Syntax

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

If you have a conflict with the first midterm, here is what you need to do ***before this Sunday***:

1. Get approval by sending me an email (to zhang@cse.psu.edu) which explains the conflict

2. Contact TA Mengran (mxf97@psu.edu) to schedule the exam

Regular Expression

Definition:

- A character
- Empty string (ε)
- Concatenation of two RE (e.g., (ab))
- Alternation of two RE, separated by “|” (e.g., (a|b))
- Closure (Kleene star) (e.g.(a*))

Scanning with RE

Read one character at a time and then

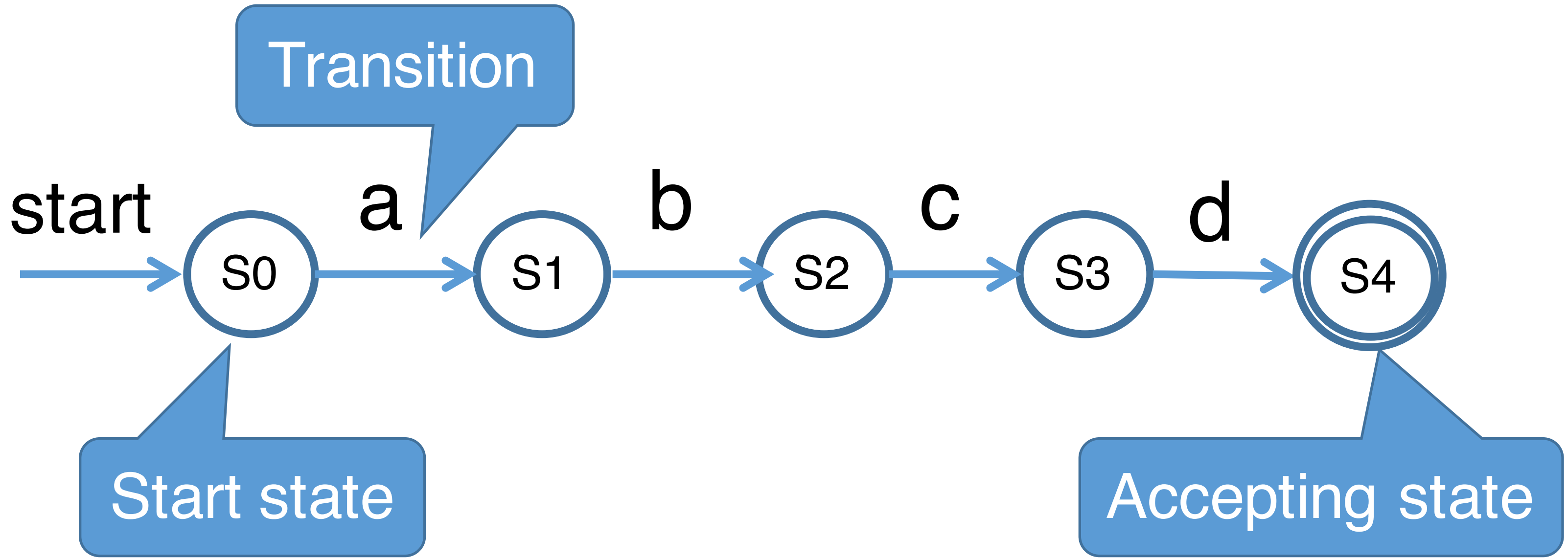
- output a token
- ignore character
- wait to see next character

```
// hello world
main()  /* main */
{for(;;)
 {printf ("Hello World!\n");}
}
```

```
ident("main") lparen rparen lbrace for lparen semi semi
rparen lbrace printf lparen string("Hello World!\n")
rparen semi rbrace rbrace
```

Finite State Automaton (FSA)

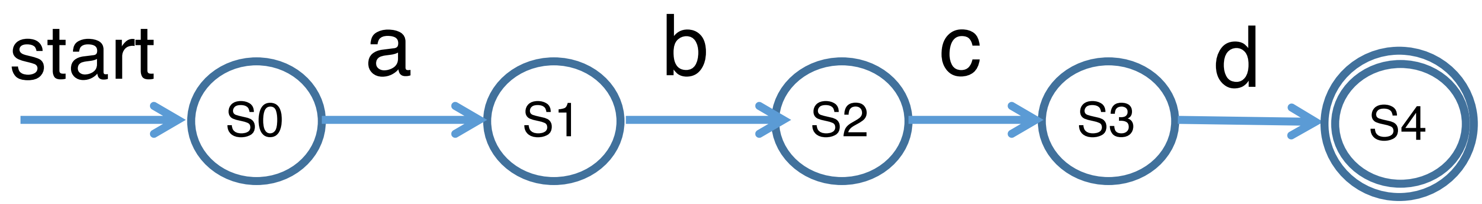
Recognize string “abcd”



Node: state
Edge: transitions

Binary Output:
{Yes, No}

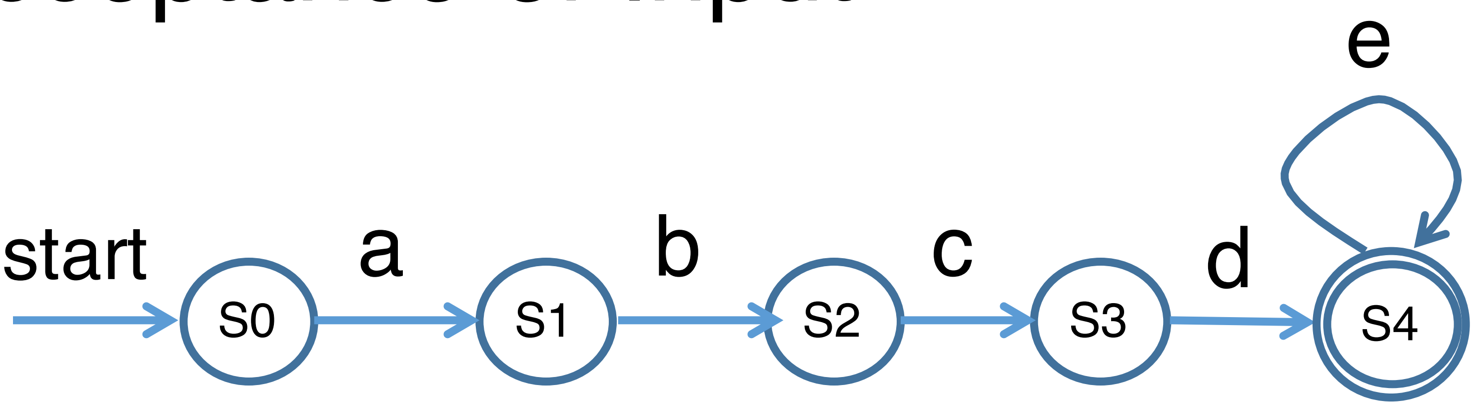
Acceptance of Input



An automaton accepts string *s* iff there is a path from the start state to one of the accepting states, such that the symbols along the path spell out *s*

E.g., the automaton above only accepts “abcd”

Acceptance of Input



E.g., the automaton above accepts "abcd", "abcde", "abcdee", …

Finite State Automaton

An automaton has

- A set of states (nodes)
- An input alphabet
- State transition function, (State, Char)→State
  (edges labeled by a set of characters)

a, stands for {a}, for simplicity

- A unique *start state* (node with "start" edge)
- One or more *accepting states* (with double circles)

Finite State Automaton

At each state

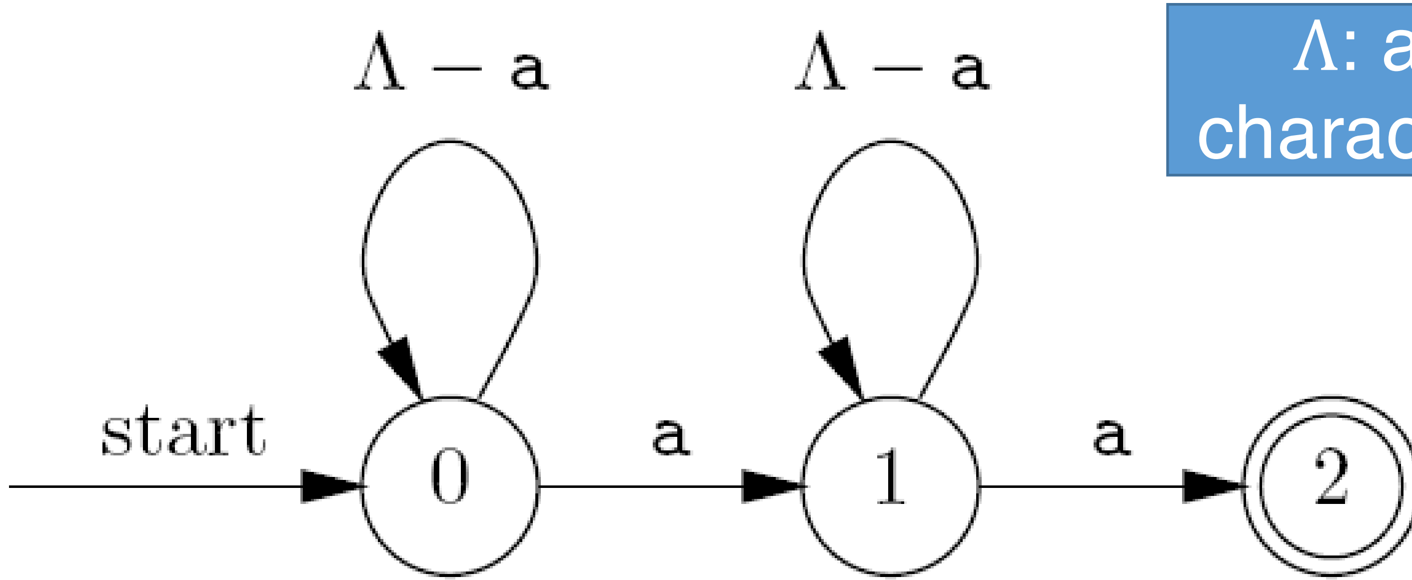
- Accepting state: accept (if end of string)
- Follow transition if it’s label includes next char.
- Otherwise: reject (die)

Deterministic FSA (DFSA)

- For each state and char., there is at most one outgoing edge (char. sets of outgoing edges are disjoint)

FSA Example

Automata that accepts string with two ‘a’s



Λ: all characters

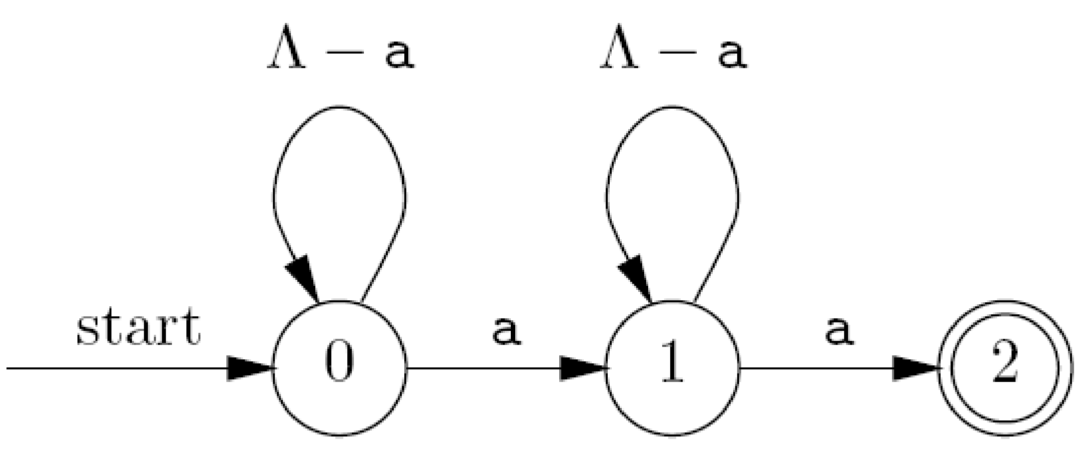
Deterministic? Two consecutive ‘a’s?

DFSA to Program

Translation to program is straightforward:

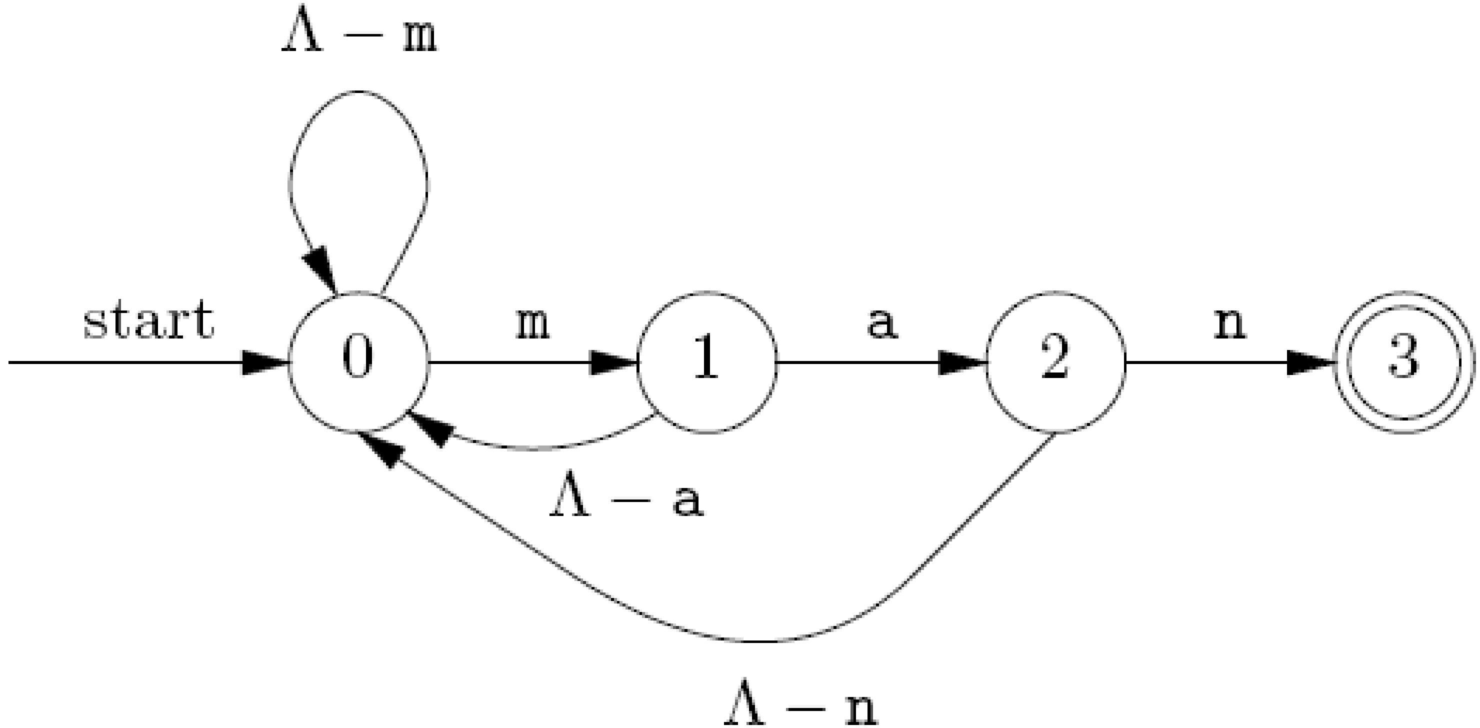
```
void s0() {
    char c = getchar();
    if (c == ‘a’) s1();
    else s0();
}
```

```
void s1() {
    char c = getchar();
    if (c == ‘a’) accept();
    else s1();
}
```



FSA Example

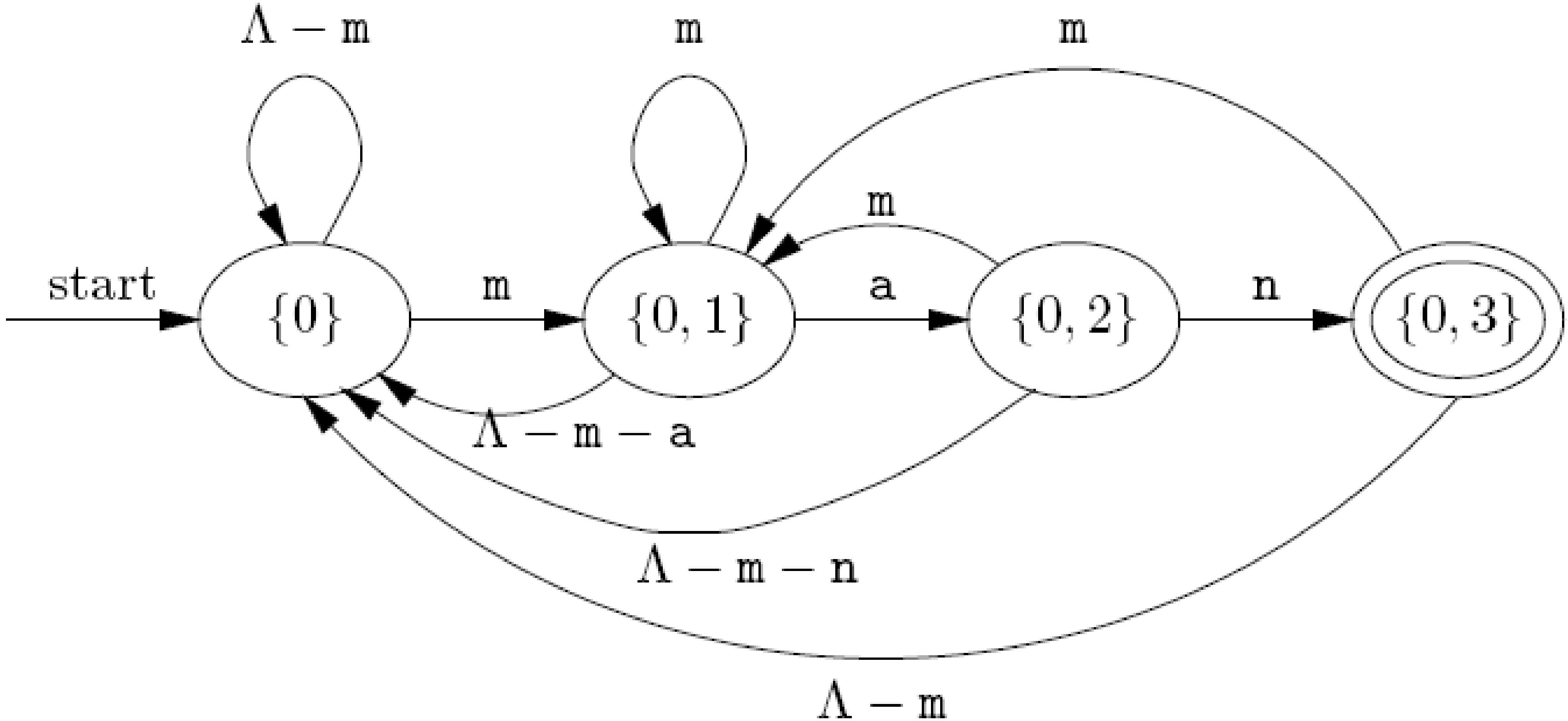
Recognize a string that ends in “man”



Accept “mman”?

FSA Example

Recognize a string that ends in “man”



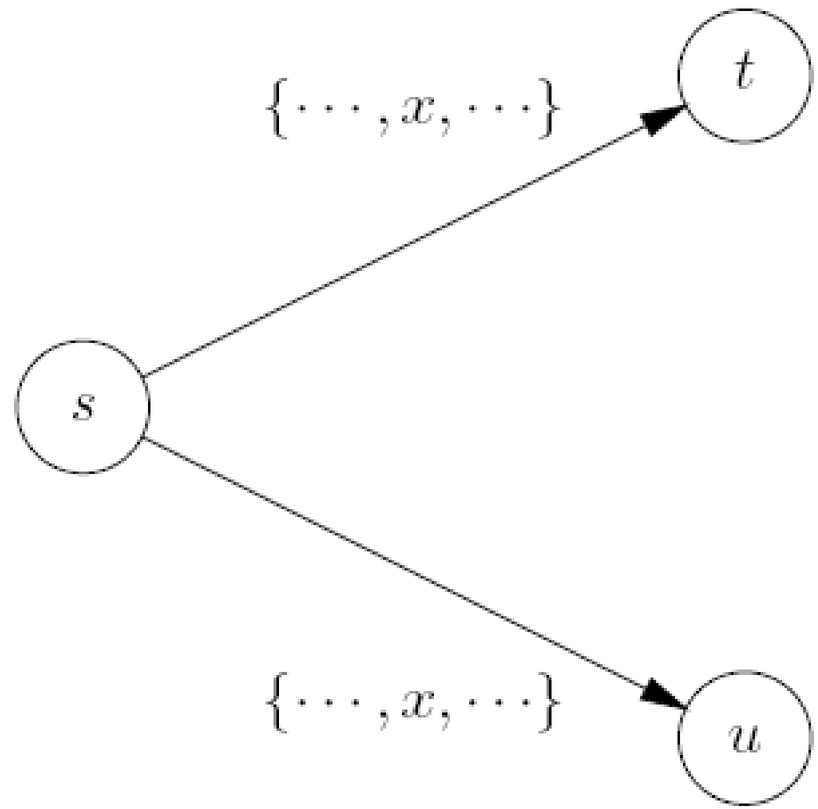
Too complex

Nondeterministic FSA

Nondeterministic FSA (NFSA)

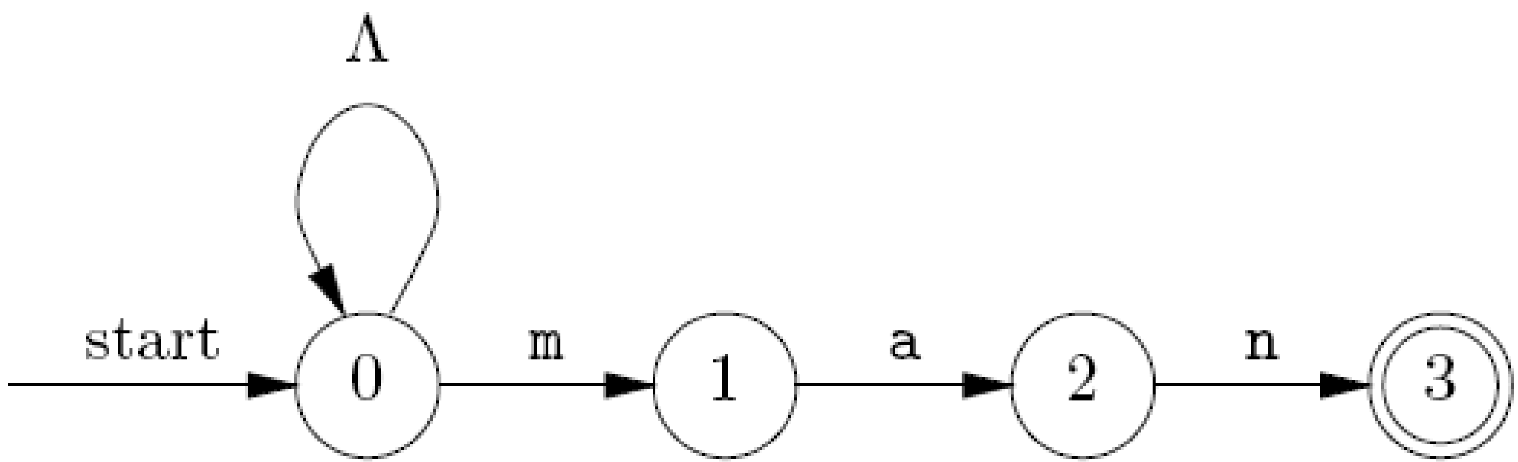
- For each state and char., there can be multiple outgoing edge

(ability to “guess”
the next step)



NFSA Example

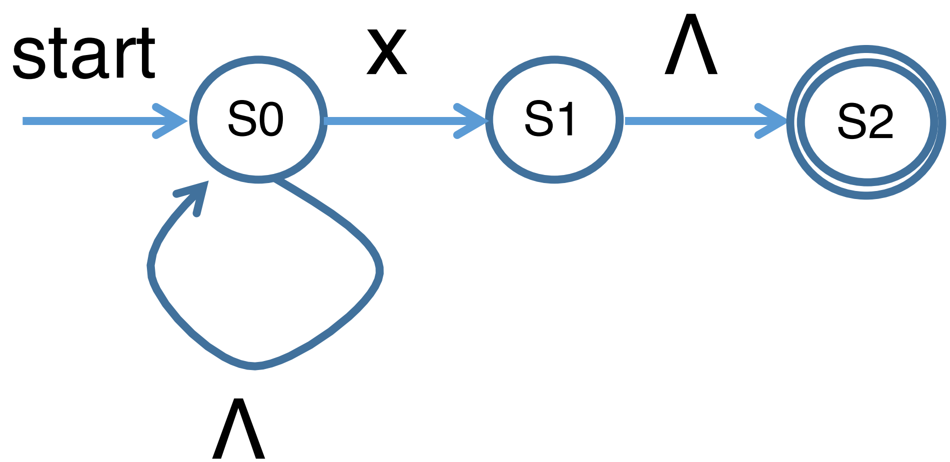
Recognize “man” in a string



Easier to design; harder to implement

NFSA Example

Recognize string whose second to last letter is “x”



Easier to design; harder to implement

NFSA to DFSA

NFSA and DFSA are equally “powerful”:

Every nondeterministic automaton can be replaced by a deterministic one, via *subset construction (not covered in this course)*.

Regular Expression to NFSA

RE and NFSA are equally "powerful":

Every regular expression R can be replaced by an NFSA (with ε*–transitions*) that accepts those strings in $L(R)$ and no others.

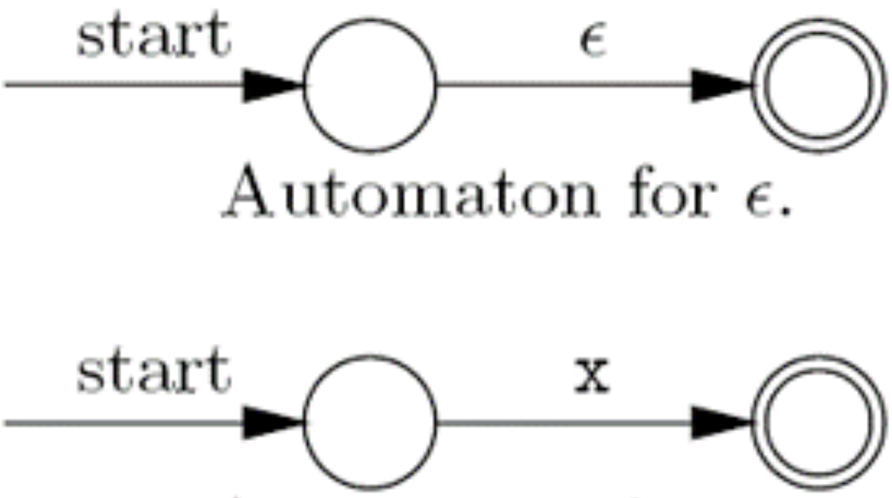
And, vise-versa *(not covered in this course)*.

FSA with ε*–transitions:*

automata with special character ε added to Λ

Regular Expressions to Automata

Base case



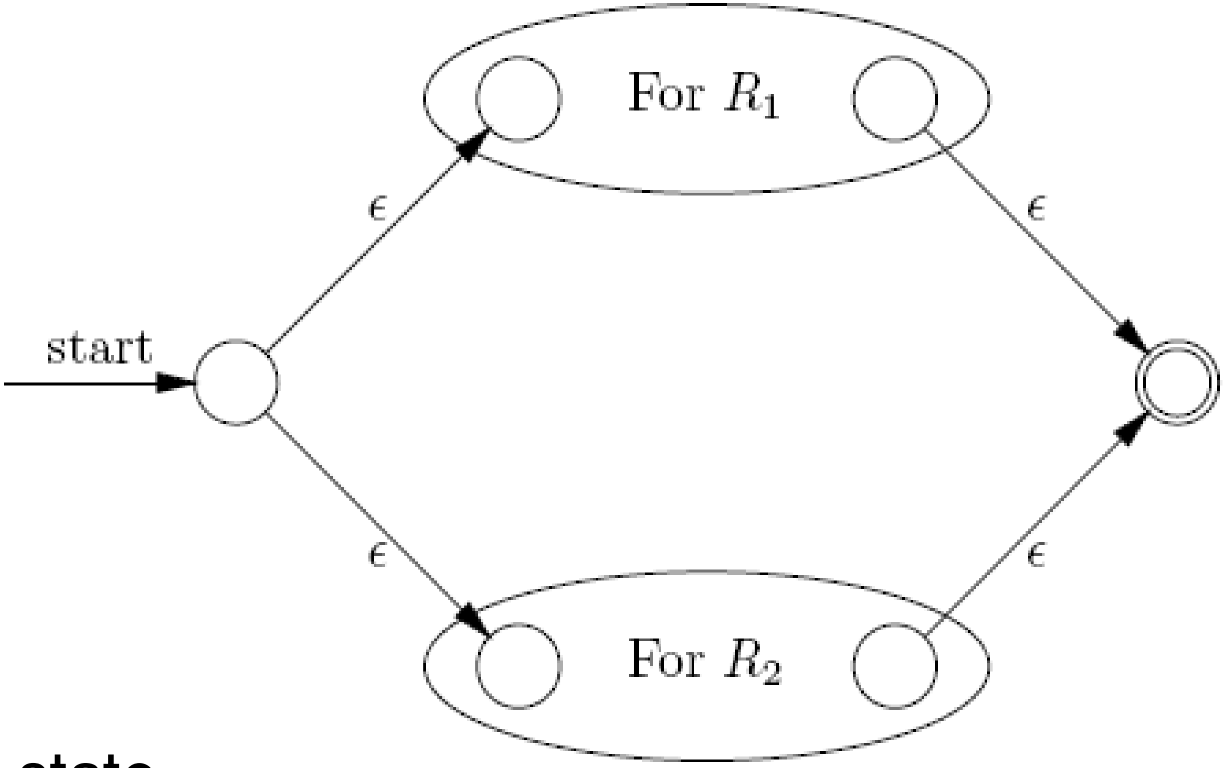
Automaton for ϵ.

Automaton for x.

One accepting state

Regular Expressions to Automata

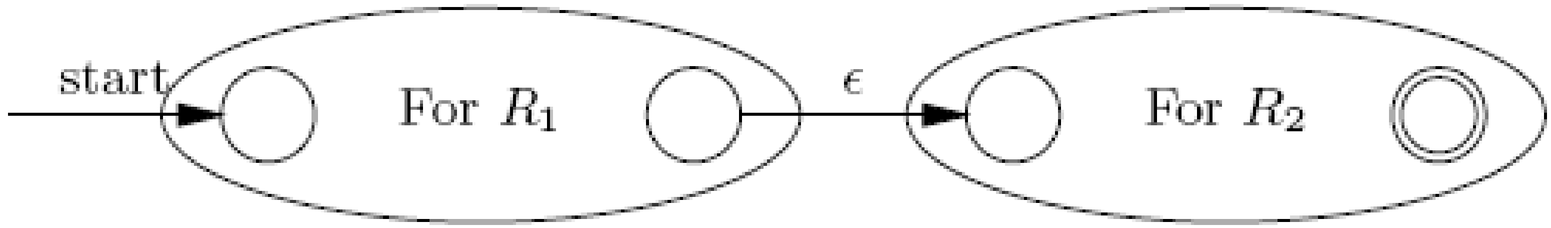
Alternation
$(R_1|R_2)$



One accepting state

Regular Expressions to Automata

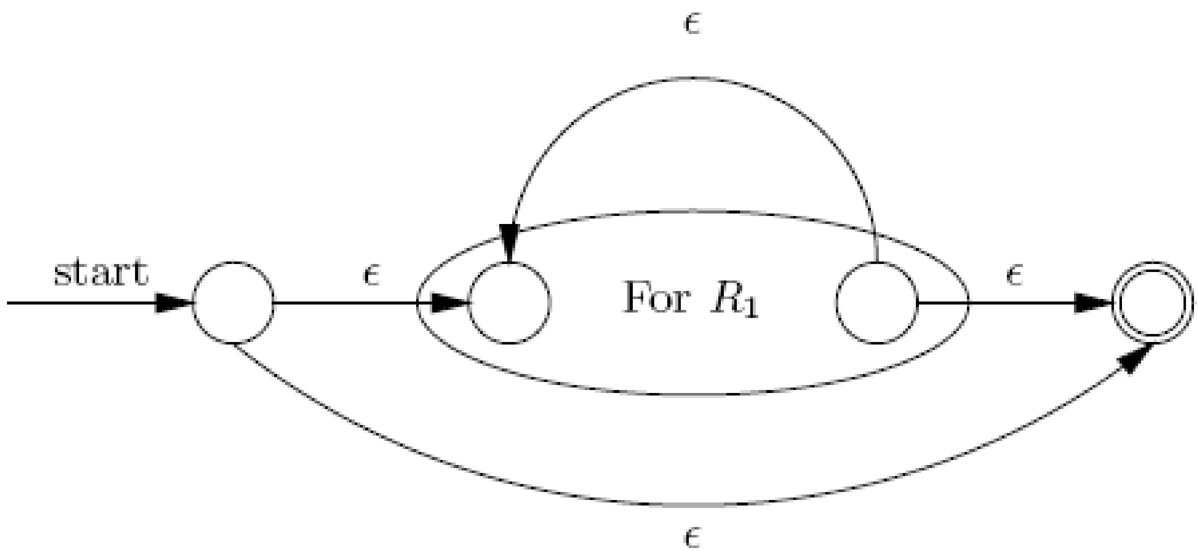
Concatination
$(R_1 R_2)$



One accepting state

Regular Expressions to Automata

Kleene star
$R_1{}^*$



One accepting state

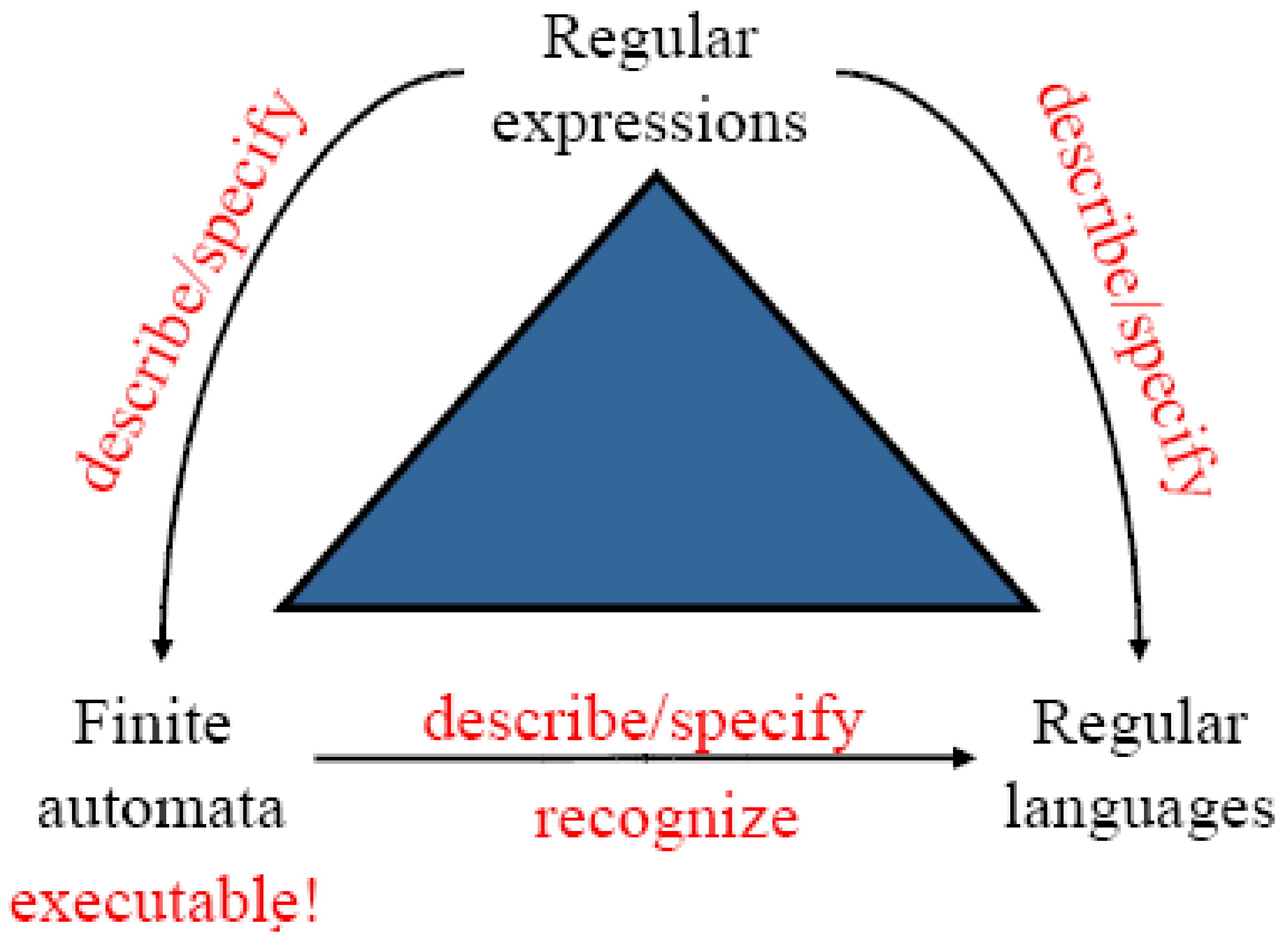
Regular Expressions to Scanner

RE → NFSA → DFSA → Scanner

Specification

Implementation

Scanner (Lexer) generator: automatically generate scanners, from a language specification



Regular expressions
describe/specify
describe/specify
Finite automata
executable!
describe/specify
recognize
Regular languages