CMSC 330: Organization of Programming Languages

Finite Automata 2

This Lecture

- Reducing NFA to DFA
  - ε-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
  - ε-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 (Σ, Q, q0, F, δ)
  - Output
    - DFA (Σ, R, r0, F, δ)
  - Using
    - ε-closure(p)
    - move(p, a)
ε-transitions and ε-closure

We say p ε q
- If it is possible to go from state p to state q by taking only ε-transitions
- If ∃ p, p₁, p₂, ... p_n, q ∈ Q such that
  (p, ε, p₁) = δ, (p₁, ε, p₂) = δ, ... , (p_n, ε, q) = δ

ε-closure(p)
- Set of states reachable from p using ε-transitions alone ε
- ε-closure(p) = {q | p ε 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
- S₁, S₂
- S₂, S₃
- S₁, S₃

ε-closures
- ε-closure(S₁) = { S₁, S₂, S₃ }
- ε-closure(S₂) = { S₂, S₃ }
- ε-closure(S₃) = { S₃ }
- ε-closure( { S₁, S₂ } ) = { S₁, S₂, S₃ } ∪ { S₂, S₃ }

ε-closure: Example 2

Following NFA contains
- S₁, S₂
- S₃, S₂
- S₁, S₂

ε-closures
- ε-closure(S₁) = { S₁, S₂, S₃ }
- ε-closure(S₂) = { S₂ }
- ε-closure(S₃) = { S₂, S₃ }
- ε-closure( { S₂, S₃ } ) = { S₂ } ∪ { S₂, S₃ }

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
  Set of states q such that [p, a, q] ∈ δ
  move(p,a) = {q | [p, a, q] ∈ δ}
- Note move(p,a) may be empty Ø
  If no transition from p with label a

move(a,p) : Example 1

Following NFA
- Σ = { a, b }

Move
- move(S₁, a) = { S₂, S₃ }
- move(S₁, b) = Ø
- move(S₂, a) = Ø
- move(S₂, b) = { S₃ }
- move(S₃, a) = Ø
- move(S₃, b) = Ø
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, F0, δ).
- Output DFA (Σ, R, r0, F, δ).

Algorithm

- Shift r0 = ε-closure(q0), add it to R.
- While ∃ an unmarked state r ∈ R:
  - Mark r.
  - For each a ∈ Σ:
    - Let S = {s | q ∈ r & move(q, a) = s}.
    - Let e = ε-closure(S).
    - If e ∈ R:
      - Let R = e ∪ R.
      - Let δ = δ ∪ {r, a, e}.
      - If e is new, add to R (unmarked).
    - If r is marked, add transition r → e.
  - If final state in R, add to 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, 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 1 (cont.)

- R = {{S1, S3}, {S2}, {S3}}.
- r ∈ R = {S3}.
- Move((S3), a) = Ø.
- Move((S3), b) = Ø.
- F = {{S1, S3}, {S3}}.
  - Since S3 ∈ F,
  - Done!

NFA → DFA Example 2

- NFA
- DFA

R = {{A}, {B, D}, {C, D}}.
NFA → DFA Example 3

- **NFA**
  
  ![NFA Diagram]

- **DFA**
  
  ![DFA Diagram]

Equivalent states: $R = \{ \{A, E\}, \{B, D, E\}, \{C, D\}, \{E\} \}$

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 distinguished 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

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 \( R \) lead to partition \( P_3 \)

![Diagram of splitting partitions]

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 \( \{S,T\}, \{U\} \)

![Diagram of resplitting partitions]

Minimizing DFA: Example 1

- DFA
  - Initial partitions
    - Accept \( \{R\} \) → \( P_1 \)
    - Reject \( \{S,T\} \) → \( P_2 \)
  - Split partition? → Not required, minimization done
    - \( \text{move}(S,a) = T \rightarrow P_2 \) → \( \text{move}(S,b) = R \rightarrow P_1 \)
    - \( \text{move}(T,a) = T \rightarrow P_2 \) → \( \text{move}(T,b) = R \rightarrow P_1 \)
  - After cleanup

![Diagram of minimization example 1]

Minimizing DFA: Example 2

- DFA
  - Initial partitions
    - Accept \( \{R\} \) → \( P_1 \)
    - Reject \( \{S,T\} \) → \( P_2 \)
  - Split partition? → Not required, minimization done
    - \( \text{move}(S,a) = T \rightarrow P_2 \) → \( \text{move}(S,b) = R \rightarrow P_1 \)
    - \( \text{move}(T,a) = S \rightarrow P_2 \) → \( \text{move}(T,b) = R \rightarrow P_1 \)
  - After cleanup

![Diagram of minimization example 2]

Minimizing DFA: Example 3

- DFA
  - Initial partitions
    - Accept \( \{R\} \) → \( P_1 \)
    - Reject \( \{S,T\} \) → \( P_2 \)
  - Split partition? → Yes, different partitions for \( B \)
    - \( \text{move}(S,a) = T \rightarrow P_2 \) → \( \text{move}(S,b) = T \rightarrow P_2 \)
    - \( \text{move}(T,a) = T \rightarrow P_2 \) → \( \text{move}(T,b) = R \rightarrow P_1 \)

![Diagram of minimization example 3]

Complement of DFA

- Given a DFA accepting language \( L \)
  - How can we create a DFA accepting its complement?
  - Example DFA
    - \( \Sigma = \{a,b\} \)

![Diagram of complement DFA]
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

Implementing DFAs

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

Implementing DFAs (Alternative)

Alternatively, use generic table-driven DFA

Given components \( I, Q, \delta, F \) of a DFA:

\[
\text{let } q = q_0 \\
\text{while (there exists another symbol } s \text{ of the input string)} \\
\text{q := } \delta(q, s) \\
\text{if } q \in F \text{ then accept} \\
\text{else reject}
\]

- \( q \) 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 $(\Sigma, Q, q_0, f_A, \delta_A)$, 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
  - $((0|1)^*11|0)^*$
- Convert to an NFA and then to a DFA
  - 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