

**CSE
31F**

Foundations of Computing I

* All slides are a combined effort between
previous instructors of the course

Building Precedence in Arithmetic Expressions

- **E** – expression (start symbol)
- **T** – term **F** – factor **I** – identifier **N** - number

E → **T** | **E+T**

T → **F** | **F*T**

F → (**E**) | **I** | **N**

I → **x** | **y** | **z**

N → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Backus-Naur Form (The same thing...)

BNF (Backus-Naur Form) grammars

- Originally used to define programming languages
- Variables denoted by long names in angle brackets, e.g.
 - <identifier>, <if-then-else-statement>,
<assignment-statement>, <condition>
 - ::= used instead of \rightarrow

BNF for C

```
statement:
  ((identifier | "case" constant-expression | "default") ":")*
  (expression? ";" |
  block |
  "if" "(" expression ")" statement |
  "if" "(" expression ")" statement "else" statement |
  "switch" "(" expression ")" statement |
  "while" "(" expression ")" statement |
  "do" statement "while" "(" expression ")" ";" |
  "for" "(" expression? ";" expression? ";" expression? ")" statement |
  "goto" identifier ";" |
  "continue" ";" |
  "break" ";" |
  "return" expression? ";"
  )

block: "{" declaration* statement* "}"

expression:
  assignment-expression%

assignment-expression: (
  unary-expression (
    "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | "&=" |
    "^=" | "|="
  )
  )* conditional-expression

conditional-expression:
  logical-OR-expression ( "?" expression ":" conditional-expression )?
```

Parse Trees

Back to middle school:

<sentence> ::= <noun phrase> <verb phrase>

<noun phrase> ::= <article> <adjective> <noun>

<verb phrase> ::= <verb> <adverb> | <verb> <object>

<object> ::= <noun phrase>

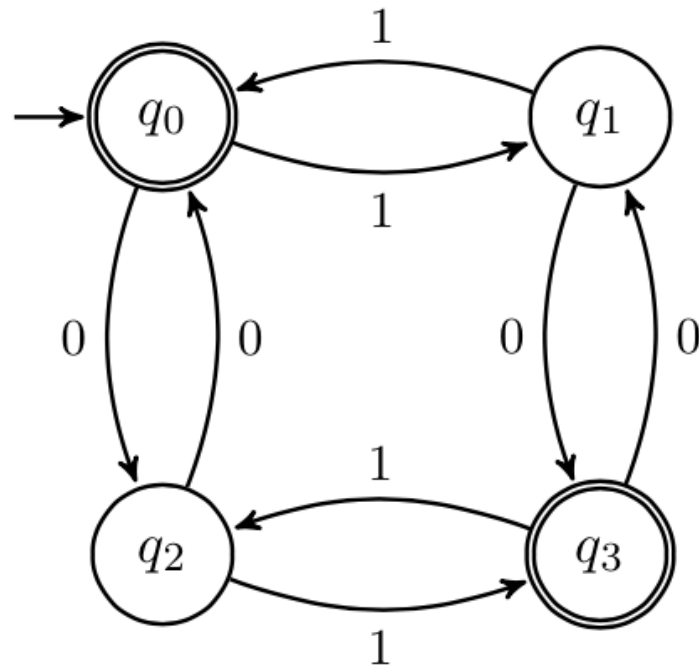
Parse:

The yellow duck squeaked loudly

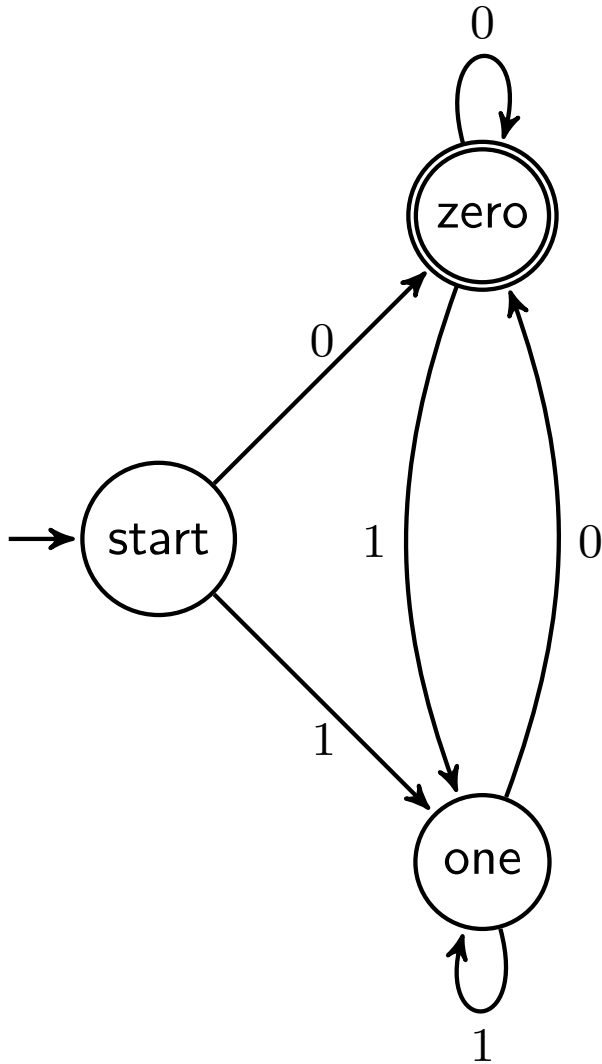
The red truck hit a parked car

CSE 311: Foundations of Computing

Lecture 20: Finite State Machines (DFAs)



A Weird Sort of Programming!



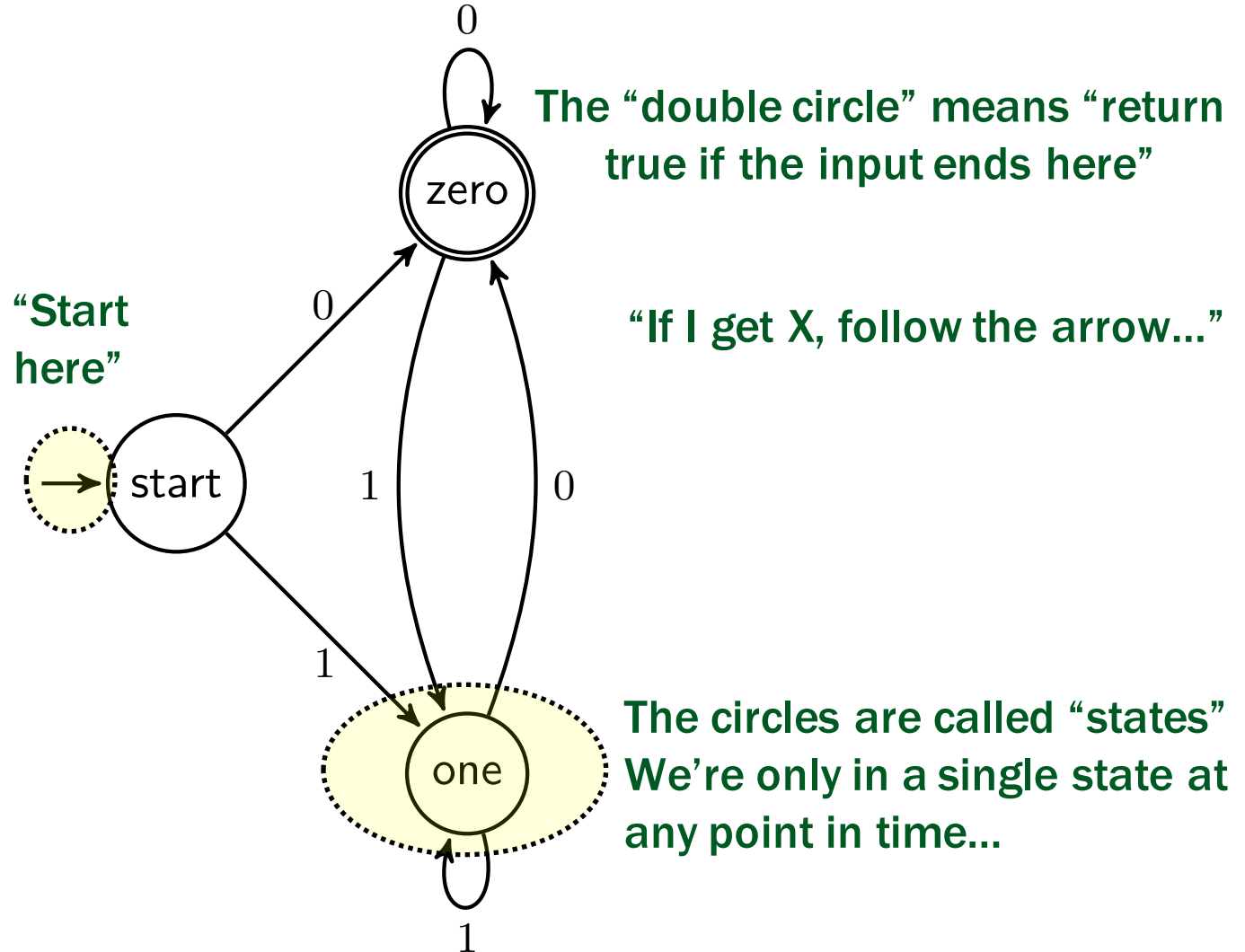
What does this “thing” do?

Take a guess!

If you had to give this “method” a name, what would it be?

`boolean isEven(binary s)`

Finite State Machines (“DFAs”)



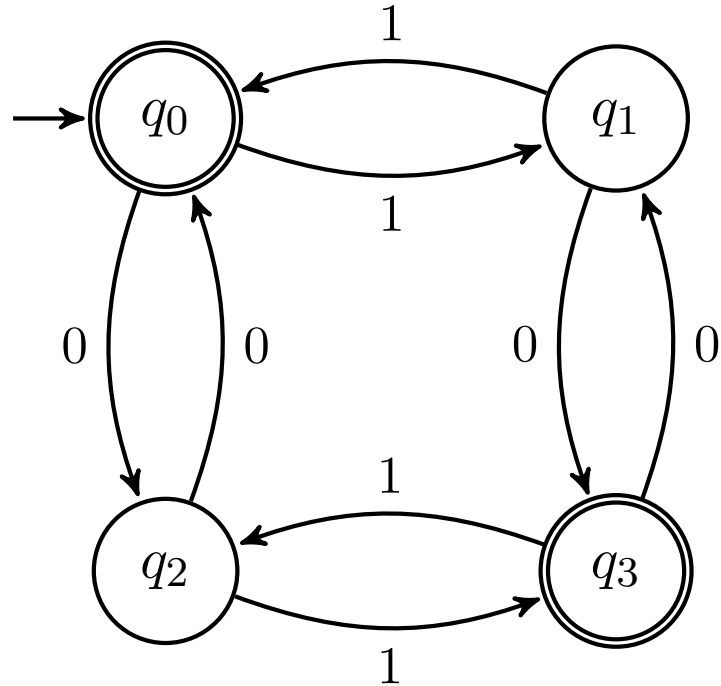
Applications of FSMs (a.k.a. finite automata)

- Implementation of regular expression matching in programs like `grep`
- Control structures for sequential logic in digital circuits
- Algorithms for communication and cache-coherence protocols
 - Each agent runs its own FSM
- Design specifications for reactive systems
 - Components are communicating FSMs

Applications of FSMs (a.k.a. finite automata)

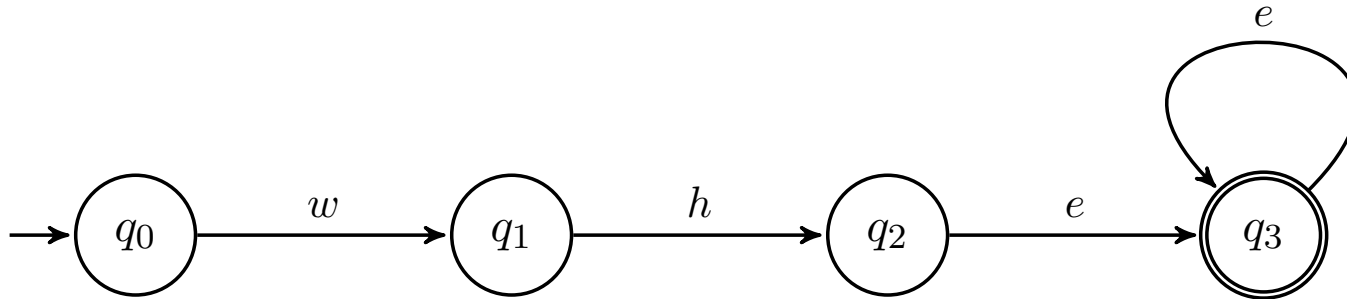
- **Formal verification of systems**
 - Is an unsafe state reachable?
- **Computer games**
 - FSMs provide worlds to explore
- **Minimization algorithms for FSMs can be extended to more general models used in**
 - Text prediction
 - Speech recognition

What language does this machine recognize?



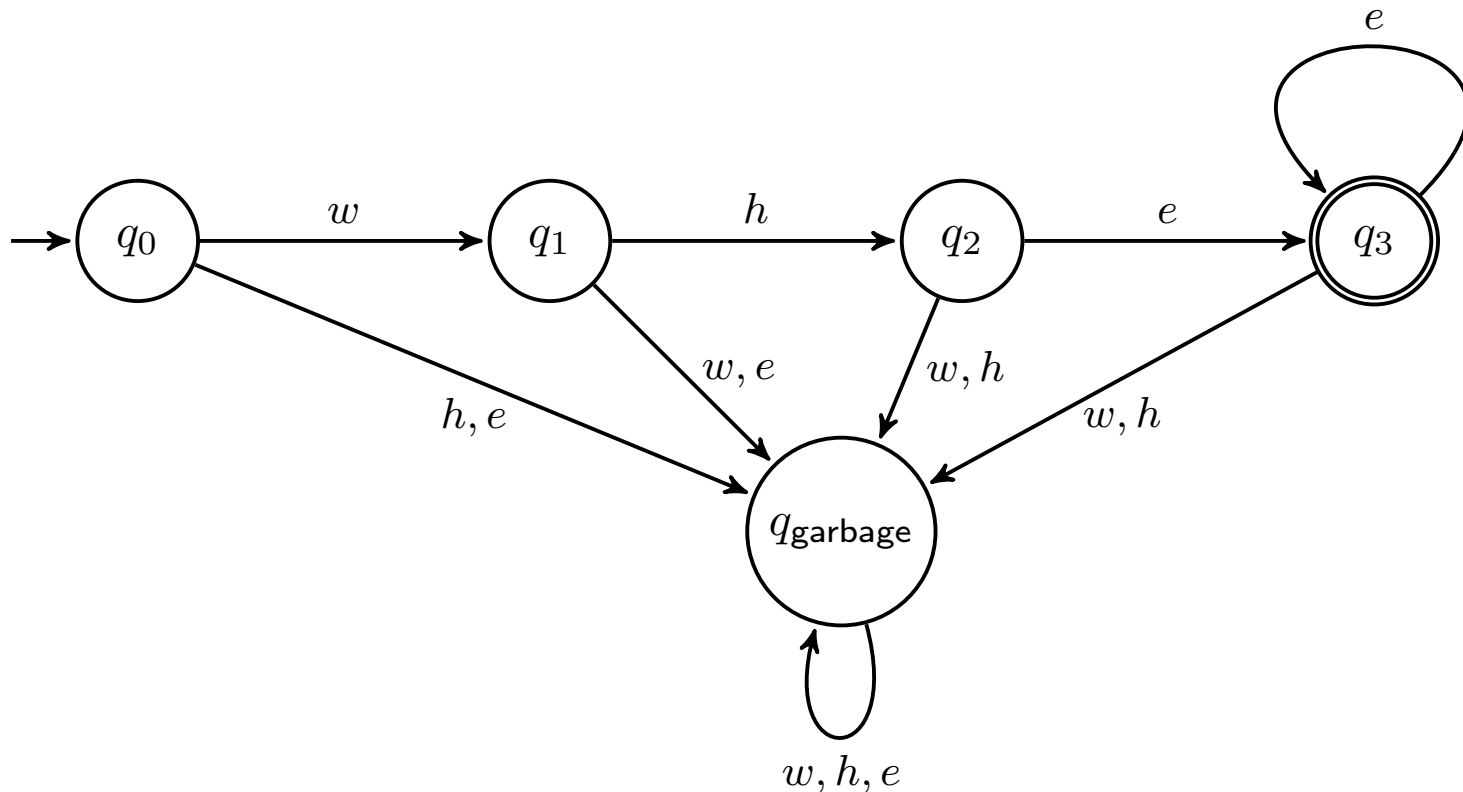
All binary strings with even length

Why is this not a DFA? Fix it!



DFAs must have a transition for every character at every state!

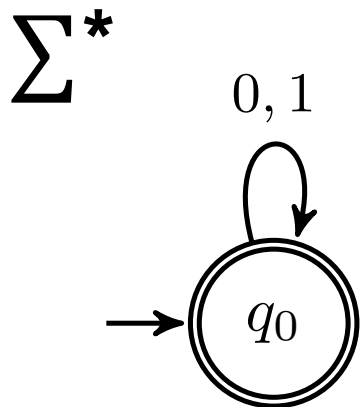
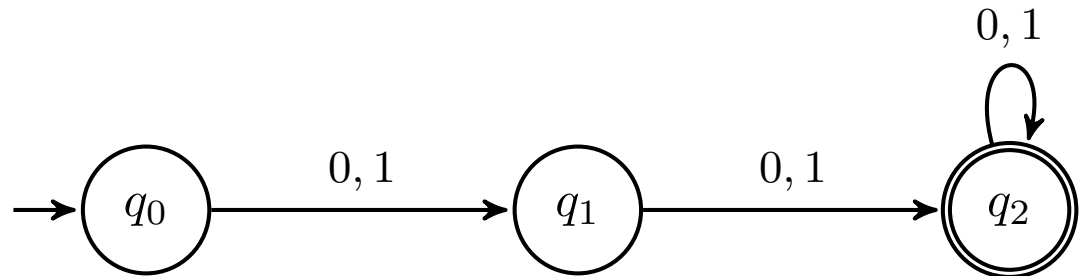
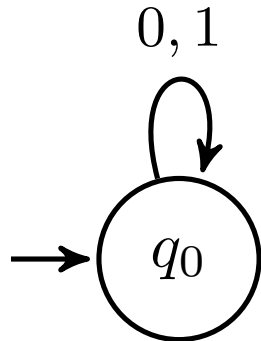
Why is this not a DFA? Fix it!



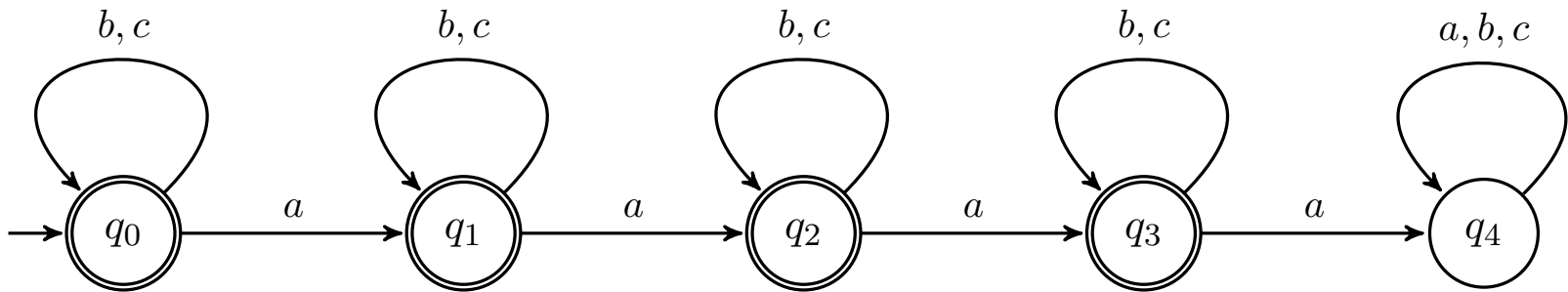
“Garbage states” are a useful concept. Whenever we KNOW we can’t accept the string, just send it to a state that will always go back to itself. This is the way of saying “return false” in DFA-land.

For each of the following languages, create a DFA

\emptyset $\{x \in \{0,1\}^* : \text{len}(x) > 1\}$

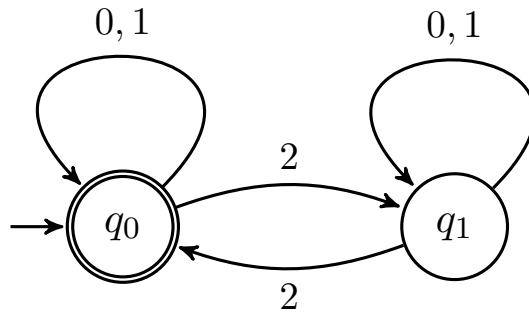


FSM that accepts strings of a's, b's, c's with no more than 3 a's



Strings over $\{0, 1, 2\}^*$

M_1 : Strings with an even number of 2's



M_2 : Strings where the sum of digits mod 3 is 0

