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 to come:** 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
  - (Almost) 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
  - 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
  - A DFA is a specific kind 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 \((ab|aba)^*\)

- \textbf{aba}
- \textbf{ababa}
  - Has paths to states S0, S1
  - Need to use \(\varepsilon\)-transition
NFA and DFA for \((ab|aba)^*\)
Quiz 1: Which string is NOT accepted by this NFA?

A. ab
B. abaa
C. abab
D. abaab
Quiz 1: Which string is NOT accepted by this NFA?

A. ab
B. abaa
C. abab
D. abaab
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

- $\Sigma = \{0, 1\}$
- $Q = \{S_0, S_1\}$
- $q_0 = S_0$
- $F = \{S_1\}$
- $\delta =$

<table>
<thead>
<tr>
<th>symbol</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

```c
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");
    }
}
```

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

[Diagram of a DFA with states S0 and S1, transitions labeled with 0 and 1]
Implementing DFAs (generic)

More generally, use generic table-driven DFA

given components $(\Sigma, Q, q_0, F, \delta)$ of a DFA:

- let $q = q_0$
- while (there exists another symbol $\sigma$ of the input string)
  - $q := \delta(q, \sigma)$;
- if $q \in F$ then
  - accept
- else reject

- $q$ is just an integer
- Represent $\delta$ 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

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

**Example**

- \(\Sigma = \{a\}\)
- \(Q = \{S_1, S_2, S_3\}\)
- \(q_0 = S_1\)
- \(F = \{S_3\}\)
- \(\delta = \{ (S_1,a,S_1), (S_1,a,S_2), (S_2,\varepsilon,S_3) \}\)
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!
Relating REs to DFAs and NFAs

- Regular expressions, NFAs, and DFAs accept the same languages! *Can convert between them*
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:** $\sigma$

Recall: NFA is $(\Sigma, Q, q_0, F, \delta)$ where
- $\Sigma$ is the alphabet
- $Q$ is set of states
- $q_0$ is starting state
- $F$ is set of final states
- $\delta$ is transition relation

$\langle \sigma \rangle = (\{\sigma\}, \{S0, S1\}, S0, \{S1\}, \{(S0, \sigma, S1)\})$

$(\Sigma, Q, q_0, F, \delta)$
Reduction

- Base case: $\varepsilon$
  
  $<\varepsilon> = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset)$

- Base case: $\emptyset$
  
  $<\emptyset> = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)$

Recall: NFA is $(\Sigma, Q, q_0, F, \delta)$
  where
  - $\Sigma$ is the alphabet
  - $Q$ is set of states
  - $q_0$ is starting state
  - $F$ is set of final states
  - $\delta$ is transition relation
Reduction: Concatenation

- Induction: $AB$

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

- $<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
Reduction: Concatenation

- Induction: $AB$

- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
- $<AB> = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, \{f_B\}, \delta_A \cup \delta_B \cup \{(f_A, \varepsilon, q_B)\})$
Reduction: Union

- Induction: $A|B$

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

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

- Induction: $A|B$

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

- **Induction:** $A^*$

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

- Induction: $A^*$

\[
\begin{align*}
\langle A \rangle &= (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A) \\
\langle A^* \rangle &= (\Sigma_A, Q_A \cup \{S0,S1\}, S0, \{S1\}, \\
&\quad \delta_A \cup \{(f_A,\varepsilon,S1), (S0,\varepsilon,q_A), (S0,\varepsilon,S1), (S1,\varepsilon,S0)\})
\end{align*}
\]
Quiz 2: Which NFA matches $a^*$?
Quiz 2: Which NFA matches $a^*$?
Quiz 3: Which NFA matches $a|b^*$?
Quiz 3: Which NFA matches $a|b^*$ ?

A.

B.

C.

D.
RE $\rightarrow$ NFA

Draw NFAs for the regular expression $(0|1)^*110^*$
Recap

- Finite automata
  - Alphabet, states…
  - \((\Sigma, Q, q_0, F, \delta)\)

- Types
  - Deterministic (DFA)
  - Non-deterministic (NFA)

- Reducing RE to NFA
  - Concatenation
  - Union
  - Closure
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

DFA ← NFA

RE can reduce DFA

RE can reduce NFA
Why NFA → DFA

- DFA is generally more efficient than NFA

Language: \((a|b)^*ab\)
Why NFA → DFA

- DFA has the same expressive power as NFAs.
  - Let language $L \subseteq \Sigma^*$, and suppose $L$ is accepted by NFA $N = (\Sigma, Q, q_0, F, \delta)$. There exists a DFA $D = (\Sigma, Q', q'_0, F', \delta')$ that also accepts $L$. ($L(N) = L(D)$)

- NFAs are more flexible and easier to build. But DFAs have no less power than NFAs.
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 ($\Sigma, Q, q_0, F_n, \delta$)
  - Output
    - DFA ($\Sigma, R, r_0, F_d, \delta$)
  - Using two subroutines
    - $\varepsilon$-closure($\delta, p$) (and $\varepsilon$-closure($\delta, Q$))
    - move($\delta, p, \sigma$) (and move($\delta, Q, \sigma$))
      - (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 , $\{p_n, \varepsilon, q\} \in \delta$

- $\varepsilon$-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($\delta$, $p$) = $\{q | p \xrightarrow{\varepsilon} q \text{ in } \delta \}$
    - $\varepsilon$-closure($\delta$, $Q$) = $\{ q | p \in Q, p \xrightarrow{\varepsilon} q \text{ in } \delta \}$
  - Notes
    - $\varepsilon$-closure($\delta$, $p$) always includes $p$
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( $\{ p_1, p_2 \}$ ) = $\{ p_1, p_2, p_3 \} \cup \{ p_2, p_3 \}$
\( \varepsilon \)-closure: Example 2

- Following NFA contains
  - \( p1 \xrightarrow{\varepsilon} p3 \)
  - \( p3 \xrightarrow{\varepsilon} p2 \)
  - \( p1 \xrightarrow{\varepsilon} p2 \)

- \( \varepsilon \)-closures
  - \( \varepsilon\)-closure\((p1) = \{ p1, p2, p3 \} \)
  - \( \varepsilon\)-closure\((p2) = \{ p2 \} \)
  - \( \varepsilon\)-closure\((p3) = \{ p2, p3 \} \)
  - \( \varepsilon\)-closure\(( \{ p2, p3 \} ) = \{ p2 \} \cup \{ p2, p3 \} \)
**ε-closure Algorithm: Approach**

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

**Algorithm**

1. Let \(R' = R\)  
   // start states
2. Repeat
   1. Let \(R = R'\)  
      // continue from previous
   2. Let \(R' = R \cup \{q \mid p \in R, (p, \varepsilon, q) \in \delta\}\)  
      // new \(\varepsilon\)-reachable states
3. Until \(R = R'\)  
   // stop when no new states

This algorithm computes a **fixed point**
**ε-closure Algorithm Example**

- **Calculate** $\epsilon$-closure($\delta, \{p1\}$)

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

Let $R' = R$
Repeat
  - Let $R = R'$
  - Let $R' = R \cup \{q \mid p \in R, (p, \epsilon, q) \in \delta\}$
Until $R = R'$
Calculating move(p, \sigma)

move(\delta, p, \sigma)

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

- Notes:
  - move(\delta, p, \sigma) is \emptyset if no transition (p, \sigma, q) \in \delta, for any q
**move(p, σ) : Example 1**

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

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

*move({p1,p2},b) = { p3 }*
move(p, σ) : Example 2

Following NFA

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

Move

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

\( \text{move}\{p_1, p_2\}, a) = \{p_2, p_3\} \)
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 $R$  
  \hspace{1cm} // DFA start state

  While $\exists$ an unmarked state $r \in R$  
  \hspace{1cm} // process DFA state $r$

    Mark $r$  
    \hspace{1cm} // each state visited once

    For each $\sigma \in \Sigma$  
    \hspace{1cm} // for each symbol $\sigma$

      Let $E = \text{move}(\delta,r,\sigma)$  
      \hspace{1cm} // states reached via $\sigma$

      Let $e = \varepsilon$-closure($\delta,E$)  
      \hspace{1cm} // states reached via $\varepsilon$

      If $e \notin R$  
      \hspace{1cm} // if state $e$ is new

        Let $R = R \cup \{e\}$  
        \hspace{1cm} // add $e$ to $R$ (unmarked)

        Let $\delta' = \delta' \cup \{r, \sigma, e\}$  
        \hspace{1cm} // add transition $r \rightarrow e$ on $\sigma$

    Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$  
    \hspace{1cm} // final if include state in $F_n$
NFA → DFA Example

• 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) = \ø
NFA → DFA Example (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 (cont.)

- \( R = \{ \{p1,p3\}, \{p2\}, \{p3\} \} \)
- \( r \in R = \{p3\} \)
- \( \text{Move}([p3],a) = \emptyset \)
- \( \text{Move}([p3],b) = \emptyset \)
- Mark \( \{p3\} \), exit loop
- \( F_d = \{\{p1,p3\}, \{p3\}\} \)
  - Since \( p3 \in F_n \)
- Done!
NFA → DFA Example 2

NFA

DFA

\[ R = \{ \{A\}, \{B, D\}, \{C, D\} \} \]
Quiz 4: Which DFA is equiv to this NFA?

NFA:

A.  

B.  

C.  

D. None of the above
Quiz 4: Which DFA is equivalent to this NFA?

NFA:

A. Rejects ε

B. Rejects ε

C. Accepts abb

D. None of the above
Actual Answer

NFA:

**NFA:**

- States: p0, p1, p2
- Transitions:
  - p0 → p1 on input a
  - p1 → p2 on input b
  - p0 → p1 on input ε
  - p2, p0 → p1 on input a
  - p1, p0 → p2 on input b
  - p1, p0 → p1 on input a
NFA → DFA Example 3

NFA

DFA

\[ R = \{ \{A, E\}, \{B, D, E\}, \{C, D\}, \{E\} \} \]