Regular Expressions

What is a Regular Expression?

- An **arithmetic expression** like $2 * (8 + 9)$ has:
  - Numbers as *operands*
  - *Operators* like `*`, `+`, etc.
- A **regular expression (RE)** has:
  - Alphabet symbols as operands
  - Regular language operators as operators

Operator Symbols: Set vs. RE

Concept	Set Notation	RE Notation
Union	$\cup$	`|`
Concatenation	Implicit (or `.`)	Implicit (or `.`)
Kleene Closure	`*`	`*`

Operator Precedence in REs

From **highest to lowest** precedence:

1. **Closure** (`*`) - evaluated first
2. **Concatenation** (implicit or `.`)
3. **Union** (`|`) - evaluated last

💡

Use **parentheses** to override precedence!

Examples: REs over {0, 1}

Regular Language	RE
{0}	`0`
{1}	`1`
{0, 1}	`0|1`
{00, 01, 10, 11}	`(0|1).(0|1)`
{ε, 0, 1, 00, 01, 10, 11, ...}	`(0|1)*`
Binary strings with no leading 0s	`0|(1.(0|1)*)`

Regular Languages: Formal Definition

> A formal language is called a **regular language** if some DFA or NFA recognizes it **or** it is specified by an RE.

Kleene's Theorem (1951)

> **REs, DFAs, and NFAs are equivalent models** to characterize the regular languages

This means:

- Any language described by an RE can be recognized by a DFA/NFA
- Any language recognized by a DFA/NFA can be described by an RE

Properties of Regular Languages

1. **Simple Specification**
    - Using a regular expression
2. **Automated Recognition**
    - Using an NFA or DFA
3. **Practical Applications**
    - Input validation
    - Pattern searching (e.g., `grep`)
    - Syntax specification

Active Learning: Match the RE to the Language

For the alphabet {a, b}, which RE matches each language?

1. **Language:** Strings starting with 'a'
2. **Language:** Strings containing only 'a's and 'b's (any number)
3. **Language:** Exactly one 'a' followed by any number of 'b's

Options:

1. `(a|b)*`
2. `a(a|b)*`
3. `ab*`

Generalized Regular Expressions

In practice, REs often include enhancements:

- **Escape mechanisms** for operator symbols in the alphabet
- **Shorthand notations:**
  - `.` for any alphabet symbol
  - Character classes like `[a-z]`
  - `^` for line start, `$` for line end
  - Negated ranges like `[^0-9]`

Enhanced Closure Operators

Common extensions to minimal REs:

Notation	Meaning
`+`	One or more
`?`	Zero or one
`{n}`	Exactly n occurrences
`{m,n}`	Between m and n occurrences

⚠️

For this class: We use minimal REs, but be aware of these in practice!

Grep: Practical Application

`grep` - Unix command-line utility for searching text

```
grep -E "pattern" filename
```

Uses regular expressions to:

- Search plain-text files
- Match lines against patterns
- Filter and extract data

💡

See shell commands notes for more details on `grep` usage

Key Takeaways

✓ Regular expressions compactly specify regular languages

✓ REs use operators: union (`|`), concatenation, closure (`*`)

✓ Precedence: closure > concatenation > union

✓ Kleene's Theorem: REs ≡ DFAs ≡ NFAs

✓ Practical uses: validation, pattern matching, syntax

✓ Generalized REs extend the minimal model with useful features