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

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

A. 

B. 

C. 

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

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

---

It's easy to build a program which mimics a DFA.
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
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”
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: $\epsilon$

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

- Base case: $\emptyset$

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

- Induction: $AB$

\[
<\text{A}> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)
\]

\[
<\text{B}> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)
\]
Reduction: Concatenation

Induction: \( AB \)

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

Induction: $A \| B$

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

- $\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
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. 

CMSC 330 Fall 2020
Quiz 2: Which NFA matches $a^*$?
Quiz 3: Which NFA matches $a|b^*$?
Quiz 3: Which NFA matches $a|b^*$?
RE → NFA

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

```
NFA
S1 -> a -> S2

S2 -> a, ε -> S3

S1, S2, S3

DFA
S1 -> a -> S1, S2, S3
```
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
    - \( \{p,\varepsilon,p_1\} \in \delta \), \( \{p_1,\varepsilon,p_2\} \in \delta \), ..., \( \{p_n,\varepsilon,q\} \in \delta \)

- \( \varepsilon\)-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(\( \delta \), \( p \)) = \{ \( q | p \xrightarrow{\varepsilon} q \) 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 $p_2 \xrightarrow{\varepsilon} p_3$

- $\varepsilon$-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
  - p₁ → p₃
  - p₃ → p₂
  - p₁ → p₂
  - Since p₁ → p₃ and p₃ → p₂

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

Input: NFA (Σ, Q, q₀, Fₙ, δ), State Set R
Output: State Set R’

Algorithm

Let R’ = R
Repeat
   Let R = R’
   Let R’ = R ∪ {q | p ∈ R, (p, ε, q) ∈ δ}
Until R = R’

This algorithm computes a fixed point
ε-closure Algorithm Example

Calculate $\varepsilon$-closure($\delta$, {$p_1$})

$\begin{align*}
R & \quad R' \\
\{p_1\} & \quad \{p_1\} \\
\{p_1\} & \quad \{p_1, p_2\} \\
\{p_1, p_2\} & \quad \{p_1, p_2, p_3\} \\
\{p_1, p_2, p_3\} & \quad \{p_1, p_2, p_3\}
\end{align*}$

Let $R' = R$
Repeat
Let $R = R'$
Let $R' = R \cup \{q | p \in R, (p, \varepsilon, q) \in \delta\}$
Until $R = R'$

Let $R' = R$
Repeat
Let $R = R'$
Let $R' = R \cup \{q | p \in R, (p, \varepsilon, q) \in \delta\}$
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 ∅ 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
  - \( \Sigma = \{ a, b \} \)

- 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

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

Move

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

move({p1,p2}, a) = { p2, p3 }
NFA $\rightarrow$ 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 $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$  
      // 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

DFA
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.)

- $R = \{ \{p1,p3\}, \{p2\}, \{p3\} \}$
- $r \in R = \{p3\}$
- $\text{Move}(\{p3\},a) = \emptyset$
- $\text{Move}(\{p3\},b) = \emptyset$
- 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 equivalent to this NFA?

NFA:

- p0 -> p1 (a)
- p1 -> p2 (b)
- p0 -> p1 (ε)

DFA options:

A. p0 -> p1 (a) -> p1, p2 (b)
B. p0 -> p1 (b)
C. p0 -> p1 (a) -> p2, p0 (a)
D. None of the above

CMSC 330 Fall 2020
Quiz 4: Which DFA is equivalent to this NFA?

NFA:

A. 

B. 

C. 

D. None of the above
Actual Answer

NFA:

\[
\begin{array}{c}
p_0 \xrightarrow{a} p_1 \xrightarrow{b} p_2,
\end{array}
\]

\[
\begin{array}{c}
p_0, p_1, p_2,
\end{array}
\]
NFA $\rightarrow$ DFA Example 3

**NFA**

**DFA**

$$R = \{ \{A,E\} , \{B,D,E\} , \{C,D\} , \{E\} \}$$
NFA $\rightarrow$ DFA Example
NFA → DFA Practice
NFA → DFA Practice