CS 160 Compilers

Lecture 6: Regular Expressions and Finite State Machine

Yu Feng

Fall 2021

Outline

- Last time: Specify [illegible]l structure using regular expressions

- Today: How to recognize strings matching regular expressions using finite automata.

- We will see deterministic finite automata (DFAs) and non-deterministic finite automata (NFAs)

- High-level story: RegEx -> NFA -> DFA -> Table

Finite automata

- Regular Expressions ⇔ Specification
- Finite Automata ⇔ Implementation
- A finite automata formally consists of:
  - An input alphabet Σ
  - A set of states S
  - A start state n
  - A set of accepting states $F \subseteq S$
  - A set of transitions state $\xrightarrow{\text{input}}$ state

Finite automata

- Transition $S_1 \rightarrow^{\alpha} S_2$

- This means: In state S_1 and input character α, go to state S_2

- If end of input and in accepting state $\Rightarrow$ accept

- Otherwise $\Rightarrow$ reject

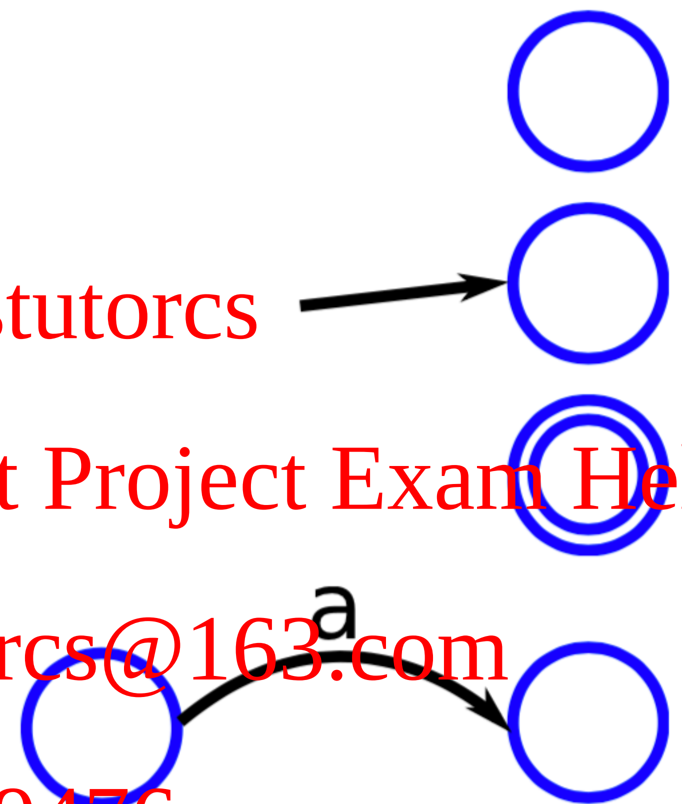
Finite Automata as State Graphs

A state:

The start state:

An accepting state:

A transition:



A simple example

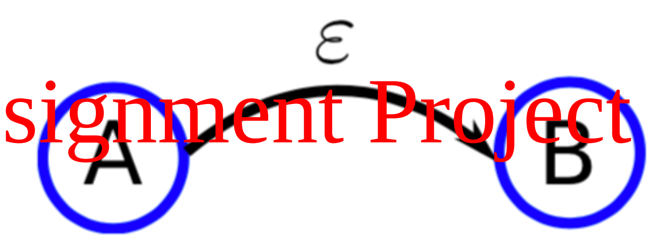
- Here is an automa[illegible]nly accepts the string ”1”:

Another simple example

- A finite automator[illegible]g any number of 1's followed by a single 0
- Alphabet: {0,1}

Epsilon transitions

- A special kind of t ε-transitions
- Machine can move from state A to B without reading any input

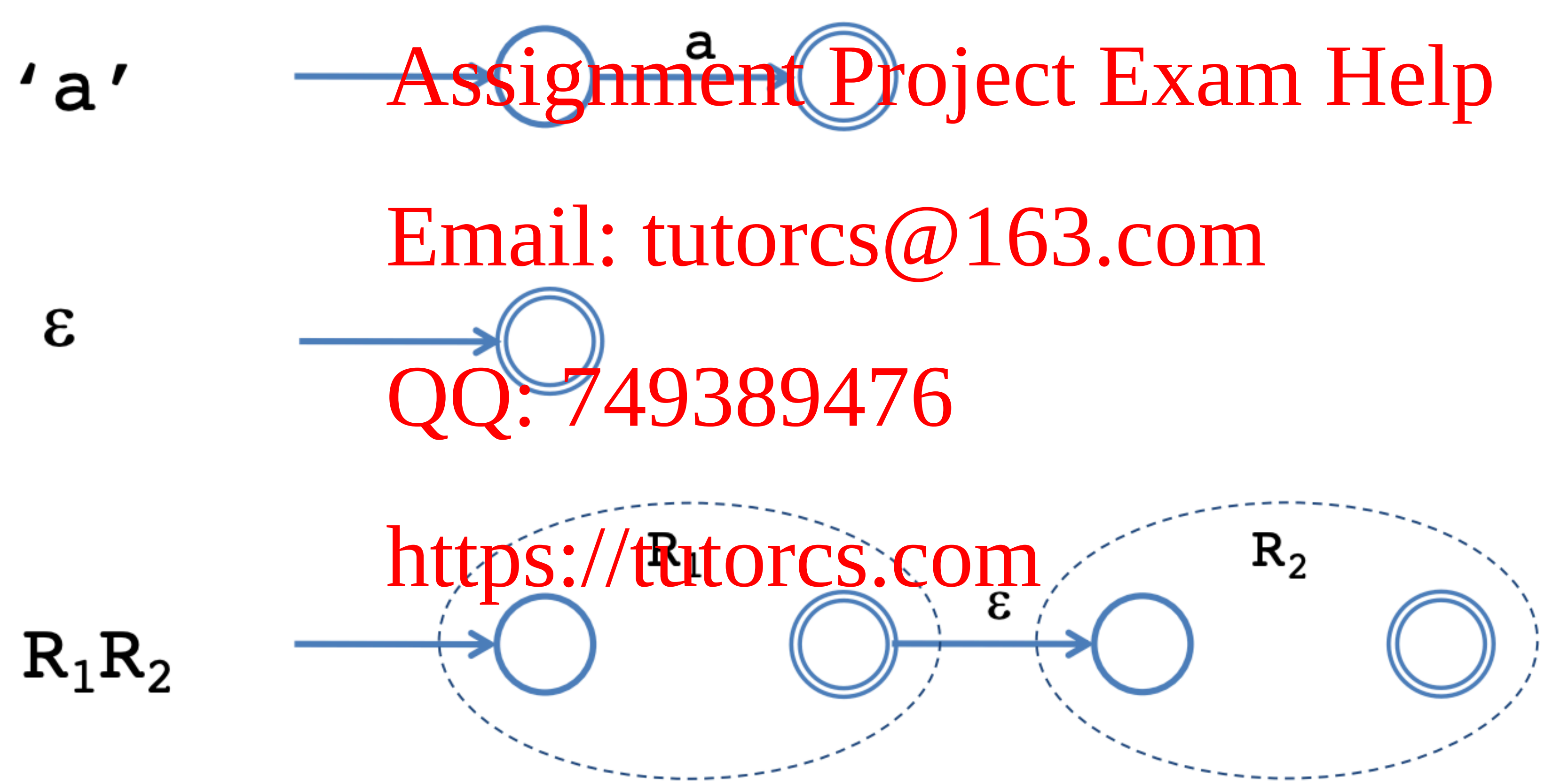


Deterministic and Nondeterministic Automata

- Deterministic Finite Automata (DFA)
  - At most one transition per input on any state
  - No ε moves
- Nondeterministic Finite Automate (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have ε-moves

RE to NFA

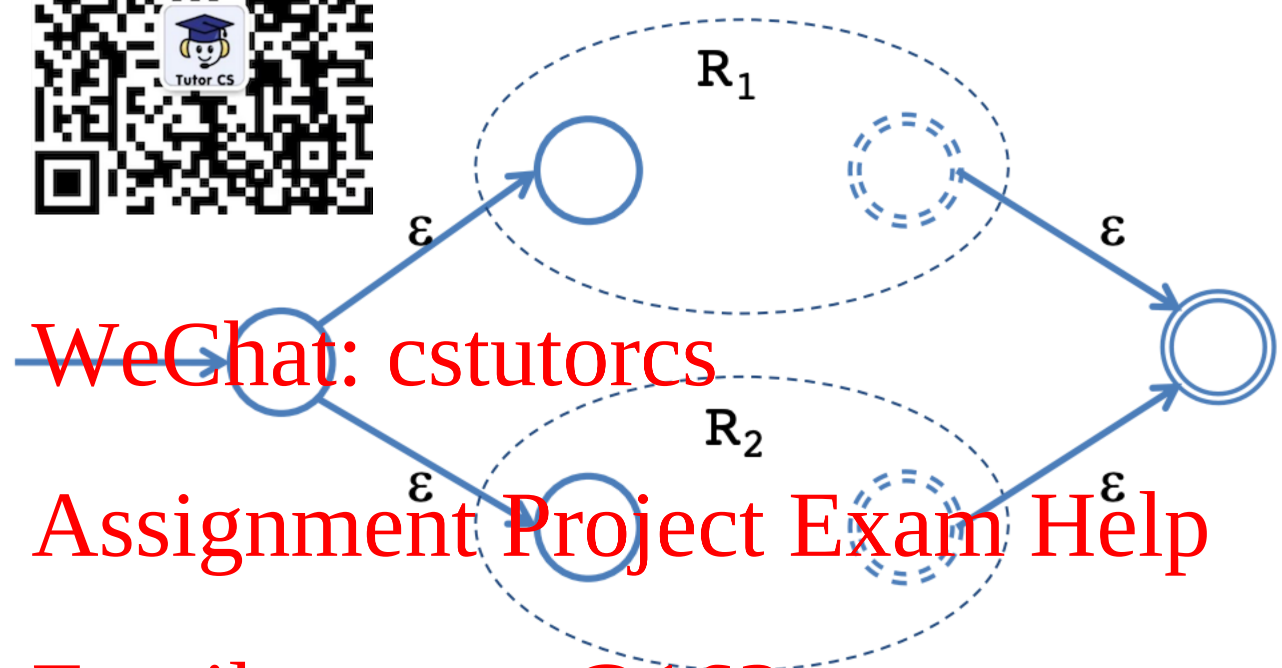
- Can we build a fin[illegible]aton for every regular expression
- Strategy: consider every possible regular expression (by induction on the structure of the regular expressions)



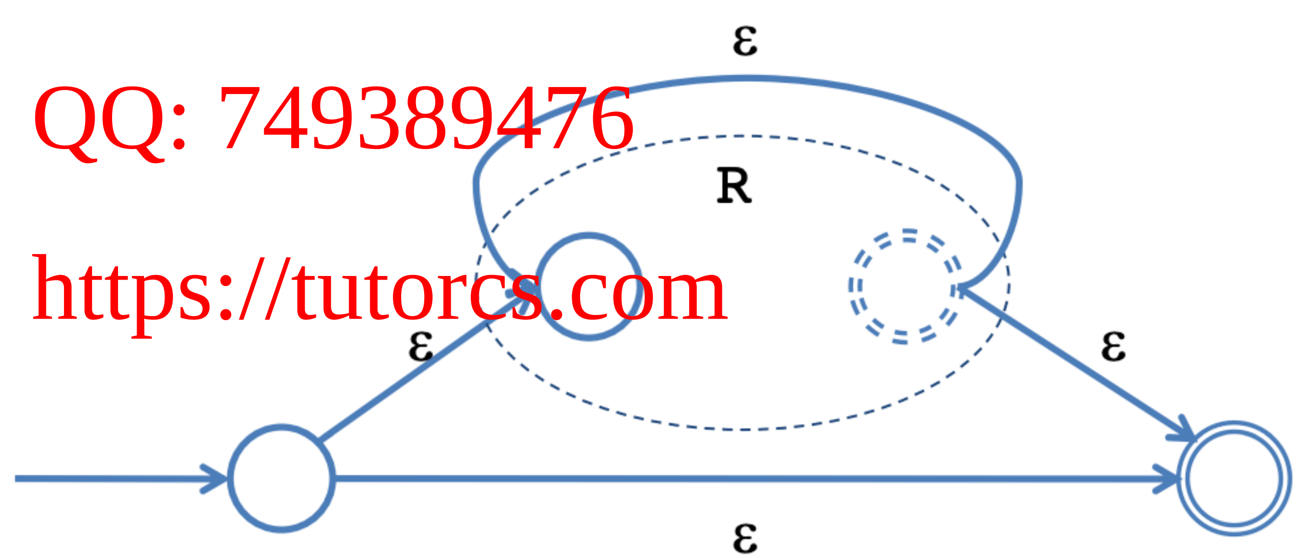
RE to NFA

- Can we build a fin[...]aton for every regular expression

$R_1 | R_2$



R*



NFA to DFA: The Trick

- Insight: Simulate th

- At any given time, the NFA is in a set of states

- State in the DFA ⇒ all (reachable) subsets of states in the NFA

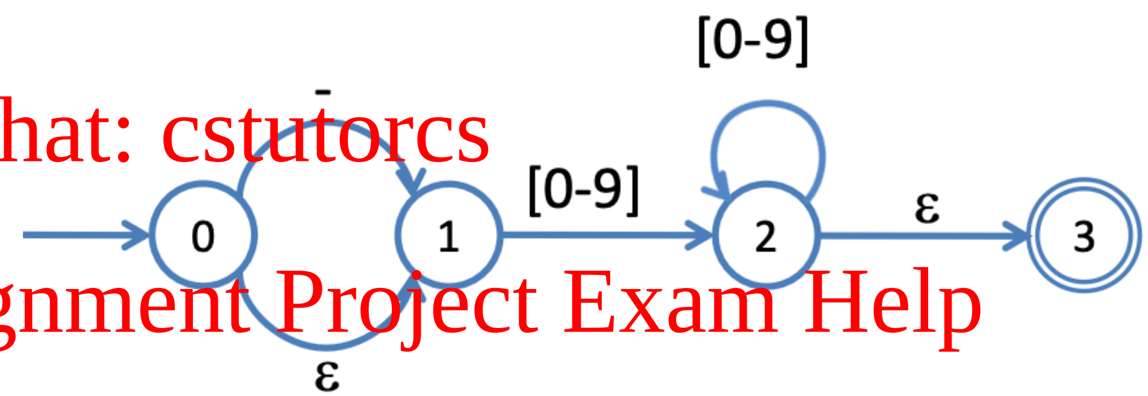
- Start State: the set of states reachable through ε moves from the NFA start state

- Add transition $A \rightarrow^{\alpha} B$ to DFA iff:

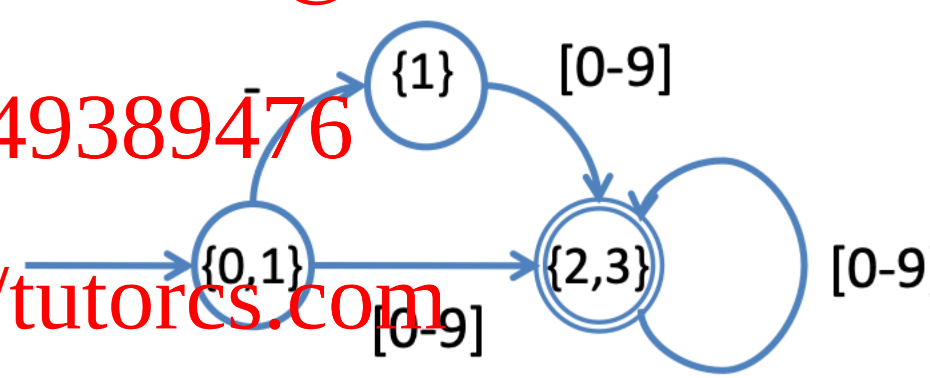
  - B is in the set of states reachable from any state in A after seeing input α, considering ε moves as well

NFA to DFA: Example

- Consider: -?[0-9]+
- NFA representation:



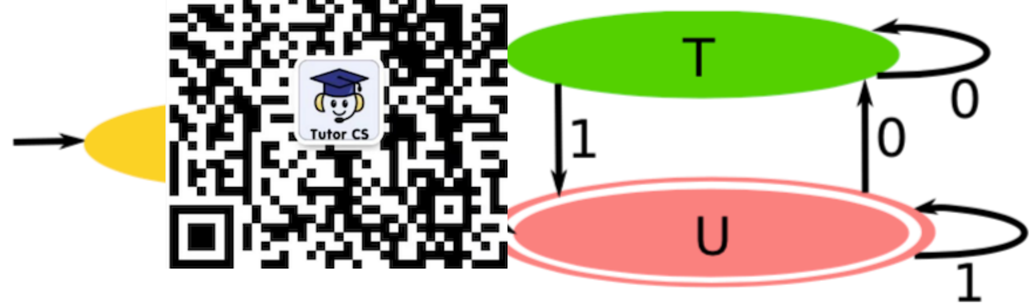
- DFA representation:



DFA: Implementation

- A DFA can be imp[illegible] by a 2D table T
  - One dimension is "states"
  - Other dimension is "input symbols"
  - For every transition $A \xrightarrow{c} B$ define T[A,c]=B
- DFA "execution": If in state A and input c, read T[A,c] = B and switch to state B
- Very efficient

Implementation of a DFA



	0	1
S	T	U
T	T	U
U	T	U

Translation from NFA to the table implementation is handled by modern lexer

TODOs by next lecture

- Hw2 will be out. G[illegible]ar with the Patina language
- Come to the discussion session if you have questions