CMSC 330: Organization of Programming Languages

DFAs, and NFAs, and Regexps
The story so far, and what’s next

- **Goal:** Develop an algorithm that determines whether a string $s$ is matched by regex $R$
  - i.e., whether $s$ is a member of $R$’s *language*

- **Approach:** Convert $R$ to a *finite automaton* $FA$ and see whether $s$ is *accepted* by $FA$
  - Details: Convert $R$ to a *nondeterministic FA* (NFA), which we then convert to a *deterministic FA* (DFA),
    - which enjoys a fast acceptance algorithm
Two Types of Finite Automata

- **Deterministic Finite Automata (DFA)**
  - Exactly one sequence of steps for each string
    - Easy to implement acceptance check
  - All examples so far

- **Nondeterministic Finite Automata (NFA)**
  - May have many sequences of steps for each string
  - Accepts if any path ends in final state at end of string
  - More compact than DFA
    - But more expensive to test whether a string matches
Comparing DFAs and NFAs

- NFAs can have more than one transition leaving a state on the same symbol

- DFAs allow only one transition per symbol
  - i.e., transition function must be a valid function
  - DFA is a special case of NFA
Comparing DFAs and NFAs (cont.)

- NFAs may have transitions with empty string label
  - May move to new state without consuming character

- DFA transition must be labeled with symbol
  - DFA is a special case of NFA
DFA for \((a|b)^*abb\)
NFA for \((a|b)^{*}abb\)

- **ba**
  - Has paths to either **S0** or **S1**
  - Neither is final, so rejected

- **babaabb**
  - Has paths to different states
  - One path leads to **S3**, so accepts string
NFA for \((ab|aba)^*\)

- **aba**
  - Has paths to states S0, S1

- **ababa**
  - Has paths to S0, S1
  - Need to use \(\epsilon\)-transition
Comparing NFA and DFA for \((ab|aba)^*\)
Quiz 1: Which DFA matches this regexp?

\[ b(b|a+b?) \]

A. [Diagram of DFA 1]

B. [Diagram of DFA 2]

C. [Diagram of DFA 3]

D. None of the above
Quiz 1: Which DFA matches this regexp?

\[ b (b | a+b?) \]

A. 

B. 

C. 

D. None of the above
Formal Definition

- A deterministic finite automaton \((DFA)\) is a 5-tuple \((\Sigma, Q, q_0, F, \delta)\) where
  - \(\Sigma\) is an alphabet
  - \(Q\) is a nonempty set of states
  - \(q_0 \in Q\) is the start state
  - \(F \subseteq Q\) is the set of final states
  - \(\delta : Q \times \Sigma \rightarrow Q\) specifies the DFA's transitions

  What's this definition saying that \(\delta\) is?

- A DFA accepts \(s\) if it stops at a final state on \(s\)
Formal Definition: Example

- $\Sigma = \{0, 1\}$
- $Q = \{S0, S1\}$
- $q_0 = S0$
- $F = \{S1\}$

$\delta$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>S1</td>
</tr>
</tbody>
</table>

or as \{ $(S0,0,S0),(S0,1,S1),(S1,0,S0),(S1,1,S1)$ \}
Implementing DFAs (one-off)

It's easy to build a program which mimics a DFA

```
cur_state = 0;
while (1) {
    symbol = getchar();
    switch (cur_state) {
    case 0: switch (symbol) {
    case '0':  cur_state = 0; break;
    case '1':  cur_state = 1; break;
    case '\n': printf("rejected\n"); return 0;
    default:   printf("rejected\n"); return 0;
    } break;
    case 1: switch (symbol) {
    case '0':  cur_state = 0; break;
    case '1':  cur_state = 1; break;
    case '\n': printf("accepted\n"); return 1;
    default:   printf("rejected\n"); return 0;
    } break;
    default: printf("unknown state; I'm confused\n");
    break;
    }
}
```
Implementing DFAs (generic)

More generally, use generic table-driven DFA

```
given components (Σ, Q, q₀, F, δ) of a DFA:
let q = q₀
while (there exists another symbol σ of the input string)
    q := δ(q, σ);
if q ∈ F then
    accept
else reject
```

- q is just an integer
- Represent δ using arrays or hash tables
- Represent F as a set
An NFA is a 5-tuple \((\Sigma, Q, q_0, F, \delta)\) where
- \(\Sigma, Q, q_0, F\) as with DFAs
- \(\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q\) specifies the NFA's transitions

**Example**

- \(\Sigma = \{a\}\)
- \(Q = \{S1, S2, S3\}\)
- \(q_0 = S1\)
- \(F = \{S3\}\)
- \(\delta = \{(S1,a,S1), (S1,a,S2), (S2,\varepsilon,S3)\}\)

An NFA accepts \(s\) if there is at least one path via \(s\) from the NFA’s start state to a final state.
NFA Acceptance Algorithm (Sketch)

- When NFA processes a string $s$
  - NFA must keep track of several “current states”
    - Due to multiple transitions with same label, and $\epsilon$-transitions
  - If any current state is final when done then accept $s$
- Example
  - After processing “a”
    - NFA may be in states
      - $S1$
      - $S2$
      - $S3$
    - Since $S3$ is final, $s$ is accepted
- Algorithm is slow, space-inefficient; prefer DFAs!
Regular expressions, NFAs, and DFAs accept the same languages! *Can convert between them*

NB. Both *transform* and *reduce* are historical terms; they mean “convert”
Reducing Regular Expressions to NFAs

- Goal: Given regular expression \( A \), construct NFA: \(<A> = (\Sigma, Q, q_0, F, \delta)\)
  - Remember regular expressions are defined recursively from primitive RE languages
  - Invariant: \(|F| = 1\) in our NFAs
    - Recall \( F = \) set of final states

- Will define \(<A>\) for base cases: \( \sigma, \varepsilon, \emptyset \)
  - Where \( \sigma \) is a symbol in \( \Sigma \)

- And for inductive cases: \( AB, A|B, A^* \)
Reducing Regular Expressions to NFAs

Base case: $\sigma$

$<\sigma> = (\{\sigma\}, \{S0, S1\}, S0, \{S1\}, \{(S0, \sigma, S1)\} )$
Reduction

- **Base case: \( \varepsilon \)**

  \[ <\varepsilon> = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset) \]

- **Base case: \( \emptyset \)**

  \[ <\emptyset> = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset) \]
Reduction: Concatenation

Induction: \( AB \)

\[
\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)
\]

\[
\langle B \rangle = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)
\]
Reduction: Concatenation

- Induction: \( AB \)

\[
\begin{align*}
\langle A \rangle &= (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A) \\
\langle B \rangle &= (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B) \\
\langle AB \rangle &= (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, \{f_B\}, \delta_A \cup \delta_B \cup \{(f_A, \epsilon, q_B)\})
\end{align*}
\]
Reduction: Union

- Induction: \( A|B \)

\[
\begin{align*}
<A> & = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A) \\
<B> & = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)
\end{align*}
\]
Reduction: Union

- Induction: $A|B$

- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
- $<A|B> = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B \cup \{S0,S1\}, S0, \{S1\}, \delta_A \cup \delta_B \cup \{(S0,\epsilon,q_A), (S0,\epsilon,q_B), (f_A,\epsilon,S1), (f_B,\epsilon,S1)\))$
Reduction: Closure

Induction: $A^*$

- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
Reduction: Closure

Induction: $A^*$

- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<A^*> = (\Sigma_A, Q_A \cup \{S0, S1\}, S0, \{S1\}, \delta_A \cup \{(f_A, \epsilon, S1), (S0, \epsilon, q_A), (S0, \epsilon, S1), (S1, \epsilon, S0)\})$
Quiz 2: Which NFA matches $a^*$ ?
Quiz 2: Which NFA matches $a^*$?
Quiz 3: Which NFA matches \(a\mid b^*\)?
Quiz 3: Which NFA matches \( a|b^* \)?
Reduction Complexity

- Given a regular expression \( A \) of size \( n \)... 
  Size = # of symbols + # of operations

- How many states does \(<A>\) have?
  - Two added for each |, two added for each *
  - \( O(n) \)
  - That’s pretty good!
Reducing NFA to DFA
Reducing NFA to DFA

- NFA may be reduced to DFA
  - By explicitly tracking the set of NFA states

- Intuition
  - Build DFA where
    - Each DFA state represents a set of NFA “current states”

- Example
Algorithm for Reducing NFA to DFA

- Reduction applied using the subset algorithm
  - DFA state is a subset of set of all NFA states

- Algorithm
  - Input
    - NFA (Σ, Q, q₀, Fₙ, δ)
  - Output
    - DFA (Σ, R, r₀, Fₜ, δ)
  - Using two subroutines
    - \( \varepsilon \)-closure(δ, p) (and \( \varepsilon \)-closure(δ, Q))
    - move(δ, p, σ) (and move(δ, Q, σ))
      - (where p is an NFA state)
ε-transitions and ε-closure

We say $p \xrightarrow{\varepsilon} q$

- If it is possible to go from state $p$ to state $q$ by taking only ε-transitions in $\delta$
- If $\exists p, p_1, p_2, \ldots, p_n, q \in Q$ such that
  - $\{p, \varepsilon, p_1\} \in \delta$, $\{p_1, \varepsilon, p_2\} \in \delta$, \ldots, $\{p_n, \varepsilon, q\} \in \delta$

ε-closure($\delta$, $p$)

- Set of states reachable from $p$ using ε-transitions alone
  - Set of states $q$ such that $p \xrightarrow{\varepsilon} q$ according to $\delta$
  - $\varepsilon$-closure($\delta$, $p$) = \{ $q$ | $p \xrightarrow{\varepsilon} q$ in $\delta$ \}
  - $\varepsilon$-closure($\delta$, $Q$) = \{ $q$ | $p \in Q$, $p \xrightarrow{\varepsilon} q$ in $\delta$ \}

Notes

- $\varepsilon$-closure($\delta$, $p$) always includes $p$
- We write $\varepsilon$-closure($p$) or $\varepsilon$-closure($Q$) when $\delta$ is clear from context
ε-closure: Example 1

Following NFA contains
- $p_1 \xrightarrow{\varepsilon} p_2$
- $p_2 \xrightarrow{\varepsilon} p_3$
- $p_1 \xrightarrow{\varepsilon} p_3$

Since $p_1 \xrightarrow{\varepsilon} p_2$ and $p_2 \xrightarrow{\varepsilon} p_3$

ε-closures
- $\varepsilon$-closure($p_1$) = \{ $p_1$, $p_2$, $p_3$ \}
- $\varepsilon$-closure($p_2$) = \{ $p_2$, $p_3$ \}
- $\varepsilon$-closure($p_3$) = \{ $p_3$ \}
- $\varepsilon$-closure( \{ $p_1$, $p_2$ \} ) = \{ $p_1$, $p_2$, $p_3$ \} \cup \{ $p_2$, $p_3$ \}
**ε-closure: Example 2**

- **Following NFA contains**
  - \( p_1 \xrightarrow{\epsilon} p_3 \)
  - \( p_3 \xrightarrow{\epsilon} p_2 \)
  - \( p_1 \xrightarrow{\epsilon} p_2 \)
  
  - Since \( p_1 \xrightarrow{\epsilon} p_3 \) and \( p_3 \xrightarrow{\epsilon} p_2 \)

- **ε-closures**
  - \( \epsilon\text{-closure}(p_1) = \{ p_1, p_2, p_3 \} \)
  - \( \epsilon\text{-closure}(p_2) = \{ p_2 \} \)
  - \( \epsilon\text{-closure}(p_3) = \{ p_2, p_3 \} \)
  - \( \epsilon\text{-closure}(\{ p_2, p_3 \}) = \{ p_2 \} \cup \{ p_2, p_3 \} \)
ε-closure Algorithm: Approach

**Input:** NFA ($\Sigma$, $Q$, $q_0$, $F_n$, $\delta$), State Set $R$

**Output:** State Set $R'$

**Algorithm**

Let $R' = R$  // start states

Repeat

Let $R = R'$  // continue from previous

Let $R' = R \cup \{q \mid p \in R, (p, \varepsilon, q) \in \delta\}$  // new ε-reachable states

Until $R = R'$  // stop when no new states

This algorithm computes a **fixed point**
Calculate $\varepsilon$-closure$(d, \{p1\})$

- $R = \{p1\}$
- $R' = \{p1\}$
- $R = \{p1\}$
- $R' = \{p1, p2\}$
- $R = \{p1, p2\}$
- $R' = \{p1, p2, p3\}$
- $R = \{p1, p2, p3\}$
- $R' = \{p1, p2, p3\}$

Let $R' = R$

Repeat

- Let $R = R'$
- Let $R' = R \cup \{q \mid p \in R, (p, \varepsilon, q) \in \delta\}$

Until $R = R'$
Calculating move(p, σ)

- move(δ, p, σ)
  - Set of states reachable from p using exactly one transition on symbol σ
    - Set of states q such that \{p, σ, q\} ∈ δ
    - move(δ, p, σ) = \{q | \{p, σ, q\} ∈ δ\}
    - move(δ, Q, σ) = \{q | p ∈ Q, \{p, σ, q\} ∈ δ\}
      - i.e., can “lift” move() to a set of states Q

- Notes:
  - move(δ, p, σ) is Ø if no transition (p, σ, q) ∈ δ, for any q
  - We write move(p, σ) or move(R, σ) when δ clear from context
move(\(p, \sigma\)) : Example 1

- Following NFA
  - \(\Sigma = \{a, b\}\)

- Move
  - \(\text{move}(p_1, a) = \{p_2, p_3\}\)
  - \(\text{move}(p_1, b) = \emptyset\)
  - \(\text{move}(p_2, a) = \emptyset\)
  - \(\text{move}(p_2, b) = \{p_3\}\)
  - \(\text{move}(p_3, a) = \emptyset\)
  - \(\text{move}(p_3, b) = \emptyset\)

\[\text{move}({p_1, p_2}, b) = \{p_3\}\]
move(p,σ) : Example 2

- Following NFA
  - $\Sigma = \{a, b\}$

- Move
  - $\text{move}(p1, a) = \{p2\}$
  - $\text{move}(p1, b) = \{p3\}$
  - $\text{move}(p2, a) = \{p3\}$
  - $\text{move}(p2, b) = \emptyset$
  - $\text{move}(p3, a) = \emptyset$
  - $\text{move}(p3, b) = \emptyset$

$\text{move}(\{p1,p2\},a) = \{p2,p3\}$
NFA → DFA Reduction Algorithm ("subset")

- **Input**: NFA ($\Sigma$, $Q$, $q_0$, $F_n$, $\delta$), Output DFA ($\Sigma$, $R$, $r_0$, $F_d$, $\delta'$)
- **Algorithm**

  Let $r_0 = \varepsilon$-closure($\delta,q_0$), add it to $R$ // DFA start state

  While $\exists$ an unmarked state $r \in R$ // process DFA state $r$
    Mark $r$ // each state visited once
    For each $\sigma \in \Sigma$ // for each symbol $\sigma$
      Let $E = \text{move}(\delta,r,\sigma)$ // states reached via $\sigma$
      Let $e = \varepsilon$-closure($\delta,E$) // states reached via $\varepsilon$
      If $e \notin R$ // if state $e$ is new
        Let $R = R \cup \{e\}$ // add $e$ to $R$ (unmarked)
        Let $\delta' = \delta' \cup \{r, \sigma, e\}$ // add transition $r \rightarrow e$ on $\sigma$
    Let $F_d = \{r | \exists s \in r \text{ with } s \in F_n\}$ // final if include state in $F_n$
NFA → DFA Example 1

- Start = $\varepsilon$-closure($\delta$,p1) = { {p1,p3} }
- R = { {p1,p3} }
- $r \in R = \{p1,p3\}$
- move($\delta$, {p1,p3}, a) = {p2}
  - $e = \varepsilon$-closure($\delta$, {p2}) = {p2}
  - R = R $\cup$ {{p2}} = { {p1,p3}, {p2} }
  - $\delta' = \delta' \cup \{\{p1,p3\}, a, \{p2\}\}$
- move($\delta$, {p1,p3}, b) = $\emptyset$

DFA

- $\{1,3\} \rightarrow \{2\}$

NFA

- p1 $\rightarrow$ a p2 $\rightarrow$ b p3 $\rightarrow$ $\varepsilon$
NFA $\rightarrow$ DFA Example 1 (cont.)

- $R = \{ \{p1, p3\}, \{p2\} \}$
- $r \in R = \{p2\}$
- $\text{move}(\delta, \{p2\}, a) = \emptyset$
- $\text{move}(\delta, \{p2\}, b) = \{p3\}$
  - $e = \varepsilon$-closure($\delta, \{p3\}$) = $\{p3\}$
  - $R = R \cup \{\{p3\}\} = \{\{p1, p3\}, \{p2\}, \{p3\}\}$
  - $\delta' = \delta' \cup \{\{p2\}, b, \{p3\}\}$
NFA → DFA Example 1 (cont.)

- \( R = \{ \{p_1,p_3\}, \{p_2\}, \{p_3\} \} \)
- \( r \in R = \{p_3\} \)
- \( \text{Move}(\{p_3\}, a) = \emptyset \)
- \( \text{Move}(\{p_3\}, b) = \emptyset \)
- Mark \( \{p_3\} \), exit loop
- \( F_d = \{\{p_1,p_3\}, \{p_3\}\} \)
  - Since \( p_3 \in F_n \)
- Done!
NFA $\rightarrow$ DFA Example 2

- NFA

- DFA

$R = \{ \{A\}, \{B,D\}, \{C,D\} \}$
NFA:

$$\begin{align*}
p0 &\rightarrow a \rightarrow p1 \\
p1 &\rightarrow b \rightarrow p2 \\
p0 &\rightarrow \varepsilon \rightarrow p1
\end{align*}$$
NFA → DFA Example 3

NFA

\[ R = \{ \{A, E\}, \{B, D, E\}, \{C, D\}, \{E\} \} \]
NFA → DFA Example
NFA → DFA Example
NFA → DFA Practice
NFA → DFA Practice
Subset Algorithm as a Fixed Point

Input: NFA \((\Sigma, Q, q_0, F, \delta)\)

Output: DFA \(M'\)

Algorithm

Let \(q_0' = \varepsilon\text{-closure}(\delta, q_0)\)

Let \(F' = \{q_0'\}\) if \(q_0' \cap F \neq \emptyset\), or \(\emptyset\) otherwise

Let \(M' = (\Sigma, \{q_0'\}, q_0', F', \emptyset)\) \hspace{1cm} \text{// starting approximation of DFA}

Repeat

Let \(M = M'\) \hspace{1cm} \text{// current DFA approx}

For each \(q \in \text{states}(M), \sigma \in \Sigma\) \hspace{1cm} \text{// for each DFA state q and symb } \sigma

Let \(s = \varepsilon\text{-closure}(\delta, \text{move}(\delta, q, \sigma))\) \hspace{1cm} \text{// new subset from q}

Let \(F' = \{s\}\) if \(s \cap F \neq \emptyset\), or \(\emptyset\) otherwise, \hspace{1cm} \text{// subset contains final?}

\(M' = M' \cup (\emptyset, \{s\}, \emptyset, F', \{(q, \sigma, s)\})\) \hspace{1cm} \text{// update DFA}

Until \(M' = M\) \hspace{1cm} \text{// reached fixed point}