6.2 Finite Automata

- simplest abstract machine
  - finite automaton can determine if words are in a language by using an
    - internal state
    - input tape
    - transition function
      - examines current state and next input symbol
      - determines next state
  - finite # of possible states
    - output a simple “accept” or “reject” if input word is in language that FA is designed for
  - input is examined one at a time
    - new state based on input symbol and current state
  - input is “consumed” -> we cannot use it again, must move onto next symbol
    - no memory
  - some state defined as “accepting states”
    - if FA stops in accepting state after reading all input, word is accepted
      - else word is rejected
  - a language is said to be decided by a FA if the FA always correctly indicates the words in and not in the language

- formal definition
  - FAs defined as a 5-tuple:
    - \( (Q, \Sigma, q_0, T, \delta) \)
      - \( Q \) is finite set of states
      - \( \Sigma \) is alphabet of input symbols
      - \( q_0 \) is starting state (one)
      - \( T \) is subset of \( Q \) giving accepting states
      - \( \delta \) is transition function
        - \( \delta : Q \times \Sigma \rightarrow Q \)
          - function that takes a state and input symbol
          - maps to state
  - best way? draw a transition graph
    - circle = state
    - annotated arrow = transition
      - transition defined as \( \delta(s_1, a) = s_2 \)
        - \( s_1 \) is “old” state
        - \( a \) is input symbol
        - \( s_2 \) is “new” state
      - also need to determine
        - starting state (indicated by arrow without previous state)
        - accepting state (double circles)
  - Example:
    - \( Q = \{1, 2, 3, 4\} \)
    - \( \Sigma = \{a, b\} \)
    - \( q_0 = 1 \)
    - \( T = \{4\} \)
    - \( \delta = \{ \)
      - \( (1, a) = 2, \)
      - \( (1, b) = 3, \)
      - \( (2, a) = 1, \)
      - \( (2, b) = 4, \)
      - \( (3, a) = 4, \)
      - \( (3, b) = 1, \)
      - \( (4, a) = 3, \)
      - \( (4, b) = 2 \)
    - \( \} \)
starts in state 1
- is ababbab in language?
  - YES!
    - other examples: ab, ba, baby, aha, bba, abb, bbaaba?
      - ab, ba, bbaaba accepted..
- what is the language that this FA accepts?
  - try simple set of words
  - look for pattern..
    - odd # of a’s and b’s

another example:
- Q = \{ 1, 2, 3 \}
- Σ = \{ a \}
- q0 = 1
- T = \{ 1 \}
- δ = 
  - \( (1, a) = 2, \)
  - \( (2, a) = 3, \)
  - \( (3, a) = 1 \)

accepting language?
- \( L = \{ \lambda, aaa, aaaaaa, aaaaaaaaa, \ldots \} \)
- multiples of 3 a’s

### 6.2.1 Regular Languages
- there are restrictions to the languages that FAs can accept
  - there are some subsets of \( Σ^* \) for which we can write a FA that can determine the words in the subset,
    - and there are subsets for which we cannot
- for now, concerned with languages that can be determined with FAs
  - referred to as \textbf{regular languages}
    - sets of reserved words for programming languages
    - set of possible variable names, operators, and numeric and string constants in programming languages
    - 1st step of compiling a program is to break program down into lexical units
      - process built on set of regular languages for the particular programming language

### 6.2.2 Regular Expressions
regular language also described by regular expression
- easier to define in recursive manner
- can produce new regular expressions using
  - union (r₁ + r₂)
  - concatenation (r₁ · r₂)
  - and star closure (r*)
  - parentheses
- nothing is a regular expressions unless it can be written in this fashion

- example regex
  - (aaa)* = {λ, aaa, aaaaaa, aaaaaaaaa, …}
  - (ab)* = {λ, ab, abab, ababab, …}
  - a*b* = {λ, a, b, aa, ab, bb, aaa, aab, abb, bbb, …}
  - (a + b)*bbb(a + b)* = {bbb, abbb, bbbb, bbba, aabbb, bbbbb, bbbab, …}
  - (a + b)(a + b)(a + b)(a + b)(a + b)* = {λ, a, b, aa, ab, bb, aaa, aab, abb, bbb, …}

rules for creating regex
- 1) ∅, λ, and all a ∈ Σ are primitive regular expressions
- 2) if r, r₁, and r₂ are regular expressions, then so are r₁ + r₂, r₁ · r₂, r*, and (r)
- 3) A string represents a regular expression iff symbols of step (1) and repeated applications of step (2) can create it

- ∅ is the empty language
  - no words
- λ is language w/ just empty word in it
  - 1 word, with no symbols
  - these are two different languages

example
- L(ab(bb)* + (bb)*) = {λ, ab, bb, abbb, bbbb, abbbbb, …}

6.2.3 Regular Grammars
- regular languages require right-linear grammars
- one in which all of its products are of the form
  - A -> xB or A -> x with x ∈ Σ
  - left hand side of product can have only a single non-terminal symbol
  - right side can have zero or more terminal symbols
  - but at most 1 nonterminal
  - nonterminal must always be right-most symbol

6.2.4 Deterministic and nondeterministic FAs
- skipped.

6.2.5 Coding FAs
- skipped

6.3 Designing FAs
must understand words in language
find pattern
e.g.,
- **L1, L4, L5, L7 not regular!!**
  - cannot be determined by FA
  - important for determining class of language
- **L1** is NOT regular
  - no memory! how can we determine if #a’s same as #b’s? (without knowing n?)
- **L4** is NOT regular
  - cannot count number of a’s and b’s
  - need 1 state for each possible count of a..
    - but FAs are FINITE!!
- **L5** not regular
  - again, must count #a’s and #b’s
  - FAs have no memory!!
- key is to know **exactly** how many to count
  - e.g., checking for 5 “c” is ok.. since we know there are exactly 5

![FIGURE 6.9](image1) Languages and sample words

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₁ = {a^n b^n : n ≥ 0}</td>
<td>(λ, ab, aabb, aaabbb, aaaaabbb, \ldots)</td>
</tr>
<tr>
<td>L₂ = {a^n b^m : n ≥ 0 and m ≥ 0}</td>
<td>(λ, a, b, aa, ab, bb, aaa, aab, abb, bbb, \ldots)</td>
</tr>
<tr>
<td>L₃ = {(ab)^n : n ≥ 0}</td>
<td>(λ, ab, abab, ababab, abababab, \ldots)</td>
</tr>
<tr>
<td>L₄ = {w : #a(w) = #b(w)}</td>
<td>(λ, ab, be, aabb, abab, abba, baab, bab, bbaa, \ldots)</td>
</tr>
<tr>
<td>L₅ = {a^n b^m : n = m + 3}</td>
<td>(aaa, aaab, aaaaab, aaaaaabbb, aaaaaabbbb, \ldots)</td>
</tr>
<tr>
<td>L₆ = {a^n b^m : n = m \mod 2}</td>
<td>(λ, ab, bb, abbb, bbabb, bbabbb, bbbbbb, \ldots)</td>
</tr>
<tr>
<td>L₇ = {a^n b^m : n &lt; m + 1}</td>
<td>(λ, b, ab, bb, aabb, bbb, abb, abbb, aabbb, \ldots)</td>
</tr>
<tr>
<td>L₈ = {a^n : n \mod 5 = 2}</td>
<td>(aa, aaaaaa, aaaaaaaaaa, aaaaaaaaaaa, \ldots)</td>
</tr>
</tbody>
</table>

![FIGURE 6.11](image2) A deterministic finite automaton for language L₃

- **FIGURE 6.9** Languages and sample words
- **FIGURE 6.11** A deterministic finite automaton for language L₃
6.3.3 Regular grammar
- skipped.

- Note: there is a way to prove a language is NOT regular
  - pumping lemma.. (yawn)