

# Regular Expressions & Automata

---



**Fawzi Emad**  
**Chau-Wen Tseng**

**Department of Computer Science**  
**University of Maryland, College Park**

## Complexity to Computability

- **Just looked at algorithmic complexity**
  - **How many steps required**
- **At high end of complexity is computability**
  - **Decidable**
  - **Undecidable**

## Complexity to Computability

- Approach complexity from different direction
- Look at simple models of computation
  - Regular expressions
  - Finite automata
  - Turing Machines

## Overview

- Regular expressions
  - Notation
  - Patterns
  - Java support
- Automata
  - Languages
  - Finite State Machines
  - Turing Machines
  - Computability


## Regular Expression (RE)

- Notation for describing **simple** string patterns
- Very useful for text processing
  - Finding / extracting pattern in text
  - Manipulating strings
  - Automatically generating web pages

## Regular Expression

- Regular expression is composed of
  - Symbols
  - Operators
    - Concatenation **AB**
    - Union **A | B**
    - Closure **A\***

## Definitions

- **Alphabet**
    - Set of symbols  $\Sigma$
    - Examples  $\Rightarrow \{a, b\}, \{A, B, C\}, \{a-z, A-Z, 0-9\} \dots$
  - **Strings**
    - Sequences of 0 or more symbols from alphabet
    - Examples  $\Rightarrow \epsilon, "a", "bb", "cat", "caterpillar" \dots$
  - **Languages**
    - Sets of strings
    - Examples  $\Rightarrow \emptyset, \{\epsilon\}, \{"a"\}, \{"bb", "cat"\} \dots$
-  empty string

## More Formally

- Regular expression describes a language over an alphabet
- $L(E)$  is language for regular expression  $E$ 
  - Set of strings generated from regular expression
  - String in language if it matches pattern specified by regular expression

## Regular Expression Construction

- Every symbol is a regular expression
  - Example “a”
- REs can be constructed from other REs using
  - Concatenation
  - Union |
  - Closure \*

## Regular Expression Construction

- Concatenation
  - A followed by B
  - $L(AB) = \{ ab \mid a \in L(A) \text{ AND } b \in L(B) \}$
- Example
  - a
    - {“a”}
  - ab
    - {“ab”}

## Regular Expression Construction

### ■ Union

- A or B
- $L(A | B) = \{ a \mid a \in L(A) \text{ OR } a \in L(B) \}$

### ■ Example

- a | b
  - {"a", "b"}

## Regular Expression Construction

### ■ Closure

- Zero or more A
- $L(A^*) = \{ a \mid a = \epsilon \text{ OR } a \in L(A) \text{ OR } a \in L(A)L(A)\dots \}$

### ■ Example

- $a^*$ 
  - $\{\epsilon, "a", "aa", "aaa", "aaaa" \dots\}$
- $(ab)^*c$ 
  - $\{"c", "abc", "ababc", "abababc" \dots\}$

## Regular Expressions in Java

- Java supports regular expressions
  - In `java.util.regex.*`
  - Applies to `String` class in Java 1.4
- Introduces additional specification methods
  - Simplifies specification
  - Does not increase power of regular expressions
  - Can simulate with concatenation, union, closure

## Regular Expressions in Java

- Concatenation
  - `ab`            `"ab"`
  - `(ab)c`          `"abc"`
- Union ( bar | or square brackets [ ] )
  - `a | b`            `"a", "b"`
  - `[abc]`            `"a", "b", "c"`
- Closure (star \*)
  - `(ab)*`             $\epsilon$ , `"ab"`, `"abab"`, `"ababab"` ...
  - `[ab]*`             $\epsilon$ , `"a"`, `"b"`, `"aa"`, `"ab"`, `"ba"`, `"bb"` ...

## Regular Expressions in Java

- **One or more (plus +)**
  - (a)+            One or more “a”s
- **Range (dash –)**
  - [a–z]            Any lowercase letters
  - [0–9]            Any digit
- **Complement (caret ^ at beginning of RE)**
  - [^a]             Any symbol except “a”
  - [^a–z]           Any symbol except lowercase letters

## Regular Expressions in Java

- **Precedence**
  - Higher precedence operators take effect first
- **Precedence order**
  - Parentheses        ( ... )
  - Closure            a\* b+
  - Concatenation     ab
  - Union              a | b
  - Range              [ ... ]

## Regular Expressions in Java

### ■ Examples

- `ab+`            “ab”, “abb”, “abbb”, “abbbb”...
- `(ab)+`           “ab”, “abab”, “ababab”, ...
- `ab | cd`          “ab”, “cd”
- `a(b | c)d`        “abd”, “acd”
- `[abc]d`           “ad”, “bd”, “cd”

### ■ When in doubt, use parentheses

## Regular Expressions in Java

### ■ Predefined character classes

- `[.]`            Any character except end of line
- `[\d]`           Digit: [0-9]
- `[\D]`           Non-digit: [^0-9]
- `[\s]`           Whitespace character: [ \t\n\r0B\f]
- `[\S]`           Non-whitespace character: [^\s]
- `[\w]`           Word character: [a-zA-Z\_0-9]
- `[\W]`           Non-word character: [^\w]

## Regular Expressions in Java

- **Literals using backslash \**
  - Need two backslash
  - Java compiler will interpret 1<sup>st</sup> backslash for String
- **Examples**
  - `\\]`      “]”
  - `\\.`       “.”
  - `\\\\`      “\”
    - 4 backslashes interpreted as `\\` by Java compiler

## Using Regular Expressions in Java

1. **Compile pattern**
  - `import java.util.regex.*;`
  - `Pattern p = Pattern.compile("[a-z]+");`
2. **Create matcher for specific piece of text**
  - `Matcher m = p.matcher("Now is the time");`
3. **Search text**
  - `Boolean found = m.find();`
    - Returns true if pattern is found in text

## Using Regular Expressions in Java

- If pattern is found in text
  - `m.group()` ⇒ string found
  - `m.start()` ⇒ index of the first character matched
  - `m.end()` ⇒ index after last character matched
  - `m.group()` is same as `substring(m.start(), m.end())`
- Calling `m.find()` again
  - Starts search after end of current pattern match
  - If no more matches, return to beginning of string

## Complete Java Example

### ■ Code

```
import java.util.regex.*;
public class RegexTest {
    public static void main(String args[]) {
        Pattern p = Pattern.compile("[a-z]+");
        Matcher m = p.matcher("Now is the time");
        while (m.find()) {
            System.out.print(m.group() + " - ");
        }
    }
}
```

### ■ Output

- `ow - is - the - time -`

## Language Recognition



- Accept string if and only if in language
- Abstract representation of **computation**
- Performing language recognition can be
  - Simple
    - Strings with **even** number of 1's
  - Hard
    - Strings representing **legal** Java programs
  - Impossible!
    - Strings representing **nonterminating** Java programs

## Automata

- Simple abstract computers
- Can be used to recognize languages
- Finite state machine
  - States + transitions
- Turing machine
  - States + transitions + tape

## Finite State Machine

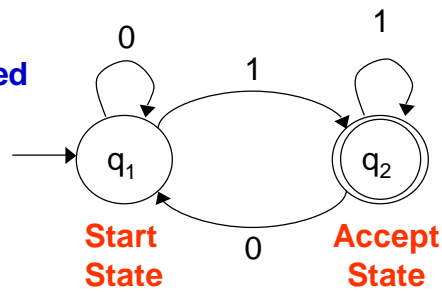
### States

- Starting 
- Accepting 
- Finite number allowed

### Transitions

- State to state
- Labeled by symbol

  $a$

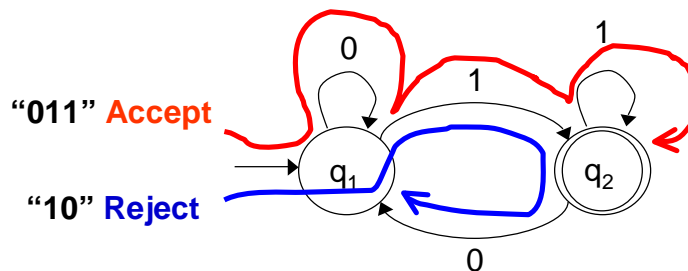


$$L(M) = \{ w \mid w \text{ ends in a } 1 \}$$

## Finite State Machine

### Operations

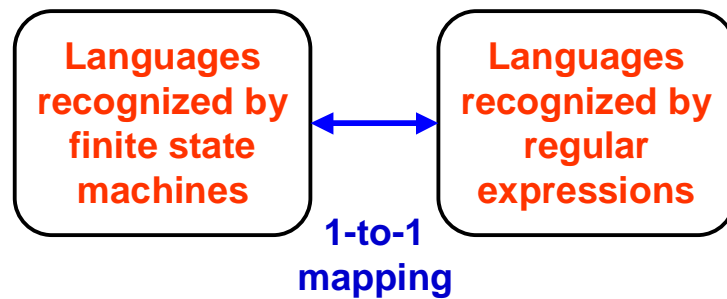
- Move along transitions based on symbol
- **Accept** string if ends up in accept state
- **Reject** string if ends up in non-accepting state



# Finite State Machine

## ■ Properties

- Powerful enough to recognize regular expressions
- In fact, finite state machine  $\Leftrightarrow$  regular expression



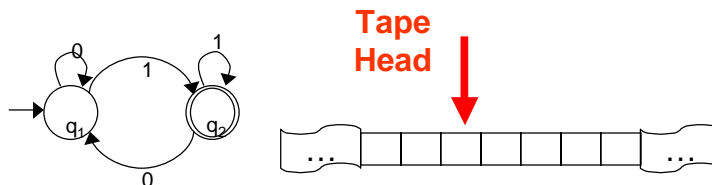
# Turing Machine

## ■ Defined by Alan Turing in 1936

## ■ Finite state machine + **tape**

## ■ Tape

- Infinite storage
- Read / write one symbol at tape head
- Move tape head one space left / right

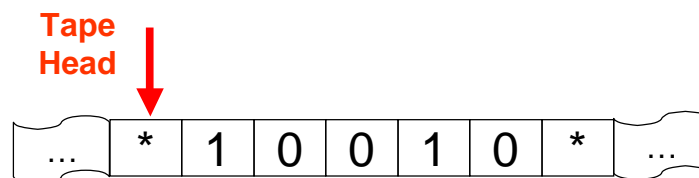


# Turing Machine

## ■ Allowable actions

- Read symbol from current square
- Write symbol to current square
- Move tape head left
- Move tape head right
- Go to next state

# Turing Machine



Current State	Current Content	Value to Write	Direction to Move	New state to enter
START	*	*	Left	MOVING
MOVING	1	0	Left	MOVING
MOVING	0	1	Left	MOVING
MOVING	*	*	No move	HALT

# Turing Machine

- **Operations**
  - Read symbol on current square
  - Select action based on symbol & current state
  - **Accept** string if in accept state
  - **Reject** string if halts in non-accepting state
  - **Reject** string if computation **does not terminate**
- **Halting problem**
  - It is **undecidable** in general whether long-running computations will eventually accept

# Computability

- **Computability**
  - A language is **computable** if it can be recognized by some algorithm with finite number of steps
- **Church-Turing thesis**
  - Turing machine can recognize any language computable on any machine
- **Intuition**
  - Turing machine captures essence of computing
    - Program (finite state machine) + Memory (tape)