Nondeterministic Finite Automata (NFA)

CS 536
Scanner: converts a sequence of characters to a sequence of tokens
Scanner and parser: master-slave relationship
Scanner implemented using FSMs
FSM: DFA or NFA
This Lecture

NFAs from a formal perspective

Theorem: NFAs and DFAs are equivalent

Regular languages and Regular expressions
NFAs, formally

\[ M \equiv (Q, \Sigma, \delta, q, F) \]

finite set of states

start state \( q \in Q \)

final states \( F \subseteq Q \)

the alphabet (characters)

transition function

\[ \delta : Q \times \Sigma \rightarrow 2^Q \]

\[
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
s1 & \{s1\} & \{s1, s2\} \\
s2 & & \\
\end{array}
\]
NFA

To check if string is in $L(M)$ of NFA $M$, simulate set of choices it could make.

At least one sequence of transitions that:
- Consumes all input (without getting stuck)
- Ends in one of the final states

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>st</td>
</tr>
<tr>
<td>s1</td>
<td>s1</td>
<td>s2</td>
</tr>
<tr>
<td>s1</td>
<td>s1</td>
<td>s1</td>
</tr>
<tr>
<td>s1</td>
<td>s1</td>
<td>s1</td>
</tr>
</tbody>
</table>
NFA and DFA are Equivalent

Two automata $M$ and $M'$ are equivalent iff $L(M) = L(M')$

Lemmas to be proven

Lemma 1: Given a DFA $M$, one can construct an NFA $M'$ that recognizes the same language as $M$, i.e., $L(M') = L(M)$

Lemma 2: Given an NFA $M$, one can construct a DFA $M'$ that recognizes the same language as $M$, i.e., $L(M') = L(M)$
Lemma 2: Given an NFA $M$, one can construct a DFA $M'$ that recognizes the same language as $M$, i.e., $L(M') = L(M)$

Part 1: Given an NFA $M$ without $\epsilon$–transitions, one can construct a DFA $M'$ that recognizes the same language as $M$

Part 2: Given an NFA $M$ with $\epsilon$–transitions, one can construct an NFA $M'$ without $\epsilon$–transitions that recognizes the same language as $M$
NFA w/o $\varepsilon$–Transitions to DFA

NFA $M$ to DFA $M'$

**Intuition:** Use a single state in $M'$ to simulate a set of states in $M$

$M$ has $|Q|$ states

$M'$ can have only up to $2^{|Q|}$ states
Defn: let $\text{succ}(s,c)$ be the set of choices the NFA could make in state $s$ with character $c$

- $\text{succ}(A,x) = \{A, B\}$
- $\text{succ}(A,y) = \{A\}$
- $\text{succ}(B,x) = \{C\}$
- $\text{succ}(B,y) = \{C\}$
- $\text{succ}(C,x) = \{D\}$
- $\text{succ}(C,y) = \{D\}$

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{A, B}</td>
<td>{A}</td>
</tr>
<tr>
<td>B</td>
<td>{C}</td>
<td>{C}</td>
</tr>
<tr>
<td>C</td>
<td>{D}</td>
<td>{D}</td>
</tr>
<tr>
<td>D</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

NFA w/o $\varepsilon$–Transitions to DFA
To build DFA: Add an edge from state $S$ on character $c$ to state $S'$ if $S'$ represents the set of all states that a state in $S$ could possibly transition to on input $c$.
Proving Lemma 2

Lemma 2: Given an NFA M, one can construct a DFA M’ that recognizes the same language as M, i.e., $L(M’) = L(M)$

Part 1: Given an NFA M without $\varepsilon$–transitions, one can construct a DFA M’ that recognizes the same language as M

Part 2: Given an NFA M with $\varepsilon$–transitions, one can construct an NFA M’ without $\varepsilon$–transitions that recognizes the same language as M
**\(\varepsilon\)-transitions**

*E.g.*: \(x^n\), where \(n\) is even or divisible by 3

Useful for taking union of two FSMs

In example, left side accepts even \(n\); right side accepts \(n\) divisible by 3
Eliminating $\varepsilon$-transitions

We want to construct $\varepsilon$-free NFA $M'$ that is equivalent to $M$

**Definition: Epsilon Closure**

$\text{eclose}(s) = \text{set of all states reachable from } s \text{ using zero or more epsilon transitions}$

<table>
<thead>
<tr>
<th>$s$</th>
<th>$\text{eclose}(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>${P, Q, R}$</td>
</tr>
<tr>
<td>$Q$</td>
<td>${Q}$</td>
</tr>
<tr>
<td>$R$</td>
<td>${R}$</td>
</tr>
<tr>
<td>$Q_1$</td>
<td>${Q_1}$</td>
</tr>
<tr>
<td>$R_1$</td>
<td>${R_1}$</td>
</tr>
<tr>
<td>$R_2$</td>
<td>${R_2}$</td>
</tr>
</tbody>
</table>
Make p an accepting state of N' iff ECLOSE(p) contains an accepting state of N.

Add an arc from p to q labeled a iff there is an arc labeled a in N from some state in ECLOSE(p) to q.

Delete all arcs labeled $\varepsilon$. 


Proving Lemma 2

**Lemma 2**: Given an NFA $M$, one can construct a DFA $M'$ that recognizes the same language as $M$, i.e., $L(M') = L(M)$

**Part 1**: Given an NFA $M$ without $\varepsilon$–transitions, one can construct a DFA $M'$ that recognizes the same language as $M$

**Part 2**: Given an NFA $M$ with $\varepsilon$–transitions, one can construct an NFA $M'$ without $\varepsilon$–transitions that recognizes the same language as $M$
Summary of FSMs

DFAs and NFAs are equivalent

An NFA can be converted into a DFA, which can be implemented via the table-driven approach.

$\varepsilon$-transitions do not add expressiveness to NFAs.

Algorithm to remove $\varepsilon$-transitions.
Regular Languages and Regular Expressions
Regular Language

Any language recognized by an FSM is a regular language

Examples:
- Single-line comments beginning with `//`
- Integer literals
- `{ε, ab, abab, ababab, abababab, .... }`
- C/C++ identifiers
Regular Expression

A pattern that defines a regular language

Regular language: set of (potentially infinite) strings

Regular expression: represents a set of (potentially infinite) strings by a single pattern

\{ \varepsilon, ab, abab, ababab, abababab, \ldots \} \Leftrightarrow (ab)^*
Why do we need them?

Each token in a programming language can be defined by a regular language

Scanner-generator input: one regular expression for each token to be recognized by scanner

Regular expressions are inputs to a scanner generator
Regular Expression

**operands:** single characters, epsilon

**operators:** from low to high precedence

“or”: \( a | b \)

“followed by”: \( a.b, \ ab \)

“Kleene star”: \( a^* \) (0 or more a-s)
Regular Expression

Conventions:

aa is a . a
a+ is aa*
letter is a|b|c|d|…|y|z|A|B|…|Z
digit is 0|1|2|…|9
not(x) all characters except x
. is any character
() parentheses for grouping, e.g., (ab)* is {ε, ab, abab, ababab,
Regexp, example

Precedence: * > . > |
digit | letter letter
   (digit) | (letter . letter)
one digit, or two letters
digit | letter letter*
   (digit) | (letter . (letter)*)
one digit, or one or more letters
digit | letter+

23
Regexp, example

Hex strings

- start with 0x or 0X
- followed by one or more hexadecimal digits
- optionally end with l or L

0(x|X)hexdigit+(L|l|ε)

where hexdigit = digit|a|b|c|d|e|f|A|…|F
Regexp, example

Integer literals: sequence of digits preceded by optional +/-

Example: -543, +15, 0007

Regular expression

(+|-|ε)digit+
Regexp, example

Single-line comments
  Example: // this is a comment

Regular expression
  //(not(‘\n’))*’\n’
Regexp, example

C/C++ identifiers: sequence of letters/digits/underscores; cannot begin with a digit; cannot end with an underscore

Example: a, _bbb7, cs_536

Regular expression

letter | (letter|_)(letter|digit|_)*letter|digit
Recap

Regular Languages

Languages recognized/defined by FSMs

Regular Expressions

Single-pattern representations of regular languages

Used for defining tokens in a scanner generator
Creating a Scanner

Scanner Generator

Last lecture: DFA to code

This lecture: NFA to DFA

Next lecture: Regexp to NFA

This lecture: token to Regexp

= Scanner