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: \( A \mid B \)
  - Closure: \( A^* \)
Definitions

Alphabet
- Set of symbols $\Sigma$
- Examples $\Rightarrow \{a, b\}, \{A, B, C\}, \{a-z,A-Z,0-9\}$…

Strings
- Sequences of 0 or more symbols from alphabet
- Examples $\Rightarrow \varepsilon$, “a”, “bb”, “cat”, “caterpillar”…

Languages
- Sets of strings
- Examples $\Rightarrow \emptyset, \{\varepsilon\}, \{“a”\}, \{“bb”, “cat”\}$…
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
- \( L(AB) = \{ \text{ab} \mid a \in L(A) \text{ AND } b \in L(B) \} \)

Example

- a
  - \{“a”\}
- ab
  - \{“ab”\}
Regular Expression Construction

**Union**

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

**Example**

- a \mid b
- \{“a”, “b”\}
Regular Expression Construction

**Closure**

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

**Example**

- \( a^* \)
  - \( \{ \varepsilon, "a", "aa", "aaa", "aaaa" \ldots \} \)
- \( (ab)^*c \)
  - \( \{ "c", "abc", "ababc", "abababc"\ldots \} \)
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 [] for chars)**
  - `a | b"a", "b"
  - `[abc]"a", "b", "c"

- **Closure (star *)**
  - `(ab)* ε, “ab”, “abab”, “ababab” …
  - `[ab]* ε, “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**

1. Parentheses: `( ... )`
   - Highest
2. Closure: `a* b+`
3. Concatenation: `ab`
4. Union: `a | b`
5. Range: `[ ... ]`
   - Lowest
Regular Expressions in Java

Examples

- ab+  
  "ab", "abb", "abbb", "abbbbb"...

- (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\x0B\f\r]
- `[\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 1st backslash for String

- Examples
  - \] "]"
  - \. ".
  - \\
  - "\"

  - 4 backslashes interpreted as \ by Java compiler
Using Regular Expressions in Java

Compile pattern

- import java.util.regex.*;
- Pattern p = Pattern.compile("[a-z]+");

Create matcher for specific piece of text

- Matcher m = p.matcher("Now is the time");

Search text

- boolean found = m.find();
  - Returns true if pattern is found anywhere in text
- boolean exact = m.matches();
  - Returns true if pattern matches entire 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 `s.substring(m.start(), m.end())`

Calling `m.find()` again

- Starts search after end of current pattern match
Complete Java Example

Code

```java
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

```plaintext
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
  - Not Simple
    - Strings with any number of a’s, followed by the same number of b’s
  - Hard
    - Strings representing legal Java programs
  - Not possible for all cases
    - 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

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

“011” Accept

“10” Reject
Finite State Machine

Properties

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

Languages recognized by finite state machines $\leftrightarrow$ Languages recognized by regular expressions

1-to-1 mapping
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

#### Table of States and Operations

<table>
<thead>
<tr>
<th>Current State</th>
<th>Current Content</th>
<th>Value to Write</th>
<th>Direction to Move</th>
<th>New state to enter</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>*</td>
<td>*</td>
<td>Left</td>
<td>MOVING</td>
</tr>
<tr>
<td>MOVING</td>
<td>1</td>
<td>0</td>
<td>Left</td>
<td>MOVING</td>
</tr>
<tr>
<td>MOVING</td>
<td>0</td>
<td>1</td>
<td>Left</td>
<td>MOVING</td>
</tr>
<tr>
<td>MOVING</td>
<td>*</td>
<td>*</td>
<td>No move</td>
<td>HALT</td>
</tr>
</tbody>
</table>

#### Diagram

- **Tape Head**: Indicates the position of the tape head.
- **Current State**: The current state of the machine.
- **Current Content**: The content of the tape at the current position.
- **Value to Write**: The value to write on the tape.
- **Direction to Move**: The direction to move the tape head.
- **New state to enter**: The new state to enter after the operation.
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

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

Both in a formal sense, and in an informal practical sense
Computability – Turing Test

Turing Test
- Test machine’s capability to demonstrate intelligence
- Proposed by Alan Turing in 1950
- Practical test for “Can machines think?”

Test procedure
1. Judge converses via text (e.g., instant messaging)
2. Pass if can’t reliably tell whether machine or human
More Turing Tests

“On the Internet, nobody knows you’re a dog.”
Computability – Reverse Turing Test

Reverse Turing test
- Computer determines whether human or machine

CAPTCHA
- Completely Automated Public Turing test to tell Computers and Humans Apart
- Challenge-response asking for word identification
  - Useful for foiling scripts
More Reverse Turing Tests