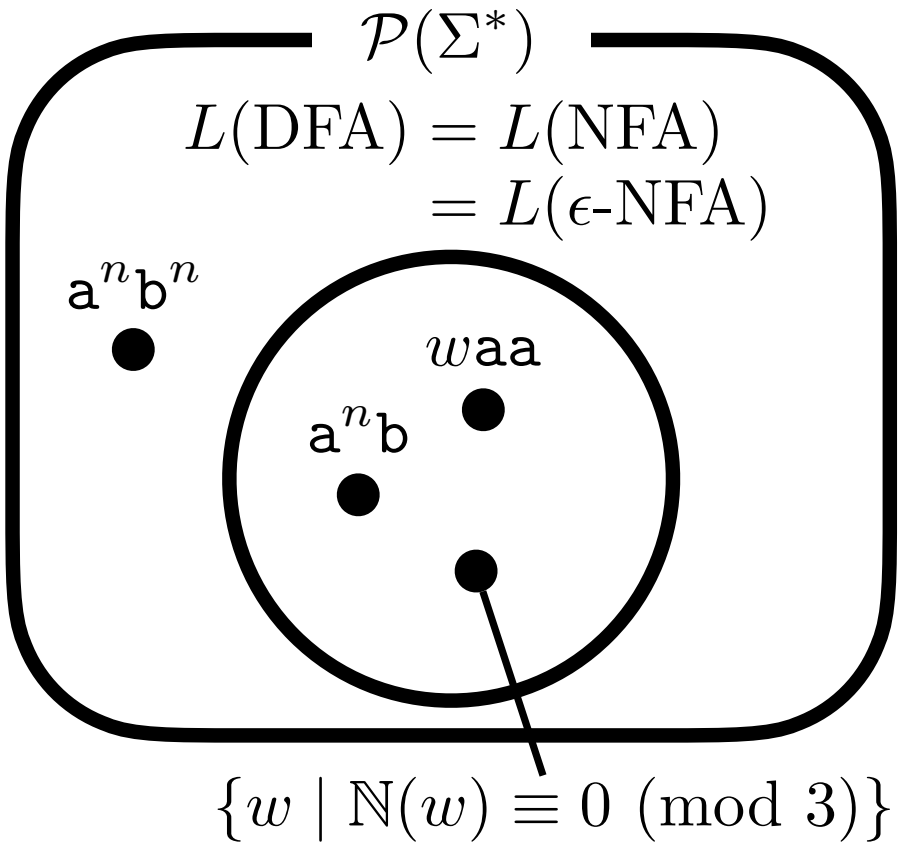
Lecture 6 – Regular Expressions and Languages

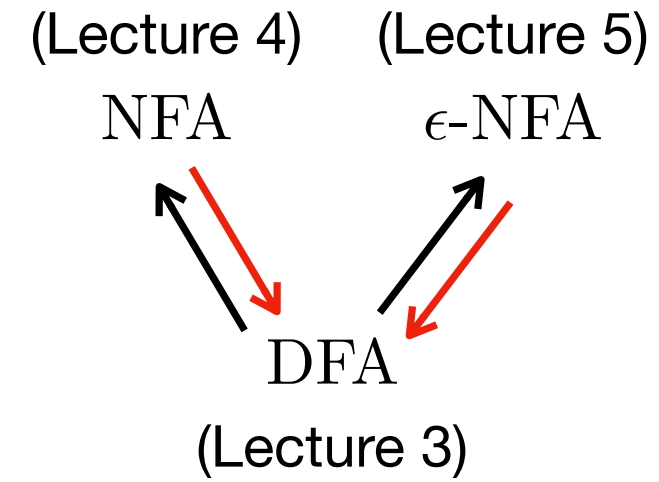
COSE215: Theory of Computation

Jihyeok Park

2025 Spring

Recall





→ : Subset Construction

Contents

Contents

Recall: Operations in Languages

We already learned the following **operations** on languages:

- **Union** of languages: $L_1 \cup L_2$
- **Concatenation** of languages: $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$
- **Kleene star** of a language: $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{n \geq 0} L^n$

For example, consider the following languages over symbols $\Sigma = \{\mathtt{a}, \mathtt{b}\}$:

$$L_1 = \{\mathtt{a}^n \mid n \geq 1\} \qquad L_2 = \{\mathtt{b}^n \mid n \geq 1\}$$

$$\begin{aligned} L_1 \cup L_2 &= \{\mathtt{a}^n \text{ or } \mathtt{b}^n \mid n \geq 1\} \\ L_1 L_2 &= \{\mathtt{a}^n \mathtt{b}^m \mid n, m \geq 1\} \neq \{\mathtt{a}^n \mathtt{b}^n \mid n \geq 1\} \\ L_1^* &= \{\mathtt{a}^n \mid n \geq 0\} \neq \{\mathtt{a}^n \mid n \geq 1\} \end{aligned}$$

Regular expressions (REs) provide a new way to define languages with above **operations** without using finite automata!

Definition of Regular Expressions

PLRG

Definition (Regular Expressions)

A **regular expression** over a set of symbols Σ is inductively defined as:

- **(Basis Case)** $\varnothing$, ϵ, and $a \in \Sigma$ are regular expressions.
- **(Induction Case)** If R_1 and R_2 are regular expressions, then so are $R_1 \mid R_2$, $R_1 R_2$, R^*, and (R).

The following is the **syntax** of regular expressions and examples:

$$
\begin{array}{rcll}
R & ::= & \varnothing & \text{(Empty)} \\
 & \mid & \epsilon & \text{(Epsilon)} \\
 & \mid & a & \text{(Symbol)} \\
 & \mid & R \mid R & \text{(Union)} \\
 & \mid & R\ R & \text{(Concatenation)} \\
 & \mid & R^* & \text{(Kleene Star)} \\
 & \mid & (R) & \text{(Parentheses)}
\end{array}
$$

$\varnothing$	ϵ	a	a|b	ab
a^*	$a(\varnothing \mid c)^*$	$(a\epsilon) \mid b^*$	$(a(bc^*b)^*)^*$	$(a\varnothing a) \mid b^*$

Precedence Order

Arithmetic expressions have the following **precedence order**:

$$\times \quad > \quad +$$

It means that multiplication ($\times$) has higher precedence than addition ($+$).
For example,

$$1 + 2 \times 3 \qquad \text{means} \qquad 1 + (2 \times 3)$$

Similarly, **regular expressions** have the following **precedence order**:

$$* \quad > \quad \cdot \quad > \quad |$$

For example,

$$\mathtt{a}|\epsilon\mathtt{b}^* \qquad \text{means} \qquad \mathtt{a}|(\epsilon(\mathtt{b}^*))$$

$$(\mathtt{a}|\epsilon)\mathtt{b}^* \qquad \text{means} \qquad (\mathtt{a}|\epsilon)(\mathtt{b}^*)$$

Definition of Regular Expressions

PLRG

```
// The definition of regular expressions
enum RE:
  case Emp                                  // ∅
  case Eps                                  // ϵ
  case Sym(symbol: Symbol)                  // a
  case Union(left: RE, right: RE)           // R1 | R2
  case Concat(left: RE, right: RE)          // R1 R2
  case Star(re: RE)                         // R*
```

In the algebraic data type (ADT) of regular expressions, we do **not need to explicitly define the parentheses** because it is already handled by the structure of the ADT.

```
// import all constructors (Emp, Eps, Sym, Union, Concat, Star) of RE
import RE.*

// a | ϵ b*
val re1: RE = Union(Sym('a'), Concat(Eps, Star(Sym('b'))))

// (a | ϵ) b*
val re2: RE = Concat(Union(Sym('a'), Eps), Star(Sym('b')))
```

Language of Regular Expressions

PLRG

Definition (Language of Regular Expressions)

For a given regular expression R on a set of symbols Σ, the **language** $L(R)$ of R is inductively defined as follows:

$$
\begin{array}{lcl}
L(\varnothing) & = & \varnothing \\
L(\epsilon) & = & \{\epsilon\} \\
L(a) & = & \{a\}
\end{array}
\qquad
\begin{array}{lcl}
L(R_1 \mid R_2) & = & L(R_1) \cup L(R_2) \\
L(R_1 R_2) & = & L(R_1) L(R_2) \\
L(R^*) & = & L(R)^* \\
L((R)) & = & L(R)
\end{array}
$$

$$
\begin{array}{lcl}
L(\mathtt{a} \mid \epsilon \mathtt{b}^*) & = & L(\mathtt{a}) \cup L(\epsilon \mathtt{b}^*) \\
& = & \{\mathtt{a}\} \cup L(\epsilon) L(\mathtt{b}^*) \\
& = & \{\mathtt{a}\} \cup \{\epsilon\} L(\mathtt{b})^* \\
& = & \{\mathtt{a}\} \cup \{\mathtt{b}\}^*
\end{array}
\qquad
\begin{array}{lcl}
& = & \{\mathtt{a}\} \cup L(\epsilon) L(\mathtt{b}^*) \\
& = & \{\mathtt{a}\} \cup \{\epsilon\} \{\mathtt{b}\}^* \\
& = & \{\mathtt{a} \text{ or } \mathtt{b}^n \mid n \geq 0\}
\end{array}
$$

$$
\begin{array}{lcl}
L((\mathtt{a} \mid \epsilon)\mathtt{b}^*) & = & L((\mathtt{a} \mid \epsilon)) L(\mathtt{b}^*) \\
& = & (L(\mathtt{a}) \cup L(\epsilon)) L(\mathtt{b})^* \\
& = & \{\mathtt{a}\mathtt{b}^n \text{ or } \mathtt{b}^n \mid n \geq 0\}
\end{array}
\qquad
\begin{array}{lcl}
& = & L(\mathtt{a} \mid \epsilon) L(\mathtt{b})^* \\
& = & (\{\mathtt{a}\} \cup \{\epsilon\}) \{\mathtt{b}\}^*
\end{array}
$$

Extended Regular Expressions

PLRG

More operators can be added to regular expressions:

$$\begin{array}{rcll} R & ::= & \cdots & \\ & | & R^+ & \text{(Kleene plus)} \\ & | & R^? & \text{(Optional)} \end{array}$$

(Note that $^+$ and $^?$ have same precedence as *.)

Actually, they are just **syntactic sugar** for the existing operators:

$$\begin{array}{rcccl} L(R^+) & = & L(RR^*) & = & L(R^*R) \\ L(R^?) & = & L(R \mid \epsilon) & = & L(\epsilon \mid R) \end{array}$$

For examples,

$$\begin{array}{rcl} L((\mathtt{ab})^+) & = & L(\mathtt{ab}(\mathtt{ab})^*) = \{(\mathtt{ab})^n \mid n \geq 1\} \\ L(\mathtt{a}^? b) & = & L((\mathtt{a} \mid \epsilon)\mathtt{b}) = \{\mathtt{ab}, \mathtt{b}\} \end{array}$$

Examples

- $L = \{\epsilon, \mathtt{a}, \mathtt{b}\}$

  $\epsilon \mid \mathtt{a} \mid \mathtt{b}$ or $(\mathtt{a} \mid \mathtt{b})^?$

- $L = \{w \in \{0,1\}^* \mid w \text{ contains exactly two } 0's\}$

  $1^*01^*01^*$

- $L = \{w \in \{0,1\}^* \mid w \text{ contains at least two } 0's\}$

  $(0 \mid 1)^*0(0 \mid 1)^*0(0 \mid 1)^*$

- $L = \{w \in \{0,1\}^* \mid w \text{ has three consecutive } 0's\}$

  $(0 \mid 1)^*000(0 \mid 1)^*$

- $L = \{w \in \{\mathtt{a},\mathtt{b}\}^* \mid \mathtt{a} \text{ and } \mathtt{b} \text{ alternate in } w\}$

  $\mathtt{a}^?(\mathtt{ba})^*\mathtt{b}^?$

Examples

PLRG

- $L = \{\mathtt{a}^n\mathtt{b}^m \mid n \geq 3 \wedge m \equiv 0 (\text{mod } 2)\}$

$$\mathtt{aaa}^+(\mathtt{bb})^*$$

- $L = \{\mathtt{a}^n\mathtt{b}^m \mid n + m \equiv 0 (\text{mod } 2)\}$

$$(\mathtt{aa})^*(\mathtt{ab})^?(\mathtt{bb})^*$$

- $L = \{w \in \{0,1\}^* \mid$ the number of 0's is divisible by 3$\}$

$$1^*(01^*01^*01^*)^*$$

- $L = \{w \in \{0,1\}^* \mid \mathbb{N}(w) \equiv 0 \pmod 3\}$ where $\mathbb{N}(w)$ is the natural number represented by w in binary

$$(0|1(01^*0)^*1)^*$$

- $L = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$ – IMPOSSIBLE ($\nexists$ RE R. $L(R) = L$)

Equivalence of Regular Expressions

We say two regular expressions R_1 and R_2 are **equivalent** ($R_1 \equiv R_2$) if their languages are the same: $L(R_1) = L(R_2)$.

Regular expressions have following equivalence relations:

- **Associativity** for union and concatenation:

$$R_1 \mid (R_2 \mid R_3) \equiv (R_1 \mid R_2) \mid R_3 \qquad \text{and} \qquad R_1(R_2R_3) \equiv (R_1R_2)R_3$$

- **Commutativity** for union:

$$R_1 \mid R_2 \equiv R_2 \mid R_1$$

- Left and right **distributive laws**:

$$(R_1 \mid R_2)R_3 \equiv R_1R_3 \mid R_2R_3 \qquad \text{and} \qquad R_1(R_2 \mid R_3) \equiv R_1R_2 \mid R_1R_3$$

Equivalence of Regular Expressions

PLRG

- $\varnothing$ and ϵ are **identity** for union and concatenation:

$$R \mid \varnothing \equiv \varnothing \mid R \equiv R \qquad \text{and} \qquad R\epsilon \equiv \epsilon R \equiv R$$

- $\varnothing$ is **annihilator** for concatenation:

$$R\varnothing \equiv \varnothing R \equiv \varnothing$$

- **Idempotent Law** for union:

$$R \mid R \equiv R$$

- Laws involving Kleene star:

$$(R^*)^* \equiv R^* \qquad \text{and} \qquad \varnothing^* \equiv \epsilon \qquad \text{and} \qquad \epsilon^* \equiv \epsilon$$

$$\epsilon \mid R^* \equiv R^* \mid \epsilon \equiv R^* \qquad \text{and} \qquad R \mid R^* \equiv R^* \mid R \equiv R^*$$

Simplifying Regular Expressions

PLRG

We can simplify regular expressions using the equivalence laws.

For example,

$$
\begin{aligned}
((\mathtt{a}\varnothing)^*(\mathtt{b}|\varnothing|\mathtt{b}^*))^* &\equiv (\varnothing^*(\mathtt{b}|\varnothing|\mathtt{b}^*))^* && (\because R\varnothing \equiv \varnothing - \text{Annihilator}) \\
&\equiv (\epsilon(\mathtt{b}|\varnothing|\mathtt{b}^*))^* && (\because \varnothing^* \equiv \epsilon) \\
&\equiv (\mathtt{b}|\varnothing|\mathtt{b}^*)^* && (\because \epsilon R \equiv R - \text{Identity}) \\
&\equiv (\mathtt{b}|\mathtt{b}^*)^* && (\because R|\varnothing \equiv R - \text{Identity}) \\
&\equiv (\mathtt{b}^*)^* && (\because R|R^* \equiv R^*) \\
&\equiv \mathtt{b}^* && (\because (R^*)^* \equiv R^*)
\end{aligned}
$$

Contents

Regular Expressions in Practice

PLRG

Most **programming languages** support **regular expressions**:

- **Scala** – `scala.util.matching.Regex` class
- **Python** – `re` module
- **JavaScript** – `RegExp` object
- **Rust** – `regex` crate
- ...

For example, we can convert a string to a regular expression (`Regex`) object by using the `r` method in Scala:

```
import scala.util.matching.Regex

val re: Regex = "(a|b)c*".r
re.matches("a")      // true
re.matches("b")      // true
re.matches("accc")   // true
re.matches("bccccc") // true
re.matches("ba")     // false
re.matches("cba")    // false
re.matches("aacc")   // false
re.matches("cccccc") // false
```

Regular Expressions in Practice

In practice, regular expressions support more syntactic sugar:

Syntax	Description
`^`	start of the line
`$`	end of the line
`.`	any character
`[]`	any character in the set
`[^ ]`	any character not in the set
`\d`	any digit
`\w`	any alphanumeric character

```
"ci[dait]*".r
```

```
"\\w+$".r
```

```
"\\d+".r
```

For example, above Scala regular expressions find patterns in each string:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut 53 et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation 42 laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate 129 esse cillum dolore eu
fugiat nulla 5323. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.
```

Regular Expressions in Research

PLRG

There are diverse active research topics for regular expressions:

- **Synthesis** – synthesize regular expressions
- **Security** – detect ReDoS (Regular Expression Denial of Service)
- **Testing** – detect bugs in regular expression engines

For example, we can synthesize regular expressions from examples:[1]

Description	
Strings have at most one pair of consecutive 1s	
Examples	
Positive	Negative
0 11 101010 00011000 00100110001	111 110011 0110110 00011001100 011100011110
Answers	
Students	A variety of uncertain answers
ALPHAREGEX	$(1?0)^*1?1?(01?)^*$

[1] [GPCE 2016] M. Lee, S. So, and H. Oh. "Synthesizing regular expressions from examples for introductory automata assignments."

Summary

PLRG

1. Regular Expressions
    - Recall: Operations in Languages
    - Definition
    - Precedence Order
    - Language of Regular Expressions
    - Extended Regular Expressions
    - Examples

2. Regular Expressions in Practice

Next Lecture

- Equivalence of Regular Expressions and Finite Automata

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr