CMSC 330: Organization of Programming Languages

DFAs, and NFAs, and Regexps (Oh my!)
Types of Finite Automata

- **Deterministic** Finite Automata (DFA)
  - Exactly one sequence of steps for each string
  - 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
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
Comparing DFAs and NFAs

- NFAs can have more than one transition leaving a state on the same symbol
  
  ![Diagram showing an NFA with multiple transitions on 'a'](image)

- 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 ε-transition
Comparing NFA and DFA for \((ab|aba)^*\)
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
    - $\varepsilon$-transitions
  - If any current state is final when done then accept $s$

- Example
  - After processing “a”
    - NFA may be in states
      - S1
      - S2
      - S3
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>S1</td>
<td>1</td>
<td>S1</td>
</tr>
</tbody>
</table>

or as $\{(S0,0,S0),(S0,1,S1),(S1,0,S0),(S1,1,S1)\}$
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
- \(\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q\) 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

- \(\Sigma = \{a\}\)
- \(Q = \{S1, S2, S3\}\)
- \(q_0 = S1\)
- \(F = \{S3\}\)
- \(\delta = \{(S1,a,S1), (S1,a,S2), (S2,\epsilon,S3)\}\)
Relating REs to DFAs and NFAs

- Regular expressions, NFAs, and DFAs accept the same languages!

![Diagram showing relationships between REs, DFAs, and NFAs]

- DFA can reduce NFA
- RE can transform DFA
- RE can reduce NFA
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: $AB$

\[ <A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A) \]
\[ <B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B) \]
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: $A | B$

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

Induction: $A|B$

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

Induction: $A^*$

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

Induction: $A^*$

- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<A^*> = (\Sigma_A, Q_A \cup \{S0, S1\}, S0, \{S1\}, \delta_A \cup \{(f_A, \varepsilon, S1), (S0, \varepsilon, q_A), (S0, \varepsilon, S1), (S1, \varepsilon, S0)\})$
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.

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!
Reducing NFA to DFA

DFA \quad \text{can reduce} \quad \text{NFA}

RE \quad \text{can reduce}
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, Fᵈ, δ)
  - Using two subroutines
    - ε-closure(δ, p) (and ε-closure(δ, S))
    - move(δ, p, a) (and move(δ, S, a))
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

- $S_1 \xrightarrow{\varepsilon} S_2$
- $S_2 \xrightarrow{\varepsilon} S_3$
- $S_1 \xrightarrow{\varepsilon} S_3$

Since $S_1 \xrightarrow{\varepsilon} S_2$ and $S_2 \xrightarrow{\varepsilon} S_3$

ε-closures

- $\varepsilon$-closure($S_1$) = \{ $S_1$, $S_2$, $S_3$ \}
- $\varepsilon$-closure($S_2$) = \{ $S_2$, $S_3$ \}
- $\varepsilon$-closure($S_3$) = \{ $S_3$ \}
- $\varepsilon$-closure( \{ $S_1$, $S_2$ \} ) = \{ $S_1$, $S_2$, $S_3$ \} \cup \{ $S_2$, $S_3$ \}
**ε-closure: Example 2**

- Following NFA contains:
  - $S_1 \xrightarrow{\varepsilon} S_3$
  - $S_3 \xrightarrow{\varepsilon} S_2$
  - $S_1 \xrightarrow{\varepsilon} S_2$
    - Since $S_1 \xrightarrow{\varepsilon} S_3$ and $S_3 \xrightarrow{\varepsilon} S_2$

- **ε-closures**
  - $\varepsilon$-closure($S_1$) = $\{ S_1, S_2, S_3 \}$
  - $\varepsilon$-closure($S_2$) = $\{ S_2 \}$
  - $\varepsilon$-closure($S_3$) = $\{ S_2, S_3 \}$
  - $\varepsilon$-closure($\{ S_2, S_3 \}$) = $\{ S_2 \} \cup \{ S_2, S_3 \}$
**ε-closure Algorithm: Approach**

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

**Algorithm**

Let \(R' = R\)  

Repeat

Let \(R = R'\)  

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

Until \(R = R'\)

This algorithm computes a **fixed point**  
- see note linked from project description
ε-closure Algorithm Example

Calculate $\epsilon$-closure($\delta$, {$S_1$})

<table>
<thead>
<tr>
<th>$R$</th>
<th>$R'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${S_1}$</td>
<td>${S_1}$</td>
</tr>
<tr>
<td>${S_1}$</td>
<td>${S_1, S_2}$</td>
</tr>
<tr>
<td>${S_1, S_2}$</td>
<td>${S_1, S_2, S_3}$</td>
</tr>
<tr>
<td>${S_1, S_2, S_3}$</td>
<td>${S_1, S_2, S_3}$</td>
</tr>
</tbody>
</table>

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,a)

move(δ,p,a)

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

• Notes:
  ➢ move(δ,p,a) is Ø if no transition (p,a,q) ∈ δ, for any q
  ➢ We write move(p,a) or move(R,a) when δ clear from context
move(a, p) : Example 1

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

Move
- $\text{move}(S1, a) = \{ S2, S3 \}$
- $\text{move}(S1, b) = \emptyset$
- $\text{move}(S2, a) = \emptyset$
- $\text{move}(S2, b) = \{ S3 \}$
- $\text{move}(S3, a) = \emptyset$
- $\text{move}(S3, b) = \emptyset$

$\text{move}(\{S1, S2\}, b) = \{ S3 \}$
move(a,p) : Example 2

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

- Move
  - move(S1, a) = \{ S2 \}
  - move(S1, b) = \{ S3 \}
  - move(S2, a) = \{ S3 \}
  - move(S2, b) = Ø
  - move(S3, a) = Ø
  - move(S3, b) = Ø

$Move(\{S1,S2\},a) = \{S2,S3\}$
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 $a \in \Sigma$  
      Let $E = \text{move}(\delta, r, a)$  
      // states reached via $a$
      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 $\delta' = \delta' \cup \{r, a, e\}$  
        // add transition $r \rightarrow e$
      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($\delta$, S1) = { {S1, S3} }
- $R = \{ \{S1, S3\}\}$
- $r \in R = \{S1, S3\}$
- move($\delta$, {S1, S3}, a) = {S2}
  - $e = \varepsilon$-closure($\delta$, {S2}) = {S2}
  - $R = R \cup \{\{S2\}\} = \{\{S1, S3\}, \{S2\}\}$
  - $\delta' = \delta' \cup \{\{S1, S3\}, a, \{S2\}\}$
- move($\delta$, {S1, S3}, b) = $\emptyset$

---

NFA

DFA
NFA → DFA Example 1 (cont.)

- \( R = \{ \{S1,S3\}, \{S2\} \} \)
- \( r \in R = \{S2\} \)
- \( \text{move}(\delta, \{S2\}, a) = \emptyset \)
- \( \text{move}(\delta, \{S2\}, b) = \{S3\} \)
  - \( e = \varepsilon\text{-closure}(\delta, \{S3\}) = \{S3\} \)
  - \( R = R \cup \{\{S3\}\} = \{ \{S1,S3\}, \{S2\}, \{S3\} \} \)
  - \( \delta' = \delta' \cup \{\{S2\}, b, \{S3\}\} \)
NFA → DFA Example 1 (cont.)

- \( R = \{ \{S1, S3\}, \{S2\}, \{S3\} \} \)
- \( r \in R = \{S3\} \)
- \( \text{Move}(\{S3\}, a) = \emptyset \)
- \( \text{Move}(\{S3\}, b) = \emptyset \)
- Mark \( \{S3\} \), exit loop
- \( F_d = \{\{S1, S3\}, \{S3\}\} \)
  - Since \( S3 \in F_n \)
- Done!
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
NFA:
Subset Algorithm as a Fixed Point

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

Output: DFA \(M'\)

Algorithm

Let \(q_0' = \varepsilon\)-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 \(q \in \text{states}(M), a \in \Sigma\) // for each DFA state \(q\) and letter \(a\)

Let \(s = \varepsilon\)-closure\((\delta, \text{move}(\delta, q, a))\) // new subset from \(q\)
Let \(F' = \{s\}\) if \(s \cap F \neq \emptyset\), or \(\emptyset\) otherwise, // subset contains final?
\(M' = M' \cup (\emptyset, \{s\}, \emptyset, F', \{(q, a, s)\})\) // update DFA

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

- $q_0' = \varepsilon$-closure($\delta, S_1) = \{S_1, S_3\}$
- $F' = \{\{S_1, S_3\}\}$ since $\{S_1, S_3\} \cap \{S_3\} \neq \emptyset$

- $M' = \{ \Sigma, \{\{S_1, S_3\}\}, \{S_1, S_3\}, \{\{S_1, S_3\}\}, \emptyset \}$
Redux: DFA to NFA Example 1 (cont)

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

- \( M' = M' \cup ( \emptyset, \{\{S2\}\}, \emptyset, \emptyset, \{\{\{S1,S3\},a,\{S2\}\}\} ) \)
- \( = \{ \Sigma, \{\{S1,S3\},\{S2\}\}, \{S1,S3\}, \{\{S1,S3\}\}, \{\{\{S1,S3\},a,\{S2\}\}\} \} \)
Redux: DFA to NFA Example 1 (cont)

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

- $M' = M' \cup$
  
  $\{ \emptyset, \{\{S3\}\}, \emptyset, \{\{S3\}\}, \{\{\{S2\},b,\{S3\}\}\}\}$

  $= \{ \Sigma, \{\{S1,S3\},\{S2\},\{S3\}\}, \{S1,S3\}, \{\{S1,S3\},\{S3\}\}, \{\{\{S1,S3\},a,\{S2\}\}, \{\{S2\},b,\{S3\}\}\}\}$

$Q'$ $q'_0$ $F'$ $\delta'$
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 $O(2^n)$
Reducing DFA to RE
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
Other Topics

- Minimizing DFA
  - Hopcroft reduction
- Complementing DFA
- Implementing 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 \textit{a} lead to identical partition \(P_2\)
  - Even though transitions on \textit{a} lead to different states

\begin{figure}[h]
\centering
\begin{tikzpicture}[node distance=1.5cm,auto,]
  \node (S) [state] {S};
  \node (T) [state, below of=S] {T};
  \node (U) [state, left of=T] {U};
  \node (V) [state, below of=U] {V};
  \node (X) [state, right of=T] {X};
  \node (Y) [state, below of=X] {Y};
  \node (Z) [state, below of=Y] {Z};

  \path[->, thick, draw=black]
    (S) edge node {a} (T)
    (T) edge node {a} (U)
    (U) edge node {a} (V)
    (X) edge node {a} (Y)
    (X) edge node {a} (Z);
\end{tikzpicture}
\caption{Diagram of state transitions}
\end{figure}
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 \(P_3\)

![Diagram showing transitions and partitions]

- From partition \(P_1\): \(S\) to \(T\) and \(U\) to \(S\), \(T\) are in the same group, \(U\) is in a different group.
- From partition \(P_2\): \(X\) to \(Y\) and \(Z\) are in the same group, \(X\) is in a different group.
- From partition \(P_4\): \(U\) to \(T\) and \(S\) to \(U\), \(T\) and \(S\) are in the same group, \(U\) is in a different group.

The transitions between the partitions indicate how the partitions are related and how the elements are moved between groups.
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 of partition resplitting](image)
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
  - move(S,b) = R ∈ P1
  - move(T,a) = T ∈ P2
  - move(T,b) = R ∈ P1
Minimizing DFA: Example 2
Minimizing DFA: Example 2

- **DFA**

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

- **Split partition? → Yes, different partitions for B**
  - \( \text{move}(S,a) = T \in P2 \) – \( \text{move}(S,b) = T \in P2 \)
  - \( \text{move}(T,a) = T \in P2 \) – \( \text{move}(T,b) = R \in P1 \)
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?
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;
    }
}

CMSC 330 Spring 2018
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 \(s\) of the input string)
    \(q := \delta(q, s)\);
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
Running Time of DFA

- How long for DFA to decide to accept/reject string $s$?
  - Assume we can compute $\delta(q, c)$ in constant time
  - Then time to process $s$ is $O(|s|)$
    - Can’t get much faster!

- Constructing DFA for RE $A$ may take $O(2^{|A|})$ time
  - But usually not the case in practice

- So there’s the initial overhead
  - But then processing strings is fast
Summary of Regular Expression Theory

- Finite automata
  - DFA, NFA

- Equivalence of RE, NFA, DFA
  - RE $\rightarrow$ NFA
    - Concatenation, union, closure
  - NFA $\rightarrow$ DFA
    - $\varepsilon$-closure & subset algorithm

- DFA
  - Minimization, complement
  - Implementation