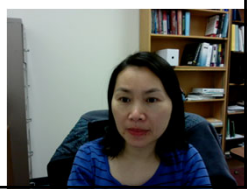
# CSCI-C311 Programming Languages

## Regular Expressions and Finite Automata

Dr. Hang Dinh



1

## Outline and Reading

- After this lecture, you will learn
  - Regular expressions
  - Deterministic Finite Automata (DFA)
  - Nondeterministic Finite Automata (NFA)
  - Relationships among regular expressions, DFA, and NFA
- Reading
  - Scott 4e - Section 2.1.1 Tokens and Regular Expressions
  - Scott 4e – Section 2.2.1 Generating a Finite Automata
  - Scott 4e – Section 2.4.1 Finite Automata (on the companion site)

2

# Why Regular Expressions

- Regular expressions are used to implement scanning (lexical analysis)
- Outside of compilers, regular expressions are used in many applications that are reduced to the **Pattern Matching** problem:
  - Validate email and/or password formats on the server side
  - Extract specific sections from an HTML page
  - Parse text data files into sections for import into a database
  - Replace values in text to clean, reformat, or change content
- Examples of particular applications using regular expressions:
  - utilities such as AWK and GREP in UNIX
  - text editors, file finders, etc

3

# What are Regular Expressions?

- A regular expression is one of the following:
  - A character (in a chosen alphabet such as the ASCII set)
  - The empty string, denoted by $\varepsilon$
  - Two regular expressions concatenated (Concatenation)
  - Two regular expressions separated by | (Alteration)
    - Some authors use $\cup$ or + instead of | to denote alteration
  - A regular expression followed by the ***** (Kleene closure)
- Examples:
  - 0
  - 1
  - 0 | 1
  - 0 | 1 | 2
  - + | - | $\varepsilon$
  - (0 | 1) (+ | - | $\varepsilon$)
  - (0 | 1 | 2)*
  - 0*

4

# Meaning of Regular Expressions

- A regular expression generates a set of strings (called a ***regular set*** or a ***regular language***) based on the following rules

| Regular Expression | Strings generated |
|---|---|
| ε | The empty string |
| Character $c$ | string $c$ |
| Two regular expressions concatenated | any string generated by the first one followed by any string generated by the second one |
| Two regular expressions separated by \| | any string generated by the first one OR any string generated by the second one |
| A regular expression followed by the Kleene star $*$ | The concatenation of zero or more strings generated by the expression in front of the star |

5

# Meaning of Regular Expressions

- Examples

| Regular Expression | Strings generated |
|---|---|
| 1 | 1 |
| 0 \| 1 | 0, 1 |
| +\| −\| ε | +, −, empty string |
| (0 \| 1) (+ \| −\| ε) | 0+, 0-, 0, 1, 1+, 1- |
| (0 \| 1)* | Any binary string |
| (0 \| 1)*(+ \| −\| ε) | Any binary string, and any binary string followed by a + or a − |

6

# Named Regular Expressions

- If a regular expression is too complicated, we give it a name
  - By using grammar rule, with the name on the left of the arrow, and the named regular expression on the right

$$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

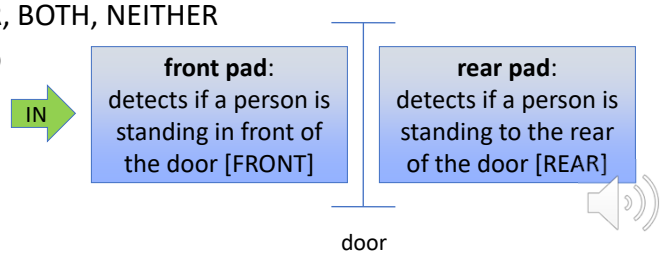
  - We then use named regular expressions to define other regular expressions

$$integer \rightarrow digit \ digit^*$$
$$decimal \rightarrow digit^* \ (. \ digit \mid digit \ .) \ digit^*$$
$$exponent \rightarrow (\mathrm{e} \mid \mathrm{E}) \ (+ \mid - \mid \varepsilon) \ integer$$
$$real \rightarrow integer \ exponent \mid decimal \ (exponent \mid \epsilon)$$
$$number \rightarrow integer \mid real$$

7

# Finite Automata

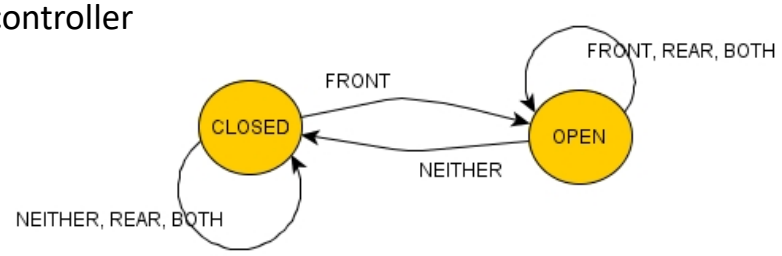
- To recognize if a string is generated by a given regular expression, we build a ***finite automaton*** for the regular expression.
- Finite automata, or finite-state machines, are models for computers with extremely limited memory.
  - used in various electromechanical devices
- Application Example: automatic door controller
  - Possible input signals: FRONT, REAR, BOTH, NEITHER
  - in either of states: OPEN or CLOSED




8

# Finite Automata Example

- Automatic door controller
  - state diagram:



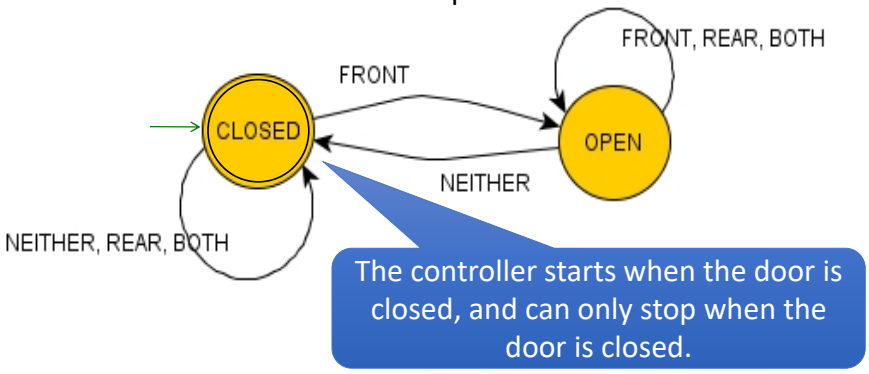

  - state transition:

| | NEITHER | FRONT | REAR | BOTH |
|---|---|---|---|---|
| CLOSED | CLOSED | OPEN | CLOSED | CLOSED |
| OPEN | CLOSED | OPEN | OPEN | OPEN |

9

# Finite Automata Example

- Automatic door controller
  - Start state: when the machine starts
  - Final states: when the machine can stop




10

# Finite Automata Formal Definition

- A finite automaton consists of five components:
  1. a finite set $S$ of states
  2. a finite set $I$ of input symbols (called the *alphabet*)
  3. a transition function $f: S \times I \rightarrow S$ representing state transition
  4. the start state $s_0 \in S$
  5. the set of final states (also called accept states) $F \subseteq S$
- This is definition of a deterministic finite automaton (DFA)
  - From a current state and a current input symbol, there's only one possible choice to move to the next state.

11

11

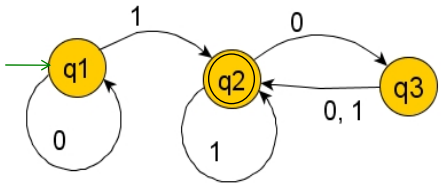
# Finite Automata Formal Definition

- Example: a finite automaton $M_1$ with five components:
  - The set of states $S = \{q1, q2, q3\}$
  - Input Alphabet $I = \{0, 1\}$
  - Transition function $f$ is described by the table:
  - The start state $q1$
  - The set of final states $F = \{q2\}$.

| $f$ | 0 | 1 |
|---|---|---|
| q1 | q1 | q2 |
| q2 | q3 | q2 |
| q3 | q2 | q2 |

$f(q3,1)=q2$

- State diagram of automaton $M_1$:




12

# Computation of DFA

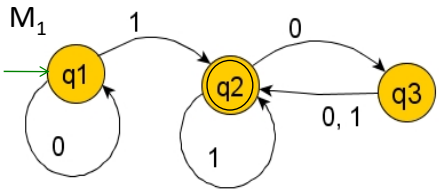
- Given a string of input symbols, a DFA will run as follows:
  - One by one, it reads an input symbol from the given string, using the transition function to move from state to state
  - When the final input symbol has been read, it will "accept" if it's in a final state; otherwise, it will "reject"

| input | 1 | 1 | 0 | |
|---|---|---|---|---|
| state | q1 | q2 | q2 | q3 |

**reject**

| input | 0 | 1 | 0 | 1 | |
|---|---|---|---|---|---|
| state | q1 | q1 | q2 | q3 | q2 |

**accept**

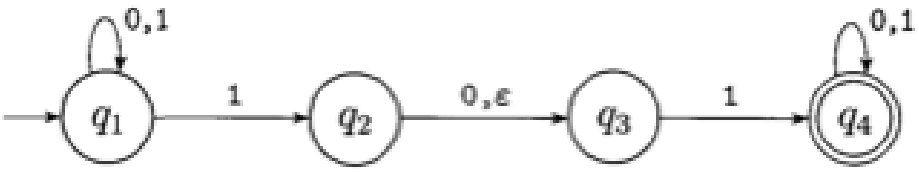



- The set of strings accepted by a DFA *M* is called the language of *M*

13

# Nondeterministic Finite Automata (NFA)

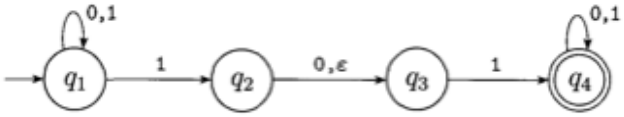
- Every DFA is a nondeterministic finite automaton (NFA)
- An NFA may have two additional features
  - Some states may have many exiting transitions with the same input symbol
  - Some arrows may have label $\varepsilon$, i.e., can proceed without reading any input.

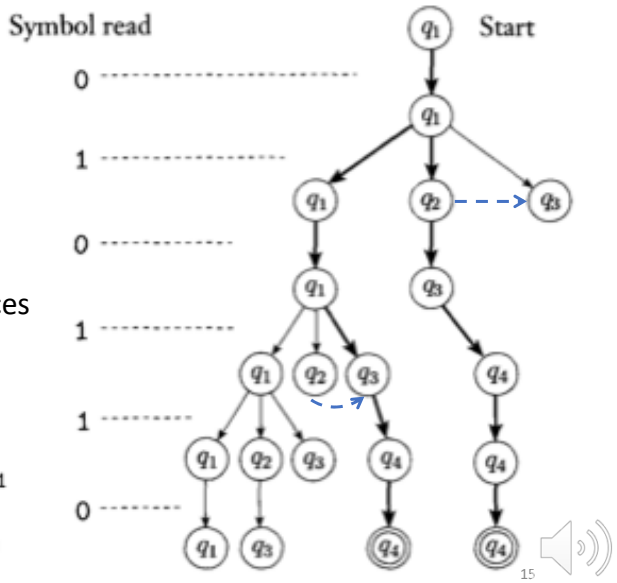



14

# Computation of NFA

- Parallel computation
  - follows all choices in parallel
- Computation tree
  - Root: start state
  - Each branching point corresponds to a point of computation with multiple choices
- The NFA accepts if some branch ends in an accept state.



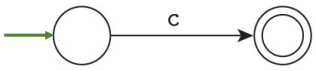

15

15

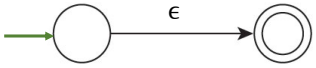
# From a Regular Expression to NFA

- Ultimate goal: for any regular expression *R*, build a DFA *M* such that

  Any string is generated by *R* if and if it is accepted by *M*.
- No algorithm for direct conversion from regular expression to DFA.
- The first step is to convert from regular expression to NFA
  - Base case: regular expression consisting of a single character *c*




  - Base case: regular expression representing the empty string $\epsilon$




16

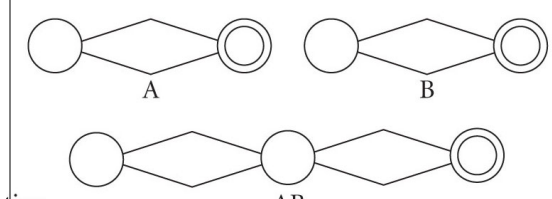
# From a Regular Expression to NFA

- NFAs for regular expressions in base case satisfy:
  1. There are no transitions into the start state
  2. There is a single final state
  3. There are no transitions out of the final state
- Use three subconstructions to build larger NFAs to represent
  - Concatenation
  - Alternation
  - Kleene closure
- Each subconstruction preserves three conditions (invariants) above.

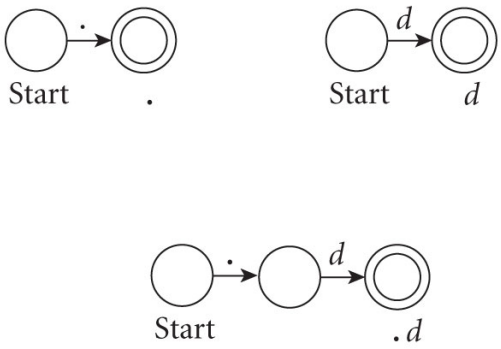
17

# From a Regular Expression to NFA: Concatenation

- General construction:



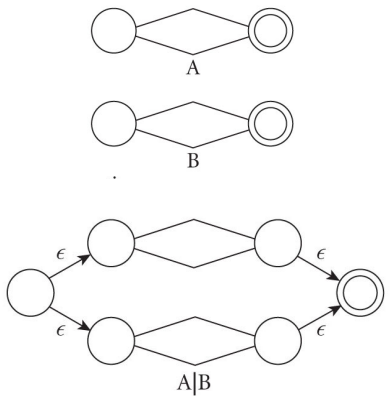

- Example:



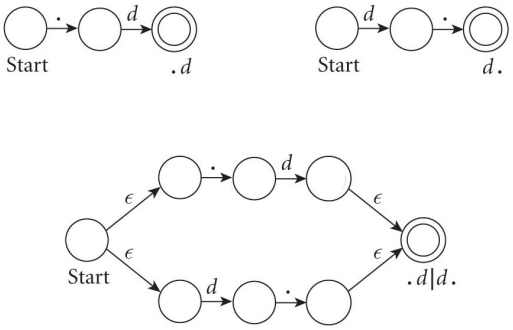

18

# From a Regular Expression to NFA: Alternation

- General construction:



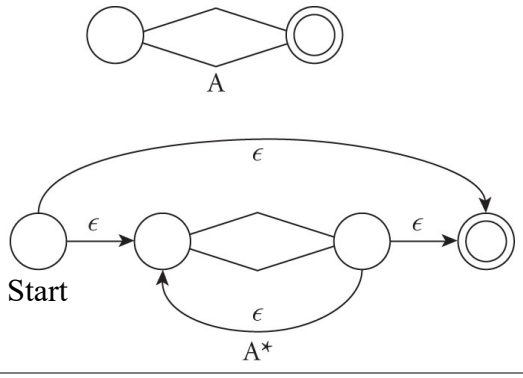

- Example:



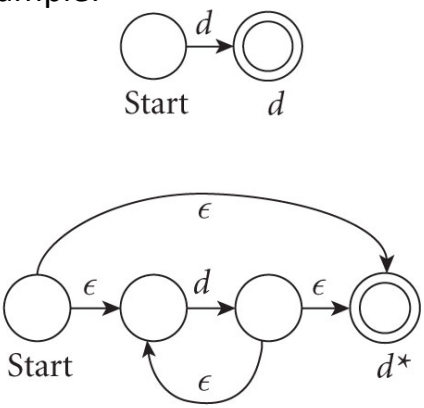

19

# From a Regular Expression to NFA: Kleene Closure

- General construction:




- Example:

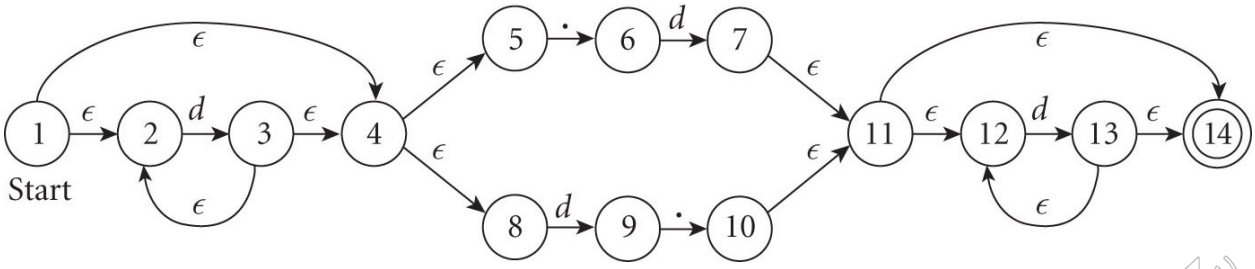



20

# From a Regular Expression to NFA: Complete Example

- NFA for the regular expression representing a decimal number:

$$d^*(.\ d \mid d\ .)\ d^*$$




21

# Equivalence of DFA, NFA, and Regular Expression

- Equivalence of DFA and NFA
  - Every DFA is an NFA
  - Every NFA has an equivalent DFA (next lecture: convert NFA to DFA)
  - "Equivalent" here means they accept the same set of strings
- Equivalence of regular expressions and DFA
  - Every regular expression has an equivalent DFA
  - Every DFA has an equivalent regular expression (see proof in Scott 4e 2.4.1)
  - "Equivalent" here means the set of strings recognized by the regular expression equals the set of strings accepted by the DFA.

22