CMSC 330: Organization of Programming Languages

Finite Automata 2
Recap

- **Languages**
  - Sets of strings
  - Operations on languages

- **Regular expressions**
  - Constants
  - Operators
  - Precedence

- **Finite automata**
  - States
  - Transitions
  - Accept strings
Examples

- Construct a DFA to accept all strings whose binary interpretation is divisible by 3
Examples

- Construct a DFA to accept a string containing a zero followed by a one
Examples

Construct a DFA to accept a string containing two consecutive zeroes followed by two consecutive ones.
The Questions

- Are FAs equivalent to regular expressions?
  - Every FA can be translated into an RE
  - Every RE can be translated into an FA
  - Yes!

- How can we generate an FA for a given RE?
  - If we can do this, we can implement RE matching

- How can we optimize an FA?
  - Many FAs can implement the same language
  - Some might be more efficient than others
Practice

Give the English descriptions and the DFA or regular expression of the following languages:

- \(((0|1)(0|1)(0|1)(0|1)(0|1))\)^*
Practice

Give the English descriptions and the DFA or regular expression of the following languages:

1. $((0|1)(0|1)(0|1)(0|1)(0|1))^*$
   - All strings with length a multiple of 5
Practice

Give the English descriptions and the DFA or regular expression of the following languages:

- \((01)^*|(10)^*|(01)^*0|(10)^*1\)
Practice

Give the English descriptions and the DFA or regular expression of the following languages:

- \((01)^*|(10)^*|(01)^*0|(10)^*1\)
  - All alternating binary strings
Practice

Give the English descriptions and the DFA or regular expression of the following languages:
Practice

Give the English descriptions and the DFA or regular expression of the following languages:

All binary strings containing the substring “11”
Give the English descriptions and the DFA or regular expression of the following languages:

All binary strings containing the substring “11”
Practice

Give the regular expressions and finite automata for the following languages

• You and your neighbors’ names
• All protein-coding DNA strings (including only ATCG and appearing in multiples of 3)
• All binary strings containing an even length substring of all 1’s
• All binary strings containing exactly two 1’s
• All binary strings that start and end with the same number
Types of Finite Automata

- Deterministic Finite Automata (DFA)
  - Exactly one sequence of steps for each string
  - All examples so far
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
Comparing DFAs and NFAs

- NFAs can have more than one transition leaving a state on the same symbol.
  
  ![Diagram of an NFA with multiple transitions 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

\[ \varepsilon \]-transition
NFA for \((a|b)^*abb\)

- **ba**
  - Has paths to either S0 or S1
  - Neither is final, so rejected
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 \((ab|aba)^*\)

- \text{aba}
  - Has paths to states \(S0, S1\)
NFA for \((ab|aba)^*\)

- \(aba\)
  - Has paths to states \(S0, S1\)
- \(ababa\)
  - Has paths to \(S0, S1\)
  - Need to use \(\varepsilon\)-transition
Another example DFA

Language?
Another example DFA

Language?
  • \((ab|aba)^*\)
Relating REs to DFAs and NFAs

- Regular expressions, NFAs, and DFAs accept the same languages!
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
- $\delta : Q \times \Sigma \to Q$ 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>$\delta$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>S1</td>
</tr>
</tbody>
</table>

or as $\{ (S0,0,S01),(S0,1,S1),(S1,0,S0),(S1,1,S1) \}$
Nondeterministic Finite Automata (NFA)

An NFA is a 5-tuple \((\Sigma, Q, q_0, F, \delta)\) where

- \(\Sigma\) 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
- \(\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q\) specifies the NFA's transitions
  - Transitions on \(\varepsilon\) are allowed – can optionally take these transitions without consuming any input
  - Can have more than one transition for a given state and symbol
    - \(\delta\) is a relation, not a function

An NFA accepts \(s\) if there is at least one path from its start to final state on \(s\)
Nondeterministic Finite Automata (NFA)

- An NFA 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
  - \(\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\} \times Q\) specifies the NFA's transitions
    - Transitions on \(\varepsilon\) are allowed – can optionally take these transitions without consuming any input
    - Can have more than one transition for a given state and symbol
      - \(\delta\) is a relation, not a function
- An NFA accepts \(s\) if there is at least one path from its start to final state on \(s\)
Reducing Regular Expressions to NFAs

- **Goal:** Given regular expression $e$, construct NFA: $<e> = (\Sigma, Q, q_0, F, \delta)$
  - Remember regular expressions are defined recursively from primitive RE languages
  - Invariant: $|F| = 1$ in our NFAs
    - Recall $F =$ set of final states

- **Base case:** $a$

  $<a> = (\{a\}, \{S0, S1\}, S0, \{S1\}, \{(S0, a, S1)\})$
Reduction (cont.)

- Base case: $\varepsilon$

$\langle \varepsilon \rangle = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset)$
Reduction (cont.)

- **Base case:** $\varepsilon$
  
  $\langle \varepsilon \rangle = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset)$

- **Base case:** $\emptyset$
  
  $\langle \emptyset \rangle = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)$
Reduction: Concatenation

Induction: $AB$

$q_A \xrightarrow{\cdot} f_A$

$q_B \xrightarrow{\cdot} f_B$

$\langle A \rangle$

$\langle B \rangle$
Reduction: Concatenation (cont.)

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: \((A \cup B)\)
Reduction: Union (cont.)

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: $A^*$
Reduction: Closure (cont.)

Induction: $A^*$

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

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 $n$...
  \[ \text{Size} = \# \text{ of symbols} + \# \text{ of operations} \]

- How many states does $<A>$ have?
Reduction Complexity

- Given a regular expression \( A \) of size \( n \)... 
  
  \[ \text{Size} = \# \text{ of symbols} + \# \text{ of operations} \]

- How many states does \( <A> \) have?
  - 2 added for each \(|\), 2 added for each \(*\)
  - \( O(n) \)
  - That’s pretty good!
Practice

- Draw NFAs for the following regular expressions and languages
  - $101^*|111$
  - all binary strings ending in 1 (odd numbers)
  - all alphabetic strings which come after “hello” in alphabetic order
Recap

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

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

Reducing RE to NFA

- Concatenation
- Union
- Closure
Next

- Reducing NFA to DFA
  - $\varepsilon$-closure
  - Subset algorithm
- Minimizing DFA
  - Hopcroft reduction
- Complementing DFA
- Implementing DFA
How NFA Works

- When NFA processes a string
  - NFA may be in several possible states
    - Multiple transitions with same label
    - $\varepsilon$-transitions

- Example
  - After processing “a”
    - NFA may be in states
      - S1
      - S2
      - S3
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 states

- Example

![Diagram showing NFA and DFA reduction](image)
Reducing NFA to DFA (cont.)

- Reduction applied using the **subset** algorithm
  - DFA state is a subset of set of all NFA states

Algorithm

- **Input**
  - NFA \((\Sigma, Q, q_0, F_n, \delta)\)

- **Output**
  - DFA \((\Sigma, R, r_0, F_d, \delta)\)

- Using two subroutines
  - \(\varepsilon\)-closure\((p)\)
  - move\((p, a)\)
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 ε-transitions
  - 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($p$)
  - Set of states reachable from $p$ using ε-transitions alone
  - Set of states $q$ such that $p \xrightarrow{\varepsilon} q$
  - $\varepsilon$-closure($p$) = $\{q \mid p \xrightarrow{\varepsilon} q \}$
  - Note
    - $\varepsilon$-closure($p$) always includes $p$
    - $\varepsilon$-closure( ) may be applied to set of states (take union)
\(\varepsilon\)-closure: Example 1

- Following NFA contains:
  - \(S_1 \xrightarrow{\varepsilon} S_2\)
  - \(S_2 \xrightarrow{\varepsilon} S_3\)
  - \(S_1 \xrightarrow{\varepsilon} S_3\)

  \(\varepsilon\)-closure for the transition \(S_1 \rightarrow S_2\) and \(S_2 \rightarrow S_3\)

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

- 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 \( S_3 \xrightarrow{\varepsilon} S_2 \)

- \( \varepsilon \text{-closures} \)
  - \( \varepsilon \text{-closure}(S_1) = \{ S_1, S_2, S_3 \} \)
  - \( \varepsilon \text{-closure}(S_2) = \{ S_2 \} \)
  - \( \varepsilon \text{-closure}(S_3) = \{ S_2, S_3 \} \)
  - \( \varepsilon \text{-closure}(\{ S_2, S_3 \}) = \{ S_2 \} \cup \{ S_2, S_3 \} \)
Calculating \text{move}(p,a)

- \text{move}(p,a)
  - Set of states reachable from $p$ using exactly one transition on $a$
    - Set of states $q$ such that $\{p, a, q\} \in \delta$
    - $\text{move}(p,a) = \{q \mid \{p, a, q\} \in \delta\}$
  - Note: $\text{move}(p,a)$ may be empty $\emptyset$
    - If no transition from $p$ with label $a$
move(a,p) : Example 1

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

- Move
  - $\text{move}(S1, a) = \{ S2, S3 \}$
  - $\text{move}(S1, b) = \emptyset$
  - $\text{move}(S2, a) = \emptyset$
  - $\text{move}(S2, b) = \{ S3 \}$
  - $\text{move}(S3, a) = \emptyset$
  - $\text{move}(S3, b) = \emptyset$
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) = $\emptyset$
  - move(S3, a) = $\emptyset$
  - move(S3, b) = $\emptyset$
NFA $\rightarrow$ DFA Reduction Algorithm

- Input NFA $(\Sigma, Q, q_0, F_n, \delta)$, Output DFA $(\Sigma, R, r_0, F_d, \delta)$
- Algorithm

  Let $r_0 = \varepsilon$-closure($q_0$), 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 $a \in \Sigma$  // for each letter $a$
      Let $S = \{s \mid q \in r \& \text{move}(q,a) = s\}$  // states reached via $a$
      Let $e = \varepsilon$-closure($S$)  // 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 $\delta = \delta \cup \{r, a, e\}$  // 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 $\rightarrow$ DFA Example 1
NFA → DFA Example 1
NFA → DFA Example 1

- Start = $\varepsilon$-closure($S_1$) = \{ $\{S_1, S_3\}$ \}
- $R = \{ \{S_1, S_3\} \}$
- $r \in R = \{S_1, S_3\}$
- Move($\{S_1, S_3\}, a$) = $\{S_2\}$
  - $e = \varepsilon$-closure($\{S_2\}$) = $\{S_2\}$
  - $R = R \cup \{\{S_2\}\} = \{ \{S_1, S_3\}, \{S_2\} \}$
  - $\delta = \delta \cup \{\{S_1, S_3\}, a, \{S_2\}\}$
- Move($\{S_1, S_3\}, b$) = $\emptyset$
NFA → DFA Example 1 (cont.)

• $R = \{ \{S1, S3\}, \{S2\} \}$
• $r \in R = \{S2\}$
• $\text{Move}(\{S2\}, a) = \emptyset$
• $\text{Move}(\{S2\}, b) = \{S3\}$
  - $\varepsilon = \varepsilon\text{-closure}(\{S3\}) = \{S3\}$
  - $R = R \cup \{\{S3\}\} = \{\{S1, S3\}, \{S2\}, \{S3\}\}$
  - $\delta = \delta \cup \{\{S2\}, b, \{S3\}\}$
NFA $\rightarrow$ DFA Example 1 (cont.)

- $R = \{ \{S1, S3\}, \{S2\}, \{S3\} \}$
- $r \in R = \{S3\}$
- $\text{Move}(\{S3\}, a) = \emptyset$
- $\text{Move}(\{S3\}, b) = \emptyset$
- $\text{Mark } \{S3\}, \text{ exit loop}$
- $F_d = \{\{S1, S3\}, \{S3\}\}$
  - Since $S3 \in F_n$
- Done!
NFA → DFA Example 2

- NFA

\[ R = \{ \{A\}, \{B,D\}, \{C,D\} \} \]
NFA → DFA Example 3

R = \{ \{A, E\}, \{B, D, E\}, \{C, D\}, \{E\} \}
Analyzing the reduction

- Any string from \{A\} to either \{D\} or \{CD\}
  - Represents a path from A to D in the original NFA

NFA

DFA
Analyzing the reduction (cont’d)

- 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 $O(2^n)$
Minimizing DFA

Result from CS theory
- Every regular language is recognizable by a minimum-state DFA that is unique up to state names.

In other words
- For every DFA, there is a unique DFA with minimum number of states that accepts the same language.
- Two minimum-state DFAs have same underlying shape.
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 refine 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
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 P2
  - Transition on \(a\) from U lead to partition P3
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\}
  - Need to split partition \{S,T,U\} into \{S,T\}, \{U\}
DFA Minimization Algorithm (1)

- **Input** DFA (Σ, Q, q₀, Fₙ, δ), **Output** DFA (Σ, R, r₀, Fₜ, δ)

- **Algorithm**
  
  Let p₀ = Fₙ, p₁ = Q – F  \hspace{1cm} // initial partitions = final, nonfinal states

  Let R = \{ p | p ∈ \{p₀,p₁\} and p ≠ Ø \}, P = Ø  \hspace{1cm} // add p to R if nonempty

  While P ≠ R do
    \hspace{1cm} // while partitions changed on prev iteration
    Let P = R, R = Ø  \hspace{1cm} // P = prev partitions, R = current partitions
    For each p ∈ P  \hspace{1cm} // for each partition from previous iteration
      \{p₀,p₁\} = split(p,P) \hspace{1cm} // split partition, if necessary
      R = R ∪ \{ p | p ∈ \{p₀,p₁\} and p ≠ Ø \} \hspace{1cm} // add p to R if nonempty
    r₀ = p ∈ R where q₀ ∈ p  \hspace{1cm} // partition w/ starting state
    Fₜ = \{ p | p ∈ R and exists s ∈ p such that s ∈ Fₙ \} \hspace{1cm} // partitions w/ final states
    δ(p,c) = q when δ(s,c) = r where s ∈ p and r ∈ q  \hspace{1cm} // add transitions
DFA Minimization Algorithm (2)

Algorithm for \texttt{split}(p, P)

Choose some \( r \in p \), let \( q = p - \{r\} \), \( m = \{\} \) // pick some state \( r \) in \( p \)

For each \( s \in q \) // for each state in \( p \) except for \( r \)

For each \( c \in \Sigma \) // for each symbol in alphabet

If \( \delta(r, c) = q_0 \) and \( \delta(s, c) = q_1 \) and // \( q \)'s = states reached for \( c \)

there is no \( p_1 \in P \) such that \( q_0 \in p_1 \) and \( q_1 \in p_1 \) then

\( m = m \cup \{s\} \) // add \( s \) to \( m \) if \( q \)’s not in same partition

Return \( p - m, m \) // \( m \) = states that behave differently than \( r \)

// \( m \) may be \( \emptyset \) if all states behave the same
// \( p - m \) = states that behave the same as \( r \)
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

- **DFA**

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

- **Split partition? → Not required, minimization done**
  - \( \text{move}(S,a) = T \in P_2 \)  
    - \( \text{move}(S,b) = R \in P_1 \)
  - \( \text{move}(T,a) = S \in P_2 \)  
    - \( \text{move}(T,b) = R \in P_1 \)
Minimizing DFA: Example 3

- **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
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 (cont.)

- 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?
Practice

Make the DFA which accepts the complement of the language accepted by the DFA below.
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
Relating REs to DFAs and NFAs

Why do we want to convert between these?
- Can make it easier to express ideas
- Can be easier to implement
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;
    }
}
```
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
Run 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
Regular Expressions in Practice

- Regular expressions are typically “compiled” into tables for the generic algorithm
  - Can think of this as a simple byte code 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
Practice

- Convert to a DFA

- Convert to an NFA and then to a DFA
  - \((0|1)^*11|0^*\)
  - Strings of alternating 0 and 1
  - \(aba^*|(ba|b)\)
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