Chap. 1.2 **NonDeterministic Finite Automata** (NFA)

- DFAs: exactly 1 new state for any state & next char
- **NFA**: machine may not work “same” each time
  - More than 1 transition rule for same state & input
    - Any one is valid
    - Choice is made with “crystal ball” – which one will lead to an accepting state if possible
  - Also $\varepsilon$ (the empty string) is allowed on an edge:
    - State transition can be made without reading any input characters
- See page 48 Fig. 1.27. two “1s” from $q_1$ & $\varepsilon$ on $q_2\rightarrow q_3$
  - Accepts all strings from \{0,1\}* containing 101 or 11
- How does computation proceed? Assume at a step where multiple options are possible – a separate copy of the NFA is started up for each, and run in parallel
  - All with the same starting state and remaining input
  - Each takes a different edge
  - Acceptance if *any* end up in an accepting state
- See Fig. 1.28 – note a “1” from $q_1$ can go to $q_2$ *or* (because of $\varepsilon$ leaving $q_2$) go to $q_3$
• Ways to think of nondeterminism
  • Parallel threads checking different paths
  • Tree of possibilities
  • NFA always “guesses” correctly (crystal ball)

• Examples
  • (p.51) Ex. 1.30 N2: a “1” in third position from end
    • Nondeterminism is knowing when we are 3 symbols from end
  • (p.52) Ex. 1.33 N3: 0k, where k is multiple of 2 or 3
    • \(\varepsilon\) edges lead to two different DFAs
      • One that accepts strings of two 0s
      • One that accepts strings of 3 0s
    • At start, crystal ball “knows” which it is
  • (p.53) Ex. 1.35 N4: \{ \(\varepsilon\), a, bb, babba, …\}
• **(p.53) NFA Formal Definition**: \( N = (Q, \Sigma, \delta, q_0, F) \)
  - \( Q, \Sigma, q_0, \) and \( F \) are all as before
  - \( \delta: Q \times \Sigma^* \rightarrow P(Q) \)
    - \( \Sigma^* = \Sigma \cup \{\varepsilon\} \) – epsilon-extended alphabet
    - \( P(Q) = \) power set of \( Q \) – set of all subsets of \( Q \)
    - Thus each member of \( P(Q) \) is a subset of \( Q \)
  - \( N \) accepts \( w \) (\( w \) a string from \( \Sigma^* \)) if
    - \( w = y_1 y_2 \ldots y_m \) where \( y_i \in \Sigma^* \) (i.e. some may be “\( \varepsilon \)”)
    - there exists a sequence of states \( r_0, r_1, \ldots r_m \) where
      - \( r_0 = q_0, \ r_m \in F \)
      - \( r_{i+1} \in \delta(r_i, y_{i+1}) \)
  - p. 54: e.g. \( N_1 \) accepts all strings containing 101 or 11
    - Look at transition table – each transition is to a set of states
      - Remember \( \phi \) is “empty set”
• (p.55) **Theorem** Every NFA has an equivalent DFA.

**Proof by construction:** given NFA, build matching DFA

**Basic idea:** matching DFA has one state for *every possible set of states* that NFA can be in at any time

• Assume given NFA $N = (Q, \Sigma, \delta, q_0, F)$

• Build DFA $M = (Q', \Sigma, \delta', q_0', F')$

• Simple case first, if *no ε rules in N*
  • $Q' = P(Q)$
  • $q_0' = \{q_0\}$
  • $F' = \{R | R \in Q', R \text{ contains an accept state from } F\}$

• for each $R \in Q'$, and $a \in \Sigma$:
  • $\delta'(R, a) = \{q | q \in Q, \text{ for some } r \in R, \delta(r,a)=q\}$
  • Note: $\delta'(R, a)$ can return empty set $\emptyset$

• If there are ε rules in N: i.e. some $\delta(q, \varepsilon) \to q'$
  • Define for any $R \in Q'$, $E(R) = \{q | q \in Q, q \text{ can be reached from some } q' \in R \text{ via } 0 \text{ or more } \varepsilon \text{ edges}\}$
    • $E(R)$ = "ε reachable states" from $R$ in 0 or more ε
  • Now $\delta'(R, a) = \{q | q \in Q, \text{ for some } r \in R, q \in E(\delta(r,a))\}$
  • Also $q_0' = E(\{q_0\})$

• If NFA has $|Q|$ states, DFA has up to $2^{|Q|}$ states

• **KEY RESULT:** NFAs are no more powerful than DFAs!
  • Just easier to design
Example 1.41: p. 56 convert NFA N₄ to DFA D

- Q = {1, 2, 3} – states of N₄
- P(Q) = {{}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}}
  - Each represents a possible state of D
- Compute E - states reachable by ε - of each state of Q’
  - E({1}) = {1, 3} – 3 because of ε from 1 to 3
  - E({2}) = {2} – no ε from 2
  - E({3}) = {3}
  - E({1,2}) = {1,2,3}
  - E({1,3}) = {1,3}
  - E({2,3}) = {2,3}
  - E({1,2,3}) = {1,2,3}
- Start state is E of N₄’s start state 1 = E({1}) = {1,3}
- Accept states are those containing any of N₄’s F states (1)
  - {{1}, {1,2}, {1,3}, {1,2,3}}
- See Fig. 1.43 p. 58
  - Note no edges into {1} or {1,2} so could eliminate
  - See Fig. 1.44 for reduced graph (no way to get to {1} or {1,2}
Details of Transitions in Fig. 1.43

- {2} in D
  - input a: {2,3} because N has a edge from 2 to 2 & 3
  - input b: {3}
- {1} in D
  - input a: φ because no a’s leave 1 in N
  - input b: {2} because b edge from 1 to 2 in N
- {3} in D
  - input a: {1,3} because in N a edge from 3 to 1
    - but also from 1 there’s an ε edge back to 3
  - input b: φ because no a’s leave 3 in N
- {1,2} in D
  - input a: {2,3} while 1 has no a edges, 2 does to {2,3}
  - input b: {2,3} N has a b edge from 1 to 2
    - and a b edge from 2 to 3
- {1,3} in D
  - input a: {1,3} while 1 has no a edges,
    - from 3 there is a edge to 1, with an ε back to 3
  - input b: {2} N has a b edge from 1 to 2
    - but no b edges from 3
- {2,3} in D
  - input a: {1,2,3} a edge from 2 to 2,
    - from 3 there is a edge to 1, with an ε back to 3
  - input b: {3} N has a b edge from 2 to 3
    - but no b edges from 3
- {1,2,3} in D
  - input a: {1,2,3} no a edges from 1
    - but a edge from 2 to 2 and 3
    - from 3 there is a edge to 1, with an ε back to 3
  - input b: {2,3} N has a b edge from 1 to 2
    - and b edge from 2 to 3
• Alternative from transition table

• N’s original transition table:

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>ε</th>
<th>E(state)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{}</td>
<td>{2}</td>
<td>{3}</td>
<td>{1,3}</td>
</tr>
<tr>
<td>2</td>
<td>{2,3}</td>
<td>{3}</td>
<td>{}</td>
<td>{2}</td>
</tr>
<tr>
<td>3</td>
<td>{1}</td>
<td>{}</td>
<td>{}</td>
<td>{3}</td>
</tr>
</tbody>
</table>

• D’s Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>E(1) = {1,3}</td>
<td>E(2) = {2}</td>
</tr>
<tr>
<td>{2}</td>
<td>E(2) U E(3) = {2} U {3} = {2,3}</td>
<td>E(3) = {3}</td>
</tr>
<tr>
<td>{3}</td>
<td>E(1) = {1,3}</td>
<td>E(4) = {}</td>
</tr>
<tr>
<td>{1,2}</td>
<td>E(1) U E(2) U E(3) = {2,3}</td>
<td>E(2) U E(3) = {2,3}</td>
</tr>
<tr>
<td>- &gt; {1,3}</td>
<td>E(1) = {1,3}</td>
<td>E(2) U E(4) = {2}</td>
</tr>
<tr>
<td>{2,3}</td>
<td>E(1) U E(2) U E(3) = {1,2,3}</td>
<td>E(3) = {3}</td>
</tr>
<tr>
<td>{1,2,3}</td>
<td>E(1) U E(2) U E(3) = {1,2,3}</td>
<td>E(2) U E(3) = {2,3}</td>
</tr>
<tr>
<td>{}</td>
<td>E(4) = {}</td>
<td>E(4) = {}</td>
</tr>
</tbody>
</table>

• To E’s that contain 1 in state, add 3 because of ε 1->3
• Each cell δ'(q',x) is Union of E(δ(q,x)) where q is in set q’
• Red states are in D’s final set
• {1,3} is D’s start state because its E(1) where 1 is N’s state