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

```
ε-transition
```

- 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 \( \varepsilon \)-transition
Comparing NFA and DFA for \((ab|aba)^*\)
Quiz 1: Which DFA matches this regexp?

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

A.

\[ \begin{array}{c}
0 \xrightarrow{b} 1 \\
1 \xrightarrow{b} 2 \\
2 \xrightarrow{b} 3 \\
3 \xrightarrow{b} 4 \\
4 \xrightarrow{a} 3
\end{array} \]

B.

\[ \begin{array}{c}
0 \xrightarrow{b} 1 \\
1 \xrightarrow{b} 2 \\
2 \xrightarrow{a} 3 \\
3 \xrightarrow{a} 2
\end{array} \]

C.

\[ \begin{array}{c}
0 \xrightarrow{b} 1 \\
1 \xrightarrow{a} 2
\end{array} \]

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

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

<table>
<thead>
<tr>
<th>input state</th>
<th>symbol</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td>S0</td>
<td>1</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td>S1</td>
<td>1</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");
                        break;
    }
}
```

CMSC 330 Fall 2019
Implementing DFAs (generic)

More generally, use generic table-driven DFA

```plaintext
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
Nondeterministic Finite Automata (NFA)

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 $\varepsilon$-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*

NB. Both *transform* and *reduce* are historical terms; they mean “convert”

CMSC 330 Fall 2019
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 $F = \text{set of final states}$

- 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:** $\sigma$

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

  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

- **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:** $AB$

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

Induction: $A | B$

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

Induction: \( A \mid B \)

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

- Induction: $A^*$

- $\langle A \rangle = (\Sigma, Q_A, q_A, \{f_A\}, \delta_A)$
Reduction: Closure

- **Induction: \( A^* \)**

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

A.

B.

C.

D.
Quiz 2: Which NFA matches $a^*$?
Quiz 3: Which NFA matches $a|b^*$?
Quiz 3: Which NFA matches $a|b^*$?
Draw NFAs for the regular expression \((0|1)^*110^*\)
Draw NFAs for the regular expression \((ab^*c|d^*a|ab)d\)
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!
Recap

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

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

Reducing RE to NFA
- Concatenation
- Union
- Closure
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

![Diagram showing NFA and DFA transitions]

NFA: S1 → S2 (a), S2 → S3 (ε)
DFA: S1 → S1, S2, S3 (a)
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
    - $\epsilon$-closure($\delta$, p) (and $\epsilon$-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
  - \( \{p,\varepsilon,p_1\} \in \delta \)
  - \( \{p_1,\varepsilon,p_2\} \in \delta \)
  - \( \ldots \)
  - \( \{p_n,\varepsilon,q\} \in \delta \)

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 \( \delta \)
  - \( \varepsilon\text{-closure}(\delta, p) = \{ q \mid p \xrightarrow{\varepsilon} q \text{ in } \delta \} \)
  - \( \varepsilon\text{-closure}(\delta, Q) = \{ q \mid p \in Q, p \xrightarrow{\varepsilon} q \text{ in } \delta \} \)

Notes

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

Following NFA contains

- p1 $\xrightarrow{\varepsilon}$ p2
- p2 $\xrightarrow{\varepsilon}$ p3
- p1 $\xrightarrow{\varepsilon}$ p3

- Since p1 $\xrightarrow{\varepsilon}$ p2 and p2 $\xrightarrow{\varepsilon}$ p3

ε-closures

- ε-closure(p1) = \{ p1, p2, p3 \}
- ε-closure(p2) = \{ p2, p3 \}
- ε-closure(p3) = \{ p3 \}
- ε-closure( \{ p1, p2 \} ) = \{ p1, p2, p3 \} \cup \{ p2, p3 \}
**ε-closure: Example 2**

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

- **ε-closures**
  - $\text{ε-closure}(p_1) = \{ p_1, p_2, p_3 \}$
  - $\text{ε-closure}(p_2) = \{ p_2 \}$
  - $\text{ε-closure}(p_3) = \{ p_2, p_3 \}$
  - $\text{ε-closure}(\{ p_2, p_3 \}) = \{ p_2 \} \cup \{ p_2, p_3 \}$
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 \(\varepsilon\)-reachable states

Until \(R = R'\) // stop when no new states

This algorithm computes a fixed point
ε-closure Algorithm Example

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

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

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

Let $R' = R$
Repeat
  - Let $R = R'$
  - Let $R' = R \cup \{q | p \in R, (p, \epsilon, q) \in \delta\}$
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, σ) : Example 1

- **Following NFA**
  - Σ = \{ a, b \}

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

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

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

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

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 $R$  // 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 $e \not\in R$  // 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) = $\emptyset$
NFA $\rightarrow$ DFA Example 1 (cont.)

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

NFA

DFA
NFA $\rightarrow$ DFA Example 1 (cont.)

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

NFA

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

NFA:

A.

B.

C.

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

NFA:

A.

B.

C.

D. None of the above
Actual Answer

NFA:
NFA → DFA Example 3

NFA

DFA

\[ R = \{ \{A, E\}, \{B, D, E\}, \{C, D\}, \{E\} \} \]
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 \(q_0' = \epsilon\)-closure\((\delta, q_0)\)

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

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

Repeat

Let \(M = M'\)  // current DFA approx

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

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

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

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

Until \(M' = M\)  // reached fixed point
Redux: NFA to DFA Example 1

- $q'_0 = \varepsilon\text{-closure}(\delta, p1) = \{p1, p3\}$
- $F' = \{\{p1, p3\}\}$ since $\{p1, p3\} \cap \{p3\} \neq \emptyset$

- $M' = \{\Sigma, \{\{p1, p3\}\}, \{p1, p3\}, \{\{p1, p3\}\}, \emptyset\}$

NFA

DFA

$Q'$  $q'_0$  $F'$  $\delta'$

$\{1,3\}$
Redux: NFA to DFA Example 1 (cont)

- \( M' = \{ \Sigma, \{\{p1,p3\}\}, \{p1,p3\}, \{\{p1,p3\}\}, \emptyset \} \)
- \( q = \{p1, p3\} \)
- \( a = a \)
- \( s = \{p2\} \)
  - since \( \text{move}(\delta,\{p1, p3\},a) = \{p2\} \)
  - and \( \varepsilon\)-closure\((\delta,\{p2\}) = \{p2\} \)
- \( F' = \emptyset \)
  - Since \( \{p2\} \cap \{p3\} = \emptyset \)
  - where \( s = \{p2\} \) and \( F = \{p3\} \)

\[ M' = M' \cup ( \emptyset, \{[p2]\}, \emptyset, \emptyset, \{([p1,p3],a,[p2])\} ) \]

\[ = \{ \Sigma, \{\{p1,p3\},[p2]\}, \{p1,p3\}, \{\{p1,p3\}\}, \{([p1,p3],a,[p2])\} \} \]
Redux: NFA to DFA Example 1 (cont)

- $M' = \{ \Sigma, \{\{S1,S3\},\{S2\}\}, \{S1,S3\}, \{\{S1,S3\}\}, \{((S1,S3), a, \{S2\})\} \} $
  - $q = \{S2\}$
  - $a = b$
  - $s = \{S3\}$
    - since $\text{move}(\delta, \{S2\}, b) = \{S3\}$
    - and $\varepsilon$-closure$(\delta, \{S3\}) = \{S3\}$
  - $F' = \{\{S3\}\}$
    - Since $\{S3\} \cap \{S3\} = \{S3\}$
    - where $s = \{S3\}$ and $F = \{S3\}$

- $M' = M' \cup \left( \emptyset, \{S3\}, \emptyset, \{\{S3\}\}, \{((S2), b, \{S3\})\} \right)
  = \{ \Sigma, \{\{S1,S3\}, \{S2\}, \{S3\}\}, \{S1,S3\}, \{\{S1,S3\}, \{S3\}\}, \{((S1,S3), a, \{S2\}), ((S2), b, \{S3\})\} \}$

NFA

DFA
Analyzing the Reduction

- Can reduce any NFA to a DFA using subset alg.
- How many states in the DFA?
  - Each DFA state is a subset of the set of NFA states
  - Given NFA with $n$ states, DFA may have $2^n$ states
    - Since a set with $n$ items may have $2^n$ subsets
  - Corollary
    - Reducing a NFA with $n$ states may be $O(2^n)$
Recap: Matching a Regexp $R$

- Given $R$, construct NFA. Takes time $O(|R|)$
- Convert NFA to DFA. Takes time $O(2^{|R|})$
  - But usually not the worst case in practice
- Use DFA to accept/reject string $s$
  - Assume we can compute $\delta(q,\sigma)$ in constant time
  - Then time to process $s$ is $O(|s|)$
    - Can’t get much faster!
- Constructing the DFA is a one-time cost
  - But then processing strings is fast
Regular Expressions in Practice

- Regular expressions are typically “compiled” into tables for the generic algorithm
  - Can think of this as a simple bytecode interpreter
  - But really just a representation of \((\Sigma, Q_A, q_A, \{f_A\}, \delta_A)\), the components of the DFA produced from the RE

- Regular expression implementations often have extra constructs that are non-regular
  - I.e., can accept more than the regular languages
  - Can be useful in certain cases
  - Disadvantages
    - Nonstandard, plus can have higher complexity
Closing the Loop: Reducing DFA to RE

DFA \rightarrow NFA

DFA \leftarrow \text{reduce} \rightarrow \text{NFA}

DFA \leftarrow \text{transform} \rightarrow \text{NFA}

DFA \leftarrow \text{transform} \rightarrow \text{RE}

DFA \leftarrow \text{reduce} \rightarrow \text{NFA}

DFA \leftarrow \text{transform} \rightarrow \text{RE}

CMSC 330 Fall 2019
Reducing DFAs to REs

- **General idea**
  - Remove states one by one, labeling transitions with regular expressions
  - When two states are left (start and final), the transition label is the regular expression for the DFA
DFA to RE example

Language over $\Sigma = \{0,1\}$ such that every string is a multiple of 3 in binary

\[
(0 + 1(0 \ 1^* \ 0)1)^* 
\]
Other Topics

- Minimizing DFA
  - Hopcroft reduction
- Complementing DFA
Minimizing DFAs

- Every regular language is recognizable by a unique minimum-state DFA
  - Ignoring the particular names of states
- In other words
  - For every DFA, there is a unique DFA with minimum number of states that accepts the same language
Minimizing DFA: Hopcroft Reduction

Intuition
- Look to distinguish states from each other
  - End up in different accept / non-accept state with identical input

Algorithm
- Construct initial partition
  - Accepting & non-accepting states
- Iteratively split partitions (until partitions remain fixed)
  - Split a partition if members in partition have transitions to different partitions for same input
    - Two states $x$, $y$ belong in same partition if and only if for all symbols in $\Sigma$ they transition to the same partition
- Update transitions & remove dead states

J. Hopcroft, “An n log n algorithm for minimizing states in a finite automaton,” 1971
Splitting Partitions

- No need to split partition \{S,T,U,V\}
  - All transitions on \(a\) lead to identical partition \(P2\)
  - Even though transitions on \(a\) lead to different states
Splitting Partitions (cont.)

- Need to split partition \{S,T,U\} into \{S,T\}, \{U\}
  - Transitions on \(a\) from \(S,T\) lead to partition \(P_2\)
  - Transition on \(a\) from \(U\) lead to partition \(P_3\)
Resplitting Partitions

- Need to reexamine partitions after splits
  - Initially no need to split partition \{S,T,U\}
  - After splitting partition \{X,Y\} into \{X\}, \{Y\} we need to split partition \{S,T,U\} into \{S,T\}, \{U\}

![Diagram showing partitions and resplitting]
Minimizing DFA: Example 1

- DFA

- Initial partitions

- Split partition
Minimizing DFA: Example 1

- **DFA**

- **Initial partitions**
  - Accept  \{ R \}  = P1
  - Reject  \{ S, T \}  = P2

- **Split partition? → Not required, minimization done**
  - move(S,a) = T ∈ P2  \quad – \quad move(S,b) = R ∈ P1
  - move(T,a) = T ∈ P2  \quad – \quad move(T,b) = R ∈ P1
Minimizing DFA: Example 2

S
  ^  v
  |    |
  a    b
T
  v    v
  |    |
  b    a
R
Minimizing DFA: Example 2

- **DFA**

- **Initial partitions**
  - Accept \{ R \} = P1
  - Reject \{ S, T \} = P2

- **Split partition? → Yes, different partitions for B**
  - move(S,a) = T ∈ P2  -- move(S,b) = T ∈ P2
  - move(T,a) = T ∈ P2  -- move (T,b) = R ∈ P1

DFA already minimal
Minimizing DFA: Example 3
Minimizing DFA: Example 3
Given a DFA accepting language \( L \)
- How can we create a DFA accepting its complement?
- Example DFA
  - \( \Sigma = \{a, b\} \)
Complement of DFA

- Algorithm
  - Add explicit transitions to a dead state
  - Change every accepting state to a non-accepting state & every non-accepting state to an accepting state

- Note this only works with DFAs
  - Why not with NFAs?
Summary of Regular Expression Theory

- Finite automata
  - DFA, NFA
- Equivalence of RE, NFA, DFA
  - RE → NFA
    - Concatenation, union, closure
  - NFA → DFA
    - ε-closure & subset algorithm
- DFA
  - Minimization, complementation