CMSC 330: Organization of Programming Languages

Finite Automata 2

Dr. Anwar Mamat
Examples

- Construct a DFA to accept all strings DO NOT contain odd number of zeroes and odd number of ones Alphabet = {0, 1}
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

![DFA Diagram]

0 |

0, 1
Examples

- Construct a DFA to accept a string containing two consecutive zeroes followed by two consecutive ones
Clicker Quiz now!
Types of Finite Automata

- Deterministic Finite Automata (DFA)

\[ \Sigma = \{a, b\} \]

- Nondeterministic Finite Automata (NFA)

\[ \Sigma = \{a, b\} \]
NFA for \((a|b)^*abb\)

- **ba** \textit{reject}

- **babaabb** \textit{accept}

\[
\begin{array}{c}
\text{S0} \xrightarrow{a} \text{S1} \xrightarrow{b} \text{S2} \xrightarrow{b} \text{S3}
\end{array}
\]
NFA for (ab|aba)*
NFA for (ab|aba)*

ababa

accept
DFA vs. NFA

ababa

CMSC 330 Fall 16
Relating REs to DFAs and NFAs

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

- $\Sigma = \{0, 1\}$
- $Q = \{S_0, S_1\}$
- $q_0 = S_0$
- $F = \{S_1\}$

or as

\[
\begin{array}{c|cc}
\delta & 0 & 1 \\
\hline
S_0 & S_0 & S_1 \\
S_1 & S_0 & S_1 \\
\end{array}
\]

or as \{ (S_0, 0, S_01), (S_0, 1, S_1), (S_1, 0, S_0), (S_1, 1, S_1) \}
Nondeterministic Finite Automata (NFA)

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

- \(\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q\) specifies the NFA's transitions.

An NFA accepts \(s\) if there is at least one path from its start to final state on \(s\).
Reducing Regular Expressions to NFAs

- Base case: $a$

- Base case: $\varepsilon$

- Base case: $\emptyset$
Reducing Regular Expressions to NFAs

- Base case: $a$

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

- **Base case**: $\varepsilon$

  $$<\varepsilon> = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset)$$

- **Base case**: $\emptyset$

  $$<\emptyset> = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)$$
Reduction: Concatenation

- Induction: $AB$
Reduction: Concatenation (cont.)

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

- **Induction: \((A|B)\)**

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

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

- **Induction: A**

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

![Diagram](image-url)
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 $n$...
  
  Size = # of symbols + # of operations

- How many states does $<A>$ have?
  
  - 2 added for each |, 2 added for each *
  - $O(n)$
  - That’s pretty good!
Reducing NFA to DFA
Reducing NFA to DFA (cont.)

- two subroutines
  - $\epsilon$-closure(p)
  - move(p, a)
\( \varepsilon \)-closures

- \( \varepsilon \)-closure(S1) = \{ S1, S2, S3 \}
- \( \varepsilon \)-closure(S2) = \{ S2, S3 \}
- \( \varepsilon \)-closure(S3) = \{ S3 \}
- \( \varepsilon \)-closure( \{ S1, S2 \} ) = \{ S1, S2, S3 \} \cup \{ S2, S3 \} \)
move(a, p) : Example 1

Move

- move(S1, a) = \{S2, S3\}
- move(S1, b) = \emptyset
- move(S2, a) = \emptyset
- move(S2, b) = \{S3\}
- move(S3, a) = \emptyset
- move(S3, b) = \emptyset
NFA → DFA Reduction Algorithm

- **Input** NFA (Σ, Q, q₀, Fₙ, δ), **Output** DFA (Σ, R, r₀, Fᵈ, δ)
- **Algorithm**

  Let \( r₀ = \varepsilon\text{-closure}(q₀) \), 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 \( S = \{ s \mid q \in r \text{ & } \text{move}(q,a) = s \} \)  
  // states reached via \( a \)

  Let \( e = \varepsilon\text{-closure}(S) \)  
  // 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 \( Fᵈ = \{ r \mid \exists s \in r \text{ with } s \in Fₙ \} \)  
  // final if include state in \( Fₙ \)
NFA → DFA Example 1

\[ \Sigma = \{0, 1\} \]
NFA $\rightarrow$ DFA Example 1

A diagram of an NFA and its corresponding DFA is shown, illustrating the conversion process from NFA to DFA.
NFA $\rightarrow$ DFA Example 2

- NFA
- DFA

$R = \{ \}$
NFA → DFA Practice
NFA $\rightarrow$ DFA Practice
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: 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

J. Hopcroft, “An $n \log n$ algorithm for minimizing states in a finite automaton,” 1971
Minimizing DFA: Example 1

- DFA

- Initial partitions

- Split partition
Minimizing DFA: Example 1

- **DFA**

  ![DFA Diagram]

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

- **Split partition? → Not required, minimization done**
  - \( \text{move}(S,a) = T \in P2 \) \quad \text{–} \quad \text{move}(S,b) = R \in P1 \)
  - \( \text{move}(T,a) = T \in P2 \) \quad \text{–} \quad \text{move}(T,b) = R \in P1 \)
Minimizing DFA: Example 3
Minimizing DFA: Example 3

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