Outline

- Nondeterminism
- Regular expressions
- Elementary reductions
Determistic Finite Automata

- A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)
  - \(Q\) is a finite set of states
  - \(\Sigma\) is an alphabet
  - \(\delta : Q \times \Sigma \rightarrow Q\) is a transition function
  - \(q_0 \in Q\) is the initial state
  - \(F \subseteq Q\) is a set of final or accepting states
**Transition diagrams**

\[ Q = \{ q_0, q_1, q_2, q_3 \} \]
\[ \Sigma = \{ 0, 1 \} \]
\[ \delta = \]

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>( q_2 )</td>
</tr>
<tr>
<td>( q_0 )</td>
<td>1</td>
<td>( q_1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>( q_3 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>1</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>0</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>( q_3 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>( q_1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>1</td>
<td>( q_2 )</td>
</tr>
</tbody>
</table>

\[ q_0 = q_0 \]
\[ F = \{ q_0 \} \]
A finite automaton

1. In state q
   a. read a symbol c
   b. move the tape head right
   b. goto state $\delta(q, c)$

2. Accept iff the FA is in a final state after reading the last symbol
Definition of regular expressions

- Let $\Sigma$ be an alphabet. The regular expressions are defined inductively as follows:

  - $\emptyset$ is a regular expression denoting $\{\}$,
  - $\epsilon$ is a regular expression denoting $\{\epsilon\}$,
  - for each $a \in \Sigma$, $a$ is a regular expression denoting $\{a\}$,
  - Assume $r$ and $s$ are regular expressions denoting sets $R$ and $S$, then
    - $rs$ denotes $RS$,
    - $r + s$ denoted $R \cup S$,
    - $r^*$ denotes $R^*$
Examples of regular expressions

- $abc$ denotes $\{abc\}$,
- $a^* b$ denotes $\{b, ab, aab, aaab, \ldots\}$,
- $(a + b)^* aa (a + b)^*$: all strings containing at least two consecutive $a$'s.
- $(0^*(101*01)^*)^*$: all binary numbers that are a multiple of 3.
Nondeterminism

- For *deterministic* finite automata, the transition function $\text{delta}$ is a *function*
  - $\text{delta} : \text{state} \times \text{symbol} \rightarrow \text{state}$
  - For each state, and each symbol, there is a unique next state
- What if we relax this?
- In a nondeterministic finite automaton, $\text{delta}$ is a transition *relation*
**Definition of an NFA**

- A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, s, F)\)

  - \(Q\) is a *finite* set of *states*
  - \(\Sigma\) is an alphabet
  - \(\delta : Q \times \Sigma \rightarrow 2^Q\) is a *transition relation*
  - \(s \in Q\) is the *initial state*
  - \(F \subseteq Q\) is a set of *final* or *accepting* states
The delta relation

- Note, the following definitions are equivalent:
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition relation
  - $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation

- The transition diagram is often drawn without empty edges:
  - if $\delta(q, c) = \{\}$, then the machine rejects and we do not draw the edge
Example NFA
**Executions (for DFAs)**

- An execution of a **deterministic** automaton $M = (Q, \Sigma, \delta, q_0, F)$ is a sequence

  $$(q_0, c_0)(q_1, c_1) \ldots (q_{n-1}, c_{n-1})q_n \in (Q \times \Sigma)^*$$

  such that:

  - $q_0$ is the start state
  - $\delta(q_i, c_i) = q_{i+1}$

- The **execution string** $s$ is $c_0 c_1 \ldots c_{n-1}$

- If there exists an execution for string $s$ with $q_n \in F$ the automaton **accepts** and $s \in L(M)$; otherwise it **rejects**
**Executions (for NFAs)**

- An execution of a **nondeterministic** automaton $M = (Q, \Sigma, \delta, q_0, F)$ is a sequence

$$ (q_0, c_0)(q_1, c_1) \ldots (q_{n-1}, c_{n-1})q_n \in (Q \times \Sigma)^* $$

such that:

- $q_0$ is the start state
- $q_{i+1} \in \delta(q_i, c_i)$

- The execution string $s$ is $c_0c_1 \ldots c_{n-1}$
- If there exists an execution for string $s$ with $q_n \in F$ the automaton accepts and $s \in L(M)$; otherwise it rejects
A nondeterministic finite automaton

1. In state q
   a. read a symbol c
   b. move the tape head right
   b. goto some state in delta(q, c)
2. Accept iff the NFA is in a final state after reading the last symbol
Execution models

1. Classical: For each state $q$ and input symbol $c$, the automaton makes a nontermistic choice among all the possible options $\delta(q, c)$; it accepts if it made all the right choices.

2. Another choice:
   - At any point, the automaton is in a superposition of states $Q_1$.
   - For each symbol $c$, it enters states $Q_2$ where $q_2 \in Q_2$ iff $\exists q_1 \in Q_1. q_2 \in \delta(q_1, c)$.
Extending the transition relation

- Defining \( \hat{\delta} \):
  
  - \( \hat{\delta}(q, \epsilon) = \{ q \} \)
  
  - \( \hat{\delta}(q, wa) = \{ p \mid \exists r \in \hat{\delta}(q, w).p \in \delta(r, a) \} \)

- A machine \( M = (Q, \Sigma, \delta, q_0, F) \) accepts string \( s \) iff 
  \( \hat{\delta}(q_0, s) \cap F \neq \{ \} \)
Expressive power of NFAs vs. DFAs

- Every DFA is an NFA
- NFAs allow more transitions
- Is the language more expressive?
  - *Is there a language* \( L = L(M) \) *for some NFA* \( M \), *where* \( L \neq L(M') \) *for some DFA* \( M' \)?
NFA/DFA Equivalence

• To prove equivalence of DFAs and NFAs we must do two things:
  – I: For each DFA, produce an NFA that accepts the same language
  – II: For each NFA, produce a DFA that accepts the same language

• This is called a reduction—reduce one construction to another equivalent one
  – We’ll do part II; part I is immediate
NFA->DFA construction

- Consider an arbitrary NFA $M = (Q, \Sigma, \delta, s, F)$; construct an equivalent DFA $M' = (Q', \Sigma, \delta', s', F')$
  
  - Let $Q' = 2^Q$
  - Let $s' = \{s\}$
  - Let $\delta'(\{q_1, \ldots, q_n\}, c) = \bigcup_{i=1}^{n} \delta(q_i, c)$
  - Let $F' = \{ t \in 2^Q \mid t \cap F \neq \{\} \}$
NFA->DFA construction: verification

**Theorem** $\delta(s, x) = \delta'(\{s\}, x)$

**Corollary** $\delta(s, x) \cap F \neq \{\} \iff \delta'(\{s\}, x) \cap F' \neq \{\}$
NFA->DFA construction: base case

Theorem $\delta(s, x) = \delta'(\{s\}, x)$

Base case $\delta(s, \epsilon) = \{s\} = \delta'(\{s\}, \epsilon)$
NFA->DFA construction: induction step

- Assume $\delta(s, x) = \delta'(\{s\}, x)$ for some $x$

- Prove $\delta(s, xa) = \delta'(\{s\}, xa)$ for any $a$

$$\delta(s, xa)$$
$$= \text{by definition of } \hat{\delta}$$
$$\delta(\delta(s, x), a)$$
$$= \text{by definition of } \hat{\delta}$$
$$\bigcup_{p \in \delta(s, x)} \delta(p, a)$$
$$= \text{by induction}$$
$$\bigcup_{p \in \delta'(s, x)} \delta(p, a)$$
$$= \text{by definition of } \delta'$$
$$\bigcup_{p \in \delta'(s, x)} \delta'(\{p\}, a)$$
$$= \text{by definition of } \delta'$$
$$\bigcup_{p \in \delta'(\{s\}, x)} \delta'(\{p\}, a)$$
$$= \text{by definition of } \delta'$$
$$\delta'(\delta'(\{s\}, x), a)$$
$$= \text{by definition of } \delta'$$
$$\delta'(\{s\}, xa)$$
NFA with epsilon transitions

• Suppose we add a new twist
  - Allow transitions that require no input
  - Label them epsilon (no input)
  - The automaton may choose to take an epsilon transition without any input

• Executions include \((q_i, \text{epsilon}) (q_{i+1}, \_\) steps

• A machine accepts if there exists an execution that accepts
Example e-NFA
E-NFA definition

• A nondeterministic finite automaton with $\epsilon$ transitions is a 5-tuple $(Q, \Sigma, \delta, s, F)$
  
  - $Q$ is a finite set of states
  - $\Sigma$ is an alphabet
  - $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ is a transition relation
  - $s \in Q$ is the initial state
  - $F \subseteq Q$ is a set of final or accepting states
E-NFA->NFA reduction

- Consider the superposition model
  - If the e-NFA could be in states \(\{q_1, q_2, \ldots, q_n\}\), add the states that could be reached by e-transitions from any state \(q_i\)
  - This is called the e-closure
E-closure example

\[ \varepsilon\text{-closure}(q_0) = \{ q_0, q_1, q_2 \} \]
\[ \varepsilon\text{-closure}(q_1) = \{ q_1, q_2 \} \]
\[ \varepsilon\text{-closure}(q_2) = \{ q_2 \} \]
Formal definition

• Define \( \varepsilon\)-closure\((P) = \bigcup_{p \in P} \varepsilon\)-closure\((p) \)
  
  - Let \( \hat{\delta}(q, \varepsilon) = \varepsilon\)-closure\((q) \)
  
  - Let \( \hat{\delta}(q, xa) = \varepsilon\)-closure\((P) \), where \( P = \{ p | \exists r \in \hat{\delta}(q, w). p \in \delta(r, a) \} \)
  
  - Define \( \delta(R, a) = \bigcup_{r \in R} \delta(r, a) \)
  
  - Define \( \hat{\delta}(R, a) = \bigcup_{r \in R} \hat{\delta}(r, a) \)

• \( M = (Q, \Sigma, \delta, s, F) \) accepts string \( x \) iff \( \hat{\delta}(s, x) \cap F \neq \{\} \)
Reduction

- Let $M = (Q, \Sigma, \delta, s, F)$ be an NFA with $\epsilon$ moves.
- Construct $M' = (Q, \Sigma, \delta', s, F')$ an NFA without $\epsilon$ moves.
  - Let $\delta'(q, a) = \hat{\delta}(q, a)$
  - Let $F' = \begin{cases} F \cup \{s\} & \text{if } \epsilon\text{-closure}(s) \cap F \neq \{\} \\ F & \text{otherwise} \end{cases}$
Proof

Prove $\delta'(s, x) = \hat{\delta}(s, x)$ by induction on the length of $x$

Note that $\delta'(s, \epsilon) = \hat{\delta}(s, \epsilon)$ may not be true. Oh well, start the induction at $|x| = 1$.

Base If $|x| = 1$ then $x = a$ and $\delta'(s, a) = \hat{\delta}(s, a)$ by definition
**Induction step**

**Step** Suppose $x = w a$

\[
\delta'(s, w a) \\
= \text{by definition} \\
\delta'(\delta'(s, w), a) \\
= \text{by induction} \\
\delta'(\hat{\delta}(s, w), a) \\
= \text{by definition} \\
\bigcup_{q \in \hat{\delta}(s, w)} \delta'(q, a) \\
= \text{by definition of } \delta' \\
\bigcup_{q \in \hat{\delta}(s, w)} \hat{\delta}(q, a) \\
= \text{by definition of } \hat{\delta} \\
\hat{\delta}(s, wa)
\]
Summary (so far)

• Proofs so far
  - Proved NFA->DFA
  - Proved e-NFA->NFA

• Next, establish total equivalence
  - Prove regex->e-NFA
  - Prove DFA->regex

• If so, all four are equivalent
Definition of regular expressions

- Let $\Sigma$ be an alphabet. The regular expressions are defined inductively as follows:
  - $\emptyset$ is a regular expression denoting $\{\}$,
  - $\epsilon$ is a regular expression denoting $\{\epsilon\}$,
  - for each $a \in \Sigma$, $a$ is a regular expression denoting $\{a\}$,
  - Assume $r$ and $s$ are regular expressions denoting sets $R$ and $S$, then
    - $* \ r \ s$ denotes $RS$,
    - $* \ r + s$ denoted $R \cup S$,
    - $* \ r^*$ denotes $R^*$
• Prove by *structural induction*: induction on the size of the regular expression

• Base cases:
  - *Show the empty RE has an e-NFA*
  - *Show the RE has an e-NFA*
  - *Show the a (symbol) RE has an e-NFA*

• Induction:
  - *Show (xy) has an e-NFA*
  - *Show (x + y) has an e-NFA*
  - *Show x* has an e-NFA*
**Base cases**

\[ r = \emptyset \]

\[ r = \varepsilon \]

\[ r = a \]
Choice

Start state

Machine 1

Any (originally) final state

Machine 2

ε

ε

ε

ε

ε

ε
Concatenation

Machine 1

ε

Machine 2
Kleene closure
**Final proof: DFA->regex**

- Think about it