Introduction

• Previous model is deterministic
  - Given a state and input, there is a unique state that we transition to
• In a nondeterministic machine
  - several choices may exist for what state to transition to next
• Nondeterminism is a generalization of determinism
  - Every deterministic machine is a nondeterministic machine
• Does nondeterminism add more power?
  - If so, in what sense?
• How does it compute?
  - Any time there is a choice, run a copy of the machine for each choice
  - If no subsequent state, that copy (computation path) dies
  - If any computation path accepts, accept the string

• Can think of nondeterminism as a tree of possibilities
  - Succeeds if one branch is possible

• Previous example:
  - Trace through 010110
  - What is language of machine?

• Why is NFA interesting? (not more powerful than DFA)
  - Allows us to express some problems in a simpler form
  - Prepares us for nondeterminism in more complex machines

Example

![Diagram of states and transitions]

• Additions
  - Allow multiple transitions from the same state and token
  - Allow no transition from a state and token
  - Allow epsilon transitions
    + Using no input to transition
Example

- What would an equivalent DFA be?
  - Do the case where just last character must be a 1
  - Where 2nd last character must be a 1
  - Where 3rd last character must be a 1

+ Note that NFA can guess when the string will end, simplifying its structure; but DFA must account for if the string has not finished yet.

- What language does it accept?
  - at state q1, it guesses that it is 3 characters from the end

- Non-determinism used to guess when end of string will be
  - If guess is wrong, computation path dies out

Example (continued)
Another Example

- Add epsilon labels from $q_2$ to $q_3$ and from $q_3$ to $q_4$
  - What language will it now accept?
  - How do we modify the DFA?
A non-deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. $Q$ is a finite set of states
2. $\Sigma$ is a finite alphabet
3. $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is the transition function
   - Note: $\mathcal{P}(Q)$ is the power set of $Q$ — all possible subsets
   - Note: $\Sigma_\epsilon$ is $\Sigma \cup \{\epsilon\}$
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

Example

- What is its formal definition?
Definition of Computation

**Definition:** Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w$ a string over $\Sigma$. We say that $N$ accepts $w$ if $w$ can be written as $y_1y_2\ldots y_m$ where $y_i \in \Sigma^*$ and there is a sequence of states $r_0, r_1, \ldots, r_m$ such that
1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1})$
3. $r_m \in F$

• Note that there are many choices for a sequence
  - It accepts if any legal sequence ends will an accept state

Overview

• Non-determinism
• Formal Definition
  ⇒ Equivalence of NFA and DFA
Equivalence of NFA and DFA

- DFA and NFA recognize the same class of languages
  - A language can be recognized by a DFA iff can be recognized by a NFA
- Surprising!!
  - NFAs seem to be more powerful than DFA due to nondeterminism
- Useful
  - Means we can use nondeterminism to show that a language is regular
  - Use NFAs to show that regular languages are closed under composition

Theorem

- 2 machines are equivalent if they recognize the same language

Theorem 1.39: Every NFA has an equivalent DFA.

- Proof Idea (ignoring epsilon transitions)
  - NFA is basically exploring a bunch of paths in parallel
  - As it is consuming its input, it can be in one of a number of states
    + Think of this as putting a finger on each possible state that NFA can be in
    + To consume next input token, update each finger
    + Might add more fingers for splits
    + Might remove finger if computation path dead-ends, or converge
  - Construct DFA in which all possible subsets of states of the NFA are states of the DFA
    + If NFA has \( k \) states, DFA has \( 2^k \) states
    + Transition function simulates how all of the fingers change
Proof (without Epsilon)

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA that recognizes some language $A$.
- Construct DFA $M = (Q', \Sigma, \delta', q'_0, F')$ recognizing $A$ as follows
  + $Q' = \mathcal{P}(Q)$. $\mathcal{P}(Q)$ is the powerset of $Q$ — all subsets
  + For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R,a) = \{q \in \delta(r,a) \mid r \in R\}$
  + Each state in $N$ can go to a set of states in $N$ on input $a$.
  + So, take union of all successor states: $\delta'(R,a) = \bigcup_{r \in R} \delta(r,a)$
  + $q'_0 = \{q_0\}$.
  + $F' = \{R \in Q'|R$ contains an accept state of $N\}$
    + $M$ accepts if one of the possible states that $N$ could be in at this point is an accept state.
- Need to argue that for any string $w$, $M$ accepts $w$ iff $N$ accepts $w$.
  + At each point in the computation, the possible states that $N$ can be is the meta-state that $M$ is in.
  + Would need to do this by induction. Simple enough that we won’t bother.
Now for Epsilon Transitions

- As we are tracing out computation in $N$, place additional fingers on all states that can also be reached by going along $\epsilon$ arrows after every step (this is simpler than before each step)
  - Consider $R \subseteq Q$
    + Let $E(R) = \{q | q \in Q$ and can be reached from a state in $R$
      by traversing 0 or more $\epsilon$ arrows\}
      Remember $Q$ is the set of states of the NFA, so $R$ is a state of the DFA, as is $E(R)$
  - Modify the transition function of $M$ to take into account the states in $N$
    reachable after $\epsilon$ transitions
    + Change $\delta'(R, a) = \{q | q \in \delta(r, a)$ for some $r \in R\}$
      to $\delta'(R, a) = \{q | q \in E(\delta(r, a))$ for some $r \in R\}$
    - Also need to modify start state of $M$ to be set of states reachable from
      with $\epsilon$ transitions: $q_0' = E(\{q_0\})$

Example

- NFA that recognizes language where last or 2nd last token is a 1
  - Step 1: Determine NFA diagram
  - Step 1: Give transition table for NFA
  - Step 2: Determine transition table for DFA (all states) ignore epsilon
    + by combining rows of NFA transition table
  - Step 3: Determine $R$ for each state
  - Step 4: Apply $R$ to start state of DFA
  - Step 5: Apply $R$ to result of Step 2 (starting with start state)
Corollary

**Corollary 1.40:** A language is regular iff some NFA recognizes it

- **Proof ⇒**
  - Let $A$ be a regular language
  - So, $M$ a DFA recognizes it
  - $M$ is also a NFA

- **Proof ⇐**
  - Let $N$ be a NFA
  - By last theorem, there is a DFA that recognizes that language
  - So language must be regular