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
  - 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
  - A DFA is a specific kind of NFA
DFA for \((a|b)^*abb\)
NFA for (a|b)*abb

- **ba**
  - Has paths to either $S_0$ or $S_1$
  - Neither is final, so rejected

- **babaabb**
  - Has paths to different states
  - One path leads to $S_3$, so accepts string
NFA for \((ab|aba)^*\)

- aba
- 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. \(aba\)
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

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

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

or as \{ (S0,0,S0), (S0,1,S1), (S1,0,S0), (S1,1,S1) \}
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;
    }
}

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

- \(\Sigma = \{a\}\)
- \(Q = \{S_1, S_2, S_3\}\)
- \(q_0 = S_1\)
- \(F = \{S_3\}\)
- \(\delta = \{(S_1,a,S_1), (S_1,a,S_2), (S_2,\varepsilon,S_3)\}\)

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”
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: $AB, A|B, A^*$
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: $\emptyset$

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

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: \( AB \)

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

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

\[
\langle AB \rangle = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, \{f_B\}, \delta_A \cup \delta_B \cup \{(f_A, \varepsilon, q_B)\})
\]
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:** \( A|B \)

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

- Induction: $A^*$

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

- Induction: $A^*$

- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<A^*> = (\Sigma_A, Q_A \cup \{S0,S1\}, S0, \{S1\},$
  $\delta_A \cup \{(f_A,\varepsilon,S1), (S0,\varepsilon,q_A), (S0,\varepsilon,S1), (S1,\varepsilon,S0)\})$
Quiz 2: Which NFA matches $a^*$ ?

A.

B.

C.

D.
Quiz 2: Which NFA matches $a^*$?
Quiz 3: Which NFA matches $a|b^*$ ?

A.

B.

C.

D.
Quiz 3: Which NFA matches $a|b^*$?
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

can reduce

RE

can reduce
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.

NFA ↔ DFA

CMSC 330 Fall 2021
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

```
S1  a  S2  ε  S3
     |     |     |
     a    a    ε
```

NFA

```
S1  a  S1, S2, S3
     |     |
     a    a
```

DFA
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 \( \varepsilon \)-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 \( \varepsilon \)-transitions alone
    - Set of states \( q \) such that \( p \xrightarrow{\varepsilon} q \) according to \( \delta \)
    - \( \varepsilon \)-closure(\( \delta \), \( p \)) = \{ \( q \mid p \xrightarrow{\varepsilon} q \) in \( \delta \) \}
    - \( \varepsilon \)-closure(\( \delta \), \( Q \)) = \{ \( q \mid p \in Q, p \xrightarrow{\varepsilon} q \) 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:
  - p₁ $\xrightarrow{\epsilon}$ p₂
  - p₂ $\xrightarrow{\epsilon}$ p₃
  - p₁ $\xrightarrow{\epsilon}$ p₃
  - Since p₁ $\xrightarrow{\epsilon}$ p₂ and p₂ $\xrightarrow{\epsilon}$ p₃

- ε-closures:
  - $\epsilon$-closure(p₁) = \{ p₁, p₂, p₃ \}
  - $\epsilon$-closure(p₂) = \{ p₂, p₃ \}
  - $\epsilon$-closure(p₃) = \{ p₃ \}
  - $\epsilon$-closure( \{ p₁, p₂ \} ) = \{ p₁, p₂, p₃ \} $\cup$ \{ p₂, p₃ \}
\(\epsilon\)-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 \(p_3 \xrightarrow{\epsilon} p_2\)

- \(\epsilon\)-closures:
  - \(\epsilon\)-closure\((p_1) =\) \(\{ p_1, p_2, p_3 \}\)
  - \(\epsilon\)-closure\((p_2) =\) \(\{ p_2 \}\)
  - \(\epsilon\)-closure\((p_3) =\) \(\{ p_2, p_3 \}\)
  - \(\epsilon\)-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 $R' = R \cup \{ q \mid p \in R, (p, \epsilon, q) \in \delta \}$  // new $\epsilon$-reachable states

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

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

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 $\text{move}(p, \sigma)$

- $\text{move}(\delta, p, \sigma)$
  - Set of states reachable from $p$ using exactly one transition on symbol $\sigma$
    - Set of states $q$ such that $\{p, \sigma, q\} \in \delta$
    - $\text{move}(\delta, p, \sigma) = \{ q \mid \{p, \sigma, q\} \in \delta \}$
    - $\text{move}(\delta, Q, \sigma) = \{ q \mid p \in Q, \{p, \sigma, q\} \in \delta \}$
      - i.e., can “lift” $\text{move}()$ to a set of states $Q$
  - Notes:
    - $\text{move}(\delta, p, \sigma)$ is $\emptyset$ if no transition $(p, \sigma, q) \in \delta$, for any $q$
    - We write $\text{move}(p, \sigma)$ or $\text{move}(R, \sigma)$ when $\delta$ clear from context
move(p, σ) : Example 1

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

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

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

- Following NFA
  - Σ = { a, b }

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

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 \(R\)  
  \(\quad\) // DFA start state

  While \(\exists\) an unmarked state \(r \in R\)  
  \(\quad\) // process DFA state \(r\)

  Mark \(r\)  
  \(\quad\) // each state visited once

  For each \(\sigma \in \Sigma\)  
  \(\quad\) // for each symbol \(\sigma\)

  Let \(E = \text{move}(\delta, r, \sigma)\)  
  \(\quad\) // states reached via \(\sigma\)

  Let \(e = \varepsilon\)-closure\((\delta, E)\)  
  \(\quad\) // states reached via \(\varepsilon\)

  If \(e \notin R\)  
  \(\quad\) // if state \(e\) is new

  Let \(R = R \cup \{e\}\)  
  \(\quad\) // add \(e\) to \(R\) (unmarked)

  Let \(\delta' = \delta' \cup \{r, \sigma, e\}\)  
  \(\quad\) // add transition \(r \rightarrow e\) on \(\sigma\)

  Let \(F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}\)  
  \(\quad\) // final if include state in \(F_n\)
NFA → DFA Example

- Start = $\varepsilon$-closure($\delta$, $p_1$) = { {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 → DFA Example (cont.)

- \( R = \{ \{p_1, p_3\}, \{p_2\} \} \)
- \( r \in R = \{p_2\} \)
- \( \text{move}(\delta, \{p_2\}, a) = \emptyset \)
- \( \text{move}(\delta, \{p_2\}, b) = \{p_3\} \)
  - \( e = \varepsilon\)-closure\((\delta, \{p_3\}) = \{p_3\} \)
  - \( R = R \cup \{\{p_3\}\} = \{ \{p_1, p_3\}, \{p_2\}, \{p_3\} \} \)
  - \( \delta' = \delta' \cup \{\{p_2\}, b, \{p_3\}\} \)

\[ \begin{align*}
\text{NFA} & \quad \text{DFA} \\
p_1 & \quad \{1, 3\} \\
p_2 & \quad \{2\} \\
p_3 & \quad \{3\}
\end{align*} \]
NFA → DFA Example (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

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?

- A.
- B.
- C.
- D. None of the above
Actual Answer

NFA:

- Start state: p0
- Transitions:
  - a from p0 to p1
  - b from p1 to p2
  - ε from p1 to p0
- Final states: p2, p0
### NFA → DFA Example 3

#### NFA

- States: $A, B, C, D, E$
- Transitions:
  - $A$: $a \rightarrow B$, $b \rightarrow C$, $\epsilon \rightarrow A$
  - $B$: $a \rightarrow D$, $\epsilon \rightarrow B$
  - $C$: $b \rightarrow A$, $\epsilon \rightarrow C$
  - $D$: $a \rightarrow D$, $\epsilon \rightarrow D$
  - $E$: $a \rightarrow A$

#### DFA

- States: $\{A, E\}, \{B, D, E\}, \{C, D\}, \{E\}$
- Transitions:
  - $\{A, E\}$: $a \rightarrow \{A, E\}$, $b \rightarrow \{C, D\}$
  - $\{B, D, E\}$: $a \rightarrow \{B, D, E\}$, $b \rightarrow \{C, D\}$
  - $\{C, D\}$: $b \rightarrow \{C, D\}$
  - $\{E\}$: $a \rightarrow \{E\}$

#### Transition Relational Set

$$R = \{ \{A, E\}, \{B, D, E\}, \{C, D\}, \{E\} \}$$
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \)

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \varepsilon\text{-closure}(\delta, E) \)

If \( e \notin R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \)
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \).

While \( \exists \) an unmarked state \( r \in R \):

1. Mark \( r \).
2. For each \( \sigma \in \Sigma \):
   - Let \( E = \text{move}(\delta, r, \sigma) \).
   - Let \( e = \varepsilon\text{-closure}(\delta, E) \).
   - If \( e \notin R \):
     - Let \( R = R \cup \{e\} \).
   - Let \( \delta' = \delta' \cup \{r, \sigma, e\} \).

Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \).
Let $r_0 = \varepsilon$-closure($\delta, q_0$), add it to $R$

While $\exists$ an unmarked state $r \in R$

   Mark $r$

   For each $\sigma \in \Sigma$

      Let $E = \text{move}(\delta, r, \sigma)$

      Let $e = \varepsilon$-closure($\delta, E$)

      If $e \notin R$

         Let $R = R \cup \{e\}$

         Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

---

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>${A, B, C}$</td>
<td>${B, C}$</td>
<td>${B, C}$</td>
</tr>
</tbody>
</table>
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \)

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \varepsilon\text{-closure}(\delta, E) \)

If \( e \notin R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \)
Let \( r_0 = \epsilon\text{-closure}(\delta,q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \)

//1

Let \( E = \text{move}(\delta,r,\sigma) \)

Let \( e = \epsilon\text{-closure}(\delta,E) \)

If \( e \notin R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>{B,C}</td>
<td></td>
</tr>
<tr>
<td>{B,C}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let $r_0 = \varepsilon$-closure($\delta, q_0$), add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$  //1

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon$-closure($\delta, E$)

If $e \not\in R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$
Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$ //1

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon\text{-closure}(\delta, E)$

If $e \notin R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$
Let $r_0 = \varepsilon$-closure($\delta,q_0$), add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$ //1

Let $E = \text{move}(\delta,r,\sigma)$

Let $e = \varepsilon$-closure($\delta,E$)

If $e \notin R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

---

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>{B,C}</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>{B,C}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

}{A,B,C} {B,C} {A,B,C}

{B,C}
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \) //0

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \varepsilon\text{-closure}(\delta, E) \)

If \( e \notin R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>{B,C}</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>{B,C}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \)

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \varepsilon\text{-closure}(\delta, E) \)

If \( e \not\in R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \)
Let $r_0 = \varepsilon$-closure($\delta, q_0$), add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$  

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon$-closure($\delta, E$)

If $e \notin R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r | \exists s \in r \text{ with } s \in F_n\}$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>{B,C}</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>{B,C}</td>
<td>{C}</td>
<td></td>
</tr>
<tr>
<td>{C}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$

$\quad$ //1

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon\text{-closure}(\delta, E)$

If $e \not\in R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>${A, B, C}$</td>
<td>${B, C}$</td>
<td>${A, B, C}$</td>
</tr>
<tr>
<td>${B, C}$</td>
<td>${C}$</td>
<td>?</td>
</tr>
<tr>
<td>${C}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let $r_0 = \varepsilon\text{-closure} (\delta, q_0)$, add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$  //1

Let $E = \text{move} (\delta, r, \sigma)$

Let $e = \varepsilon\text{-closure} (\delta, E)$

If $e \notin R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$
Let \( r_0 = \varepsilon\text{-closure}(\delta,q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)
- Mark \( r \)
- For each \( \sigma \in \Sigma \)
  - Let \( E = \text{move}(\delta,r,\sigma) \)
  - Let \( e = \varepsilon\text{-closure}(\delta,E) \)
  - If \( e \notin R \)
    - Let \( R = R \cup \{e\} \)
    - Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)
- Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \)
Let $r_0 = \varepsilon$-closure($\delta, q_0$), add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$ //1

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon$-closure($\delta, E$)

If $e \notin R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>{B,C}</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>{B,C}</td>
<td>{C}</td>
<td>{B,C}</td>
</tr>
<tr>
<td>{C}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \)

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \varepsilon\text{-closure}(\delta, E) \)

If \( e \notin R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r \mid \exists \ s \in r \text{ with } s \in F_n\} \)

---

<table>
<thead>
<tr>
<th>( {A, B, C} )</th>
<th>( {B, C} )</th>
<th>( {A, B, C} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B, C}</td>
<td>{C}</td>
<td>{B, C}</td>
</tr>
<tr>
<td>{C}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \)

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \varepsilon\text{-closure}(\delta, E) \)

If \( e \not\in R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r | \exists s \in r \text{ with } s \in F_n\} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B, C}</td>
<td>{B, C}</td>
<td>{A, B, C}</td>
</tr>
<tr>
<td>{B, C}</td>
<td>{C}</td>
<td>{B, C}</td>
</tr>
<tr>
<td>{C}</td>
<td>{C}</td>
<td></td>
</tr>
</tbody>
</table>
Let $r_0 = \varepsilon$-closure($\delta,q_0$), add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$   //0

Let $E = \text{move}(\delta,r,\sigma)$

Let $e = \varepsilon$-closure($\delta,E$)

If $e \notin R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$
Let $r_0 = \varepsilon$-closure$(\delta, q_0)$, add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$

//1

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon$-closure$(\delta, E)$

If $e \not\in R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in R \text{ with } s \in F_n\}$
Let \( r_0 = \varepsilon\text{-closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in \Sigma \)  

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \varepsilon\text{-closure}(\delta, E) \)

If \( e \not\in R \)

Let \( R = R \cup \{e\} \)

Let \( \delta' = \delta' \cup \{r, \sigma, e\} \)

Let \( F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\} \)

### Detailed NFA → DFA Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>{B,C}</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>{B,C}</td>
<td>{C}</td>
<td>{B,C}</td>
</tr>
<tr>
<td>{C}</td>
<td>{C}</td>
<td>{C}</td>
</tr>
</tbody>
</table>
Let $r_0 = \varepsilon$-closure($\delta, q_0$), add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$  //1

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon$-closure($\delta, E$)

If $e \notin R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$
Let $r_0 = \varepsilon$-closure($\delta, q_0$), add it to $R$.

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in \Sigma$

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon$-closure($\delta, E$)

If $e \not\in R$

Let $R = R \cup \{e\}$

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$
Detailed NFA → DFA Example: Completed

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,B,C}</td>
<td>{B,C}</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>{B,C}</td>
<td>{C}</td>
<td>{B,C}</td>
</tr>
<tr>
<td>{C}</td>
<td>{C}</td>
<td>{C}</td>
</tr>
</tbody>
</table>
NFA → DFA Example
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
Closing the Loop: Reducing DFA to RE

DFA \rightarrow \text{can reduce} \rightarrow \text{NFA} \rightarrow \text{can transform} \rightarrow \text{RE} \rightarrow \text{can transform} \rightarrow \text{DFA}

CMSC 330 Fall 2021
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

$$(0 + 1(0\ 1\ ^*\ 0)1)^*$$
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**
Splitting Partitions

- No need to split partition \{S, T, U, V\}
  - All transitions on \(a\) lead to identical partition \(P_2\)
  - 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}
Minimizing DFA: Example 1

- DFA

- Initial partitions

- Split partition
Minimizing DFA: Example 1

- **DFA**

- **Initial partitions**
  - Accept: \( \{ \text{R} \} = P_1 \)
  - Reject: \( \{ \text{S, T} \} = P_2 \)

- **Split partition?** → Not required, minimization done
  - \( \text{move}(S,a) = T \in P_2 \)  - \( \text{move}(S,b) = R \in P_1 \)
  - \( \text{move}(T,a) = T \in P_2 \)  - \( \text{move}(T,b) = R \in P_1 \)
Minimizing DFA: Example 2
Minimizing DFA: Example 2

DFA

- Initial partitions
  - Accept: \( \{ \text{R} \} = P_1 \)
  - Reject: \( \{ \text{S, T} \} = P_2 \)

- Split partition?
  - Yes, different partitions for B
    - move(S,a) = T \in P_2
    - move(S,b) = T \in P_2
    - move(T,a) = T \in P_2
    - move(T,b) = R \in P_1

DFA already minimal
Brzozowski’s Algorithm: DFA Minimization

1. Given a DFA, reverse all the edges, make the initial state an accept state, and the accept states initial, to get an NFA

2. NFA-> DFA

3. For the new DFA, reverse the edges (and initial-accept swap) get an NFA

4. NFA -> DFA
Brzozowski's algorithm

DFA

NFA

Minimum DFA

CMSC 330 Fall 2021
Complement of DFA

- 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
    - $\varepsilon$-closure & subset algorithm

- DFA
  - Minimization, complementation