Finite Automata 2

This Lecture
- Reducing NFA to DFA
  - \( \varepsilon \)-closure
  - Subset algorithm
- Minimizing DFA
  - Moore reduction
- Complementing DFA
- Implementing 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 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 \( (\Sigma, Q, q_0, F, \delta) \)
  - Output
    - DFA \( (\Sigma, R, r_0, F_d, \delta) \)
  - Using
    - \( \varepsilon \)-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 \( \varepsilon \)-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 \( \exists \ p \xrightarrow{\varepsilon} q \)
    - ε-closure(\( p \)) = \( \{ q \mid p \xrightarrow{\varepsilon} q \} \)
  - Note
    - ε-closure(\( p \)) always includes \( p \)
    - ε-closure(\( ) \) may be applied to set of states (take union)

ε-closure: Example 1

- Following NFA contains
  - \( S_1 \xrightarrow{\varepsilon} S_2 \)
  - \( S_2 \xrightarrow{\varepsilon} S_3 \)
  - \( S_1 \xrightarrow{\varepsilon} S_3 \)
- 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(\( \{ S_1, S_2 \} \)) = \{ \( S_1, S_2, S_3 \) \} ∪ \{ \( S_2, S_3 \) \}

ε-closure: Example 2

- Following NFA contains
  - \( S_1 \xrightarrow{\varepsilon} S_3 \)
  - \( S_3 \xrightarrow{\varepsilon} S_2 \)
  - \( S_1 \xrightarrow{\varepsilon} S_2 \)
- 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(\( \{ S_2, S_3 \} \)) = \{ \( S_2 \) \} ∪ \{ \( S_2, S_3 \) \}

ε-closure: Practice

- Find ε-closures for following NFA
  - 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 \( a \)
    - Set of states \( q \) such that \( \{ p, a, q \} \in \delta \)
    - move(\( p, a \)) = \{ \( q \mid \{ p, a, q \} \in \delta \} \)
  - Note 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
  - 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

• Σ = { 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

NFA → DFA Reduction Algorithm

• Input NFA (Σ, Q, q0, F, δ)
• Output DFA (Σ, R, r0, Fd, δ)

Algorithm

• Let r0 = ε-closure(q0), add it to R // DFA start state
• While ∃ an unmarked state r ∈ R // process DFA state r
  • Mark r // each state visited once
  • For each a ∈ Σ // for each letter a
    • Let S = { s | q ∈ r & move(q,a) = s } // states reached via a
    • Let e = ε-closure(S) // states reached via ε
    • If e ∈ R // if state e is new
      • Let R = e ∪ R // add e to R (unmarked)
    • Let δ = δ ∪ { r, a, e } // add transition r-e
  • Let Fd = { r | ∃ s ∈ r with s ∈ F } // final if include state in F

NFA → DFA Example 1

• Start = ε-closure(S1) = { S1,S3 }
• R = { S1,S3 }
• r ∈ R = { S1,S3 }
• Move(S1,S3,a) = S2
  • e = ε-closure(S2) = { S2 }
  • R = R ∪ { S2 } = { S1,S3 }, ( S2 )
  • δ = δ ∪ ( { S1,S3 }, a, { S2 } )
• Move(S1,S3,b) = Ø

NFA → DFA Example 1 (cont.)

• R = { S1,S3 }, ( S2 )
• r ∈ R = { S2 }
• Move(S2),a) = Ø
• Move(S2,b) = { S3 }
  • e = ε-closure(S3) = { S3 }
  • R = R ∪ { S3 } = { S1,S3 }, ( S2 ), ( S3 )
  • δ = δ ∪ { S2 }, b, ( S3 )

NFA → DFA Example 2
NFA → DFA Example 3

- NFA
- DFA

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

Equivalence of DFAs and NFAs

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

Equivalent 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: Moore 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
    - 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 \( R \) 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 \( \{X\} \), \( \{Y\} \)
  - Need to split partition \( \{S,T,U\} \) into \( \{S,T\} \), \( \{U\} \)

Minimizing DFA: Example 1

- DFA

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

- Split partition? → Not required, minimization done
  - move(\(S,a\)) = \(T \rightarrow P_2\)
  - move(\(S,b\)) = \(R \rightarrow P_1\)
  - move(\(T,a\)) = \(T \rightarrow P_2\)
  - move(\(T,b\)) = \(R \rightarrow P_1\)

Minimizing DFA: Example 2

- DFA

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

- Split partition? → Not required, minimization done
  - move(\(S,a\)) = \(T \rightarrow P_2\)
  - move(\(S,b\)) = \(R \rightarrow P_1\)
  - move(\(T,a\)) = \(S \rightarrow P_2\)
  - move(\(T,b\)) = \(R \rightarrow P_1\)

Minimizing DFA: Example 3

- DFA

- Initial partitions
  - Accept \( \{R\} \) → \( P_1 \) minimal
  - Reject \( \{S,T\} \) → \( P_2 \)

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

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

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

- Alternatively, use generic table-driven DFA
  - q is just an integer
  - Represent δ 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 $(\Sigma, Q_A, q_0, f_A, \delta_A)$, 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
    > $\epsilon$-closure & subset algorithm
- DFA
  - Minimization, complement
  - Implementation