CMSC 330: Organization of Programming Languages

This Lecture
- Reducing NFA to DFA
  - $\epsilon$-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
    - $\epsilon$-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

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 ($\Sigma, Q, q_0, \delta$)
  - Output
    - DFA ($\Sigma, R, r_0, F, \delta$)
  - Using
    - $\epsilon$-closure($p$)
    - move($p, a$)

Last Lecture
- Finite automata
  - Alphabet, states...
  - ($\Sigma, Q, q_0, F, \delta$)
- Types
  - Deterministic (DFA)
  - Non-deterministic (NFA)
ε-transitions and ε-closure

- We say \( p \xrightarrow{\epsilon} q \)
  - 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
    \( (p, p_1) \in \delta, (p_1, p_2) \in \delta, \ldots, (p_n, q) \in \delta \)

ε-closure(\( p \))
- Set of states reachable from \( p \) using ε-transitions alone
  - Set of states \( q \) such that \( p \xrightarrow{\epsilon} q \)
  - ε-closure(\( p \)) = \{ \( q \mid p \xrightarrow{\epsilon} q \) \}
- Note ε-closure(\( p \)) always includes \( p \)
  - ε-closure(\( p \)) may be applied to set of states (take union)

ε-closure: Example 1

- Following NFA contains
  - \( S_1 \xrightarrow{a} S_2 \)
  - \( S_2 \xrightarrow{b} S_3 \)
  - \( S_1 \xrightarrow{\epsilon} S_3 \)

ε-closures
- ε-closure(\( S_1 \)) = \{ \( S_1, S_2, S_3 \) \}
- ε-closure(\( S_2 \)) = \{ \( S_2, S_3 \) \}
- ε-closure(\( S_3 \)) = \{ \( S_3 \) \}
- ε-closure(\{ \( S_1, S_2 \) \}) = \{ \( S_1, S_2, S_3 \) \cup \{ S_2, S_3 \) \}

ε-closure: Example 2

- Following NFA contains
  - \( S_1 \xrightarrow{a} S_2 \)
  - \( S_3 \xrightarrow{b} S_2 \)
  - \( S_1 \xrightarrow{\epsilon} S_2 \)

ε-closures
- ε-closure(\( S_1 \)) = \{ \( S_1, S_2, S_3 \) \}
- ε-closure(\( S_2 \)) = \{ \( S_2 \) \}
- ε-closure(\( S_3 \)) = \{ \( S_2, S_3 \) \}
- ε-closure(\{ \( S_2, S_3 \) \}) = \{ \( S_2 \) \cup \{ S_2, S_3 \) \}

ε-closure: Practice

- Find ε-closures for following NFA

Find ε-closures for the NFA you construct for
- The regular expression \( (0|1)^*111(0^*1) \)

Calculating move(\( p, a \))

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

move(\( a, p \)) : Example 1

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

Move
- move(\( S_1, a \)) = \{ \( S_2, S_3 \) \}
- move(\( S_1, b \)) = \emptyset
- move(\( S_2, a \)) = \emptyset
- move(\( S_2, b \)) = \{ \( S_3 \) \}
- move(\( S_3, a \)) = \emptyset
- move(\( S_3, b \)) = \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 → DFA Reduction Algorithm**

- Input NFA ($\Sigma$, Q, $q_0$, F$_{\text{in}}$, $\delta$), Output DFA ($\Sigma$, R, $r_0$, F$_{\text{fin}}$, $\delta$)

- Algorithm
  - Let $r_0 = \varepsilon$-closure($q_0$), add it to R  
    - DFA start state
  - While 3 unmarked state $r \in R$
    - Mark r  
      - process DFA state $r$
    - For each $a \in \Sigma$
      - Let S = { s | q \in R & move(q,a) = s }
        - states reached via a
      - Let $e = \varepsilon$-closure(S)
        - states reached via $\varepsilon$
      - If $e \in R$
        - If state e is new
          - add e to R (unmarked)
          - add transition $r \rightarrow e$
      - Let $F_d = \{ r | \exists s \in r \text{ with } s \in F_{\text{in}} \}$  
        - final if include state in $F_{\text{fin}}$
  - Add i
  - Add transition $r \rightarrow \sigma$

**NFA → DFA Example 1**

- Start = $\varepsilon$-closure(S1) = { S1, S3 }  
  - DFA start state
- $R = \{ (S1, S3) \}$  
  - DFA start state
- $r \in R = (S1, S3)$  
  - DFA start state
- $\delta = \varepsilon$-closure((S1, S3)) = { S2 }  
  - DFA start state
- Move((S1, S3), a) = (S2)  
  - DFA transition
- Move((S1, S3), b) = $\emptyset$  
  - DFA transition

**NFA → DFA Example 1 (cont.)**

- $R = \{ (S1, S3), (S2) \}$  
  - DFA start state
- $r \in R = (S2)$  
  - DFA start state
- Move((S2), a) = $\emptyset$  
  - DFA transition
- Move((S2), b) = { S3 }  
  - DFA transition
- $\delta = \varepsilon$-closure((S3)) = { S3 }  
  - DFA transition
- $\delta = \delta \cup \{(S1, S3), a, (S2)\}$  
  - DFA transition
- Move((S1, S3), b) = $\emptyset$  
  - DFA transition

**NFA → DFA Example 1 (cont.)**

- $R = \{ (S1, S3), (S2), (S3) \}$  
  - DFA start state
- $r \in R = (S3)$  
  - DFA start state
- Move((S3), a) = $\emptyset$  
  - DFA transition
- Move((S3), b) = $\emptyset$  
  - DFA transition
- $F_d = \{ (S1, S3), (S3) \}$  
  - DFA transition
  - Since $S3 \in F_{\text{fin}}$
- Done!
NFA → DFA Example 3

- **NFA**
  - Graph with states A, B, C, D, and E.
  - Transitions labeled with a, b, c, and ε.

- **DFA**
  - Graph with states A, B, C, D, and E.
  - Transitions labeled with a, b, c, and ε.

\[ R = \{ (A,E), (B,D,E), (C,D), (E) \} \]

---

Equivalence of DFAs and NFAs

- Any string from \( S \) to either \( D \) or \( CD \)
  - Represents a path from A to D in the original NFA

---

Equivalence of DFAs and NFAs (cont.)

- 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 for states that can be distinguish 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 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 P2
  - Even though transitions on \( a \) lead to different states
Splitting Partitions (cont.)

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

<table>
<thead>
<tr>
<th>P1</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P3</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

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 \(\langle \Sigma, Q, q_0, F, \delta \rangle\). Output DFA \(\langle \Sigma, R, r_0, F, \delta \rangle\)
- Algorithm
  - Let \(p_0 = F_0, p = Q - F\) // initial partitions = final, nonfinal states
  - Let \(R = \{ p | p \in \{ p_0, p \} \text{ and } p \neq \emptyset \}\), \(P = \emptyset\) // add to \(R\) if nonempty
  - While \(P \neq R\) do
    - Let \(P = R, R = \emptyset\) // while partitions changed on prev iteration
    - For each \(p \in P\) // for each partition from previous iteration
      - \(p_0 = p, p \in R\) \(\Rightarrow\) split partition, if necessary
      - \(R = R \cup \{ p | p \in \{ p_0, p \} \text{ and } p \neq \emptyset \}\) // add to \(R\) if nonempty
      - \(F_0 = \{ p | p \in R\} \text{ and exists } s \in p \text{ such that } s \in F_0\} // partition w/ starting state
      - \(F_0 = \{ p | p \in R\} \text{ and exists } s \in p \text{ such that } s \in F_0\} // partition w/ final states
      - \(\delta(p,c) = q \text{ when } \delta(s,c) = r \text{ where } s \in p \text{ and } r \in q\) // add transitions

DFA Minimization Algorithm (2)

- Algorithm for \(\text{split}(p,P)\)
  - Choose some \(r \subseteq p\), let \(q = p - \{r\}, m = \{\}\) // pick some state \(r\) in \(p\)
  - For each \(s \subseteq q\) // for each state in \(p\) except for \(r\)
    - For each \(c \in \Sigma\) // for each symbol in alphabet
      - If \(\delta(r,c) = q_1\) and \(\delta(s,c) = q_2\) and \(q_1 \neq q_2\)
        - \(m = m \cup \{s\}\) // add \(s\) to \(m\) if \(q_1\)'s not in same partition
      - \(m = m \cup \{r\}\) // \(m\) is states that behave differently than \(r\)
      - \(m\) may be \(\emptyset\) if all states behave the same
      - \(p - m = \text{states that behave the same as } r\)

Minimizing DFA: Example 1

- DFA
  
<table>
<thead>
<tr>
<th>P2</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
</tr>
</tbody>
</table>

  - Initial partitions
    - Accept \(\{R\}\) \(\rightarrow\) P1
    - Reject \(\{S, T\}\) \(\rightarrow\) P2
  - Split partition? \(\rightarrow\) Not required, minimization done
    - \(\text{move}(S,a) = T \rightarrow P_2\)
    - \(\text{move}(T,a) = T \rightarrow P_2\)

Minimizing DFA: Example 2

- DFA
  
<table>
<thead>
<tr>
<th>P2</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
</tr>
</tbody>
</table>

  - Initial partitions
    - Accept \(\{R\}\) \(\rightarrow\) P1
    - Reject \(\{S, T\}\) \(\rightarrow\) P2
  - Split partition? \(\rightarrow\) Not required, minimization done
    - \(\text{move}(S,a) = T \rightarrow P_2\)
    - \(\text{move}(T,a) = T \rightarrow P_2\)
Minimizing DFA: Example 3

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

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

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

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

```
while (1) {
    symbol = getchar();
    switch (cur_state) {
        case 0: switch (symbol) {
            case '0':  cur_state = 0; break;
            case '1':  cur_state = 1; break;
            default:   printf("unknown state; I'm confused\n"); break;
        }
        case '1':  cur_state = 1; break;
        case '
': printf("rejected\n"); return 0;
        default:   printf("rejected\n"); return 0;
    }
}
```

Implementing DFAs (Alternative)

Alternatively, use generic table-driven DFA

```
given components (Σ, Q, q₀, F, δ) of a DFA:
let q = q₀
while (there exists another symbol s of the input string)
    q = δ(q, s);
if q ∈ F then accept
else reject
```

Run Time of DFA

- How long for DFA to decide to accept/reject string s?
  - Assume we can compute δ(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 (Σ, Q, q₀, F, δ₀, δ₁), 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)*110*
  - Strings of alternating 0 and 1
  - aba"[(ba]b"

Summary of Regular Expression Theory

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