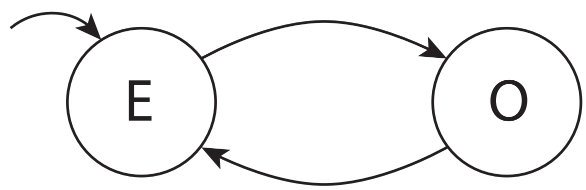
# CS 163 Discrete Structures: Finite State Machines and Regular Expressions

March 21, 2022

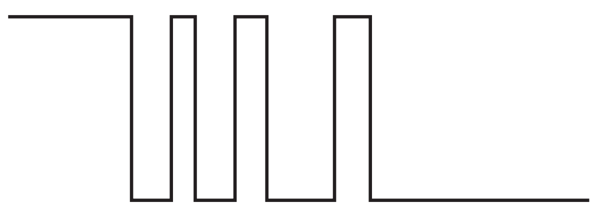
## 1 Introduction

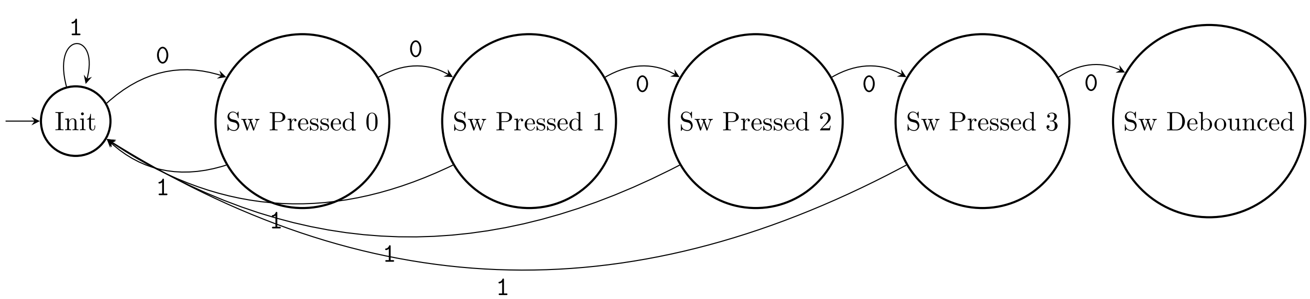



## 2 Finite Automata

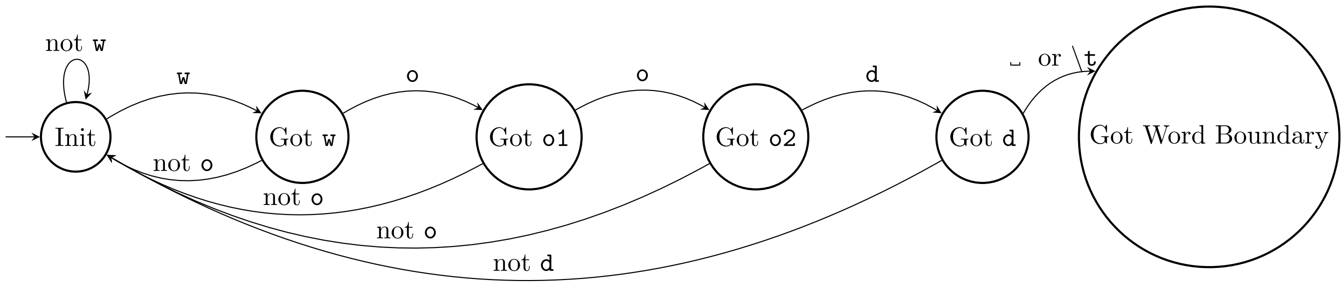
| | |
|---|---|
| **Input Symbols**, $\Sigma$ | Set of inputs that the machine recognizes |
| **Output Symbols** | Set of outputs that can be produced by the machine |
| **Accepting State** | |
| **Input String** | Sequence of input symbols |

**Example** Build an FSM to debounce a switch.






**Example** Make an FSM to detect the word "wood."




| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| h | o | w | ␣ | m | u | c | h | ␣ | w | o | o | d | ␣ | c | a | n | ␣ | a | ␣ | w | o | |
| | | | | | | | | | | | | | | | | | | | | | | |
| o | d | c | h | u | c | k | ␣ | c | h | u | c | k | ␣ | i | f | ␣ | a | ␣ | w | o | o | |
| | | | | | | | | | | | | | | | | | | | | | | |
| d | c | h | u | c | k | ␣ | c | o | u | l | d | ␣ | c | h | u | c | k | ␣ | w | o | o | d |
| | | | | | | | | | | | | | | | | | | | | | | |

**Example** Make an FSM to determine if the number of a's in an input string is even or odd.

$\Sigma = \{a, b\}$ Set of inputs that the machine recognizes
$O = \{E, O\}$ Set of output symbols

**Language** The set of all input strings that are accepted by a state machine.

# 3 Regular Expressions

## 3.1 Notation

- $\Sigma$ Alphabet of possible input symbols
- * Means all possible sequences of elements in a set.
  **Example:** $\{0 \cup 1\}^* = \{\}, \{0\}, \{1\}, \{0,0\}, \{0,1\}, \{1,0\}, \{1,1\}, ...$
- $+$ Shorthand for $RR^*$
  **Example:** $(a \cup b)^+ = (a \cup b) \circ (a \cup b)^*$
  Strings of $a$ and/or $b$ of at least length 1
- $\circ$ Concatenation
  **Example:** $a \circ b = ab$
  **Note:** When no other operator is present, concatenation is implied.
- $\cup$ Union.

**Regular Expression** Expressions built with the $\cup$, $\circ$, and $*$ operators. The value of a regular expression is a language.

**Fact** Any valid regular expression can be converted to a finite state machine.

**Examples**

| | |
|---|---|
| $(0 \cup 1)^*$ | The set of all possible strings of 0's and 1's of any length, including empty strings. |
| $\Sigma^*1\Sigma*$ | Strings with at least one 1. |
| $1^*(01^+)^*$ | Strings where every 0 is followed by at least one 1. |
| $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$ | Strings that start and end with the same symbol |
| $0^3$ | Strings with three 0's |

## 3.2 RexEx in `vim`

| | |
|---|---|
| / $\wedge$ \+ | Equivalent to $\wedge$ $^+$, which searches for spaces at the beginning of lines. $\wedge$ refers to beginning of line. |
| /s \{2} | Equivalent to $s^2$, which searches for sequences of two s |
| /baa \+ | Equivalent to $baa^+$, which searches for *ba* followed by an indeterminate number of a's. |
| $[HhPp]$ackers | Finds all occurances of hackers or packers with uppercase or lowercase first letter. |
| :%s/foo/bar/g | Searches for all occorances of foo and changes them to bar |
| :%s/$\wedge$ \+//g | Searches for spaces at the beginning of lines and replaces them with nothing |

**Example that doesn't work** $0^n1^n$ to match an arbitrary number of 0's followed by exactly the same number of 1's.

# 4 Grammars

A grammar is *a set of rules that defines how to generate valid strings from an alphabet of symbols.* Two types we talk about are **regular** and **context-free**.

**Terminal and Nonterminal Symbols**

**Productions**

## 4.1 Regular Grammars

In a regular grammar, **you can have EITHER terminals OR nonterminals on the right side of a production, but not both.**

## 4.2 Context-Free Grammars

In a context-free grammar, **you can have BOTH terminals AND nonterminals on the right side of a production.**