sooner by minimality of prefix y).
- For q_ε and q_a, check that the only way to get to q_{aba} in exactly three moves is by reading the string aba.
- Conclude that $w = aba$, and so x contains aba. QED