Nondeterministic Finite Automata (NFAs)

Wednesday September 15, 2021
Announcements

• HW1 due Sun 9/19 11:59pm EST
  • Upload solutions to Gradescope
  • LaTex is great!
  • Handwritten and scanned/photo is perfectly fine
  • I must be able to read your answers!
    • Illegible solutions will not receive any credit

• Please post HW questions to Piazza
  • Don’t email me directly
  • So others can benefit from the discussion, and potentially help out!

• Monday 9/13 lecture video posted

• Welcome new students!
  • Make sure to catch up ASAP
Last Time: Finite State Automaton, a.k.a. DFAs

DEFINITION 1.5

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the states,
2. \(\Sigma\) is a finite set called the alphabet,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

• **Key characteristic:**
  • Has a **finite** number of states
  • I.e., it’s a computer with a finite amount of memory
    • Can’t dynamically allocate

• Often used for **text matching**
Combining DFAs?

Password Requirements

» Passwords must have a minimum length of ten (10) characters - but more is better!
» Passwords **must include at least 3** different types of characters:
  » upper-case letters (A-Z)
  » lower-case letters (a-z)
  » symbols or special characters (%, &, *, $, etc.)
  » numbers (0-9)
» Passwords cannot contain all or part of your email address
» Passwords cannot be re-used

To match all requirements, can we combine smaller DFAs?

For more information, visit: https://www.umb.edu/it/password
Combining DFAs

Problem 1: What should be the transition labels?

Problem 2: Once we enter one of the machines, can’t go back to the other one!

\[ M_1: \text{Check special chars} \]

\[ M_2: \text{Check uppercase} \]

\[ M_3: \text{OR} \]

Combined machine adds new start state

We need a different kind of machine!

Idea: nondeterminism allows being in multiple states (i.e., multiple machines) at once!
Nondeterminism

Deterministic computation

- start
- ... states
- accept or reject

Nondeterministic computation

- ... reject
- ... states
- accept

Nondeterministic computation can be in multiple states at the same time
Nondeterministic Finite Automata (NFA)

**Definition 1.37**

A *nondeterministic 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,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

**Compare with DFA:**

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states*.

**Difference**

- Power set, i.e. a transition results in *set of states*
Power Sets

• A power set is the set of all subsets of a set

• **Example**: $S = \{a, b, c\}$

• Power set of $S =$
  • $\{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$
  • **Note**: includes the empty set!
A **nondeterministic 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. \(\delta: Q \times \Sigma \varepsilon \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

\[
\Sigma_{\varepsilon} = \Sigma \cup \{\varepsilon\}
\]

**Definition 1.37**

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the **states**, 
2. \(\Sigma\) is a finite set called the **alphabet**, 
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**, 
4. \(q_0 \in Q\) is the **start state**, and 
5. \(F \subseteq Q\) is the **set of accept states**.
A nondeterministic 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,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.
The formal description of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0, 1\}$,
3. $\delta$ is given as

\[
\begin{array}{c|ccc}
 & 0 & 1 & \varepsilon \\
\hline
q_1 & \{q_1\} & \{q_1, q_2\} & \emptyset \\
q_2 & \{q_3\} & \emptyset & \{q_3\} \\
q_3 & \emptyset & \{q_4\} & \emptyset \\
q_4 & \{q_4\} & \{q_4\} & \emptyset \\
\end{array}
\]

4. $q_1$ is the start state, and
5. $F = \{q_4\}$. 

\[
\begin{array}{c}
q_1 \\
\downarrow \varepsilon \\
q_2 \\
\downarrow 0 \\
q_3 \\
\downarrow 1 \\
q_4 \\
\end{array}
\] 

\[
\begin{array}{c}
q_1 \\
\downarrow 1 \\
q_2 \\
\downarrow 0 \\
q_3 \\
\downarrow 1 \\
q_4 \\
\end{array}
\] 

\[
\begin{array}{c}
q_1 \\
\downarrow \varepsilon \\
q_2 \\
\downarrow 0 \\
q_3 \\
\downarrow 1 \\
q_4 \\
\end{array}
\] 

\[
\begin{array}{c}
q_1 \\
\downarrow 0 \\
q_2 \\
\downarrow 1 \\
q_3 \\
\downarrow 0 \\
q_4 \\
\end{array}
\]
Running Programs, NFAs (JFLAP demo): \texttt{010110}
A nondeterministic machine can be in multiple states at the same time!

This is an accepting computation because at least one path ends in an accept state.
NFAs vs DFAs

**DFAs**
- Can only be in **one** state
- Transition:
  - Must read 1 char
- Acceptance:
  - If final state is accept state

**NFAs**
- Can be in **multiple** states
- Transition
  - Can read no chars
  - i.e., empty transition
- Acceptance:
  - If **one of** final states is accept state
Running an NFA Program: Formal Model

Define the extended transition function: \( \hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \)

- **Inputs:**
  - Some beginning state \( q \) (not necessarily the start state)
  - Input string \( w = w_1w_2 \cdots w_n \)
- **Output:**
  - Set of ending states

(Defined recursively)

- **Base case:** \( \hat{\delta}(q, \epsilon) = \{q\} \)
- **Recursive case:**
  - If: \( \hat{\delta}(q, w') = \{q_1, \ldots, q_k\} \)
    - where \( w' \in \Sigma^* = w_1 \cdots w_{n-1} \) and \( w_n \in \Sigma \)
  - Then: \( \hat{\delta}(q, w'w_n) = \bigcup_{i=1}^{k} \hat{\delta}(q_i, w_n) \)

No empty transitions
NFA Extended delta Example

\[ \hat{\delta}(q, \varepsilon) = \{q\} \]

\[ \hat{\delta}(q, w'w_n) = \bigcup_{i=1}^{k} \delta(q_i, w_n) \]

where \( w' \in \Sigma^* = w_1 \cdots w_{n-1} \)
and \( w_n \in \Sigma \)

\[ \hat{\delta}(q, w') = \{q_1, \ldots, q_k\} \]

- \( \hat{\delta}(q_0, \varepsilon) = \)
- \( \hat{\delta}(q_0, 0) = \)
- \( \hat{\delta}(q_0, 00) = \)
- \( \hat{\delta}(q_0, 001) = \)
Adding Empty Transitions

• Define the set $\varepsilon$-REACHABLE$(q)$
  • ... to be all states reachable from $q$ via one or more empty transitions

(Defined recursively)

• **Base case:** $q \in \varepsilon$-REACHABLE$(q)$

• **Inductive case:**
  
  \[
  \varepsilon$-REACHABLE$(q) = \{ r \mid p \in \varepsilon$-REACHABLE$(q)$ and $r \in \delta(p, \varepsilon) \} 
  \]

A state is in the reachable set if ...

... there is an empty transition to it from another state in the reachable set
$\varepsilon$-REACHABLE Example

$\varepsilon$-REACHABLE(1) = \{1, 2, 3, 4, 6\}
Running an NFA Program: Formal Model

Define the extended transition function:

\[ \hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \]

- **Inputs:**
  - Some beginning state \( q \) (not necessarily the start state)
  - Input string \( w = w_1w_2 \cdots w_n \)
- **Output:**
  - Set of ending states

(Defined recursively)

- **Base case:** \( \hat{\delta}(q, \epsilon) = \varepsilon\text{-REACHABLE}(q) \)
- **Recursive case:**
  - If:
    \[ \hat{\delta}(q, w') = \{q_1, \ldots, q_k\} \]
    where \( w' \in \Sigma^* = w_1 \cdots w_{n-1} \)
    and \( w_n \in \Sigma \)
  - Then:
    \[ \hat{\delta}(q, w'w_n) = \varepsilon\text{-REACHABLE}\left( \bigcup_{i=1}^{k} \delta(q_i, w_n) \right) \]
An NFA’s Language

• For NFA \( N = (Q, \Sigma, \delta, q_0, F) \) \( N \) accepts \( w \) if \( \hat{\delta}(q_0, w) \cap F \neq \emptyset \)
  • i.e., if the final states have at least one accept state

• Language of \( N = L(M) = \{ w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \} \)

• \( Q \): How does an NFA’s language relate to regular languages
  • Reminder: A language is regular if a DFA recognizes it
NFAs and Regular Languages

*Theorem:*  
• A language $A$ is regular if and only if some NFA $N$ recognizes it.
How to Prove a Theorem: $X \iff Y$

- $X \iff Y = \text{“}X \text{ if and only if } Y\text{”} = X \iff Y = X \iff Y$
- **Proof at minimum** has 2 parts:
  1. $\Rightarrow$ if $X$, then $Y$
     - i.e., assume $X$, then use it to prove $Y$
     - “forward” direction
  2. $\Leftarrow$ if $Y$, then $X$
     - i.e., assume $Y$, then use it to prove $X$
     - “reverse” direction
NFAs and Regular Languages

Theorem:
• A language $A$ is regular if and only if some NFA $N$ recognizes it.

Must prove:
• $\Rightarrow$ If $A$ is regular, then some NFA $N$ recognizes it
  • Easier
    • We know: if $A$ is regular, then a DFA recognizes it.
    • Easy to convert DFA to an NFA! (see HW2)
• $\Leftarrow$ If an NFA $N$ recognizes $A$, then $A$ is regular.
  • Harder
    • Idea: Convert NFA to DFA
How to convert NFA→DFA?

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the states,
2. \(\Sigma\) is a finite set called the alphabet,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

A nondeterministic 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. \(\delta: Q \times \Sigma_e \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

Proof idea:
Let each “state” of the DFA be a set of states in the NFA.
In a DFA, all these states at each step must be only one state.

So design a state in the DFA to be a set of NFA states!
Convert NFA→DFA, Formally

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

• An equivalent DFA $M$ has states $Q' = \mathcal{P}(Q)$ (power set of $Q$)
Example:

The NFA $N_4$

A DFA $D$ that is equivalent to the NFA $N_4$
NFA→DFA

Have: \( N = (Q, \Sigma, \delta, q_0, F') \)

Want to: construct a DFA \( M = (Q', \Sigma, \delta', q_0', F') \)

1. \( Q' = \mathcal{P}(Q) \). A state for \( M \) is a set of states in \( N \)

2. For \( R \in Q' \) and \( a \in \Sigma \),
   \[
   \delta'(R, a) = \bigcup_{r \in R} \delta(r, a)
   \]
   \( R = \) a state in \( M = \) a set of states in \( N \)

3. \( q_0' = \{q_0\} \)

4. \( F' = \{R \in Q' \mid R \text{ contains an accept state of } N\} \)

To compute next state for \( R \), compute next states of each NFA state \( r \) in \( R \), then union results into one set.
NFA→DFA  Proof of Correctness

• Let  \( N = (Q_N, \Sigma, \delta_N, q_0, F_N) \)

• And let NFA→DFA(\(N\)) = \( D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D) \)

• Correctness criteria: \( L(N) = L(D) \)

• We will prove a stronger statement: \( \hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w) \)
  • That is, for all strings \( w \), the DFA and NFA end in the same set of states
NFA→DFA  Proof of Correctness

• Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
• And let $\text{NFA→DFA}(N) = D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$

Theorem: \( \hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w) \)

Proof: (by induction on length of $w$)

• Base case $w = \epsilon$ \( \hat{\delta}_D(\{q_0\}, \epsilon) \) and \( \hat{\delta}_N(q_0, \epsilon) \) are \( \{q_0\} \)

• Inductive case $w = xa$
  • IH: \( \hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x) \), call this set of states $R$
  • NFA last step (from $\delta_N$ definition) \( \bigcup_{r \in R} \delta_N(r, a) \)
  • DFA last step (from NFA→DFA definition) \( \bigcup_{r \in R} \delta_N(r, a) \)

This produces a set bc of the definition of NFAs

Go back and review previous definitions to confirm

No empty transitions
NFA→DFA_ε

• Have: \( N = (Q, \Sigma, \delta, q_0, F) \)
• Want to: construct a DFA \( M = (Q', \Sigma, \delta', q_0', F') \)
1. \( Q' = \mathcal{P}(Q) \).
2. For \( R \in Q' \) and \( a \in \Sigma \),
   \[
   \delta'(R, a) = \bigcup_{r \in R} \varepsilon\text{-REACHABLE}(\delta(r, a))
   \]
3. \( q_0' = \{q_0\} \varepsilon\text{-REACHABLE}(\{q_0\}) \)
4. \( F' = \{ R \in Q' \mid R \text{ contains an accept state of } N \} \)
NFA→DFA$_\varepsilon$  Proof of Correctness

• Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
• And let NFA→DFA$_\varepsilon(N) = D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$

• **Correctness criteria:** $L(N) = L(D)$

• We will prove a stronger statement: $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$
  • That is, for all strings $w$, the DFA and NFA end in the same set of states

(Same as before)
NFA→DFA_{\varepsilon}  Proof of Correctness

• Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
• And let $\text{NFA→DFA}(N) = D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$

Theorem: $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$

Proof: (by induction on length of $w$)

• **Base case** $w = \varepsilon$  $\hat{\delta}_D(\{q_0\}, \varepsilon)$ and $\hat{\delta}_N(q_0, \varepsilon)$ are $\{q_0\}$

• **Inductive case** $w = xa$
  • **IH:** $\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x)$, call this set of states $R$
  • NFA last step (from $\delta_N$ definition) $\bigcup_{r \in R} \delta_N(r, a)$
  • DFA last step (from NFA→DFA definition) $\bigcup_{r \in R} \delta_N(r, a)$
Proving that NFAs Recognize Reg Langs

**Theorem:**

A language $A$ is regular if and only if some NFA $N$ recognizes it.

**Proof:**

$\Rightarrow$ If $A$ is regular, then some NFA $N$ recognizes it

- We know: if $A$ is regular, then a DFA recognizes it
- So convert DFA to an NFA

$\Leftarrow$ If an NFA $N$ recognizes $A$, then $A$ is regular

- We know: if a DFA recognizes a language, then it is regular
- So convert NFA to DFA ...
- ... Using NFA→DFA algorithm we just defined! ■ (Q.E.D.)
Combining DFAs

Problem 1: What should be the transition labels?

\[ M_3: \text{OR} \]

\[ q_0 \]

\[ \epsilon \]

\[ M_1: \text{Check special chars} \]

\[ \epsilon \]

\[ M_2: \text{Check uppercase} \]

Problem 2: Once we enter one of the machines, can’t go back to the other one!

This is an NFA!

Can be in multiple states at once, but is still equivalent to a regular language!

This allows us to check multiple machines (i.e., multiple machines) at once!
Next Time: More “Combining” Operations

Construction of $N$ to recognize $A_1 \circ A_2$
In-class Quiz 9/15

On gradescope