

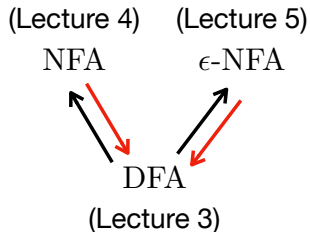
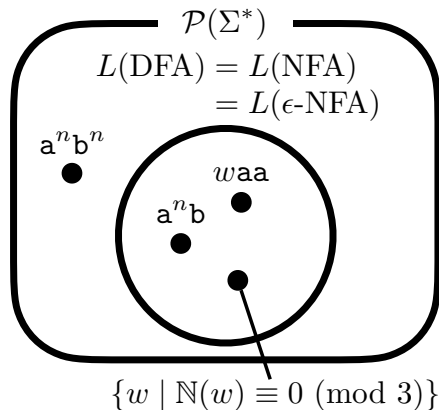
Lecture 6 – Regular Expressions and Languages

COSE215: Theory of Computation

Jihyeok Park



2026 Spring



\rightarrow : Subset Construction

1. Regular Expressions

- Recall: Operations in Languages

- Definition

- Precedence Order

- Language of Regular Expressions

- Extended Regular Expressions

- Examples

2. Regular Expressions in Practice

1. Regular Expressions

- Recall: Operations in Languages

- Definition

- Precedence Order

- Language of Regular Expressions

- Extended Regular Expressions

- Examples

2. Regular Expressions in Practice

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: $L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{n \geq 0} L^n$

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

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

$$L_1 \cup L_2 = \{a^n \text{ or } b^n \mid n \geq 1\}$$

$$L_1 L_2 = \{a^n b^m \mid n, m \geq 1\} \neq \{a^n b^n \mid n \geq 1\}$$

$$L_1^* = \{a^n \mid n \geq 0\} \neq \{a^n \mid n \geq 1\}$$

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

Definition (Regular Expressions)

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

- **(Basis Case)** \emptyset , 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_1R_2 , R^* , and (R) .

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

$R ::= \emptyset$	(Empty)	\mid	$R \mid R$	(Union)
$\mid \epsilon$	(Epsilon)	\mid	$R R$	(Concatenation)
$\mid a$	(Symbol)	\mid	R^*	(Kleene Star)
		\mid	(R)	(Parentheses)

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

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

\times $>$ $+$

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

For example,

$1 + 2 \times 3$ means $1 + (2 \times 3)$

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

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

For example,

$a|\epsilon b^*$ means $a|(\epsilon(b^*))$

$(a|\epsilon)b^*$ means $(a|\epsilon)(b^*)$

```
// The definition of regular expressions
enum RE:
  case Emp                //  $\emptyset$ 
  case Eps                //  $\epsilon$ 
  case Sym(symbol: Symbol) //  $a$ 
  case Union(left: RE, right: RE) //  $R_1 \mid R_2$ 
  case Concat(left: RE, right: RE) //  $R_1 R_2$ 
  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.*

//  $a \mid \epsilon b^*$ 
val re1: RE = Union(Sym('a'), Concat(Eps, Star(Sym('b'))))

//  $(a \mid \epsilon) b^*$ 
val re2: RE = Concat(Union(Sym('a'), Eps), Star(Sym('b')))
```

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:

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

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

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

More operators can be added to regular expressions:

$$R ::= \dots \\ \quad | R^+ \quad (\text{Kleene plus}) \\ \quad | R^? \quad (\text{Optional})$$

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

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

$$L(R^+) = L(RR^*) = L(R^*R)$$

$$L(R^?) = L(R|\epsilon) = L(\epsilon|R)$$

For examples,

$$L((ab)^+) = L(ab(ab)^*) = \{(ab)^n \mid n \geq 1\}$$

$$L(a^?b) = L((a|\epsilon)b) = \{ab, b\}$$

- $L = \{\epsilon, a, b\}$
 $\epsilon | a | b$ or $(a | b)^?$
- $L = \{w \in \{0, 1\}^* \mid w \text{ contains exactly two } 0\text{'s}\}$
 $1^* 0 1^* 0 1^*$
- $L = \{w \in \{0, 1\}^* \mid w \text{ contains at least two } 0\text{'s}\}$
 $(0 | 1)^* 0 (0 | 1)^* 0 (0 | 1)^*$
- $L = \{w \in \{0, 1\}^* \mid w \text{ has three consecutive } 0\text{'s}\}$
 $(0 | 1)^* 000 (0 | 1)^*$
- $L = \{w \in \{a, b\}^* \mid a \text{ and } b \text{ alternate in } w\}$
 $a^? (ba)^* b^?$

- $L = \{a^n b^m \mid n \geq 3 \wedge m \equiv 0 \pmod{2}\}$

$$aaa^+(bb)^*$$

- $L = \{a^n b^m \mid n + m \equiv 0 \pmod{2}\}$

$$(aa)^*(ab)^?(bb)^*$$

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

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

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

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

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

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:

$$R_1 | (R_2 | R_3) \equiv (R_1 | R_2) | R_3 \quad \text{and} \quad R_1(R_2 R_3) \equiv (R_1 R_2) R_3$$

- **Commutativity** for union:

$$R_1 | R_2 \equiv R_2 | R_1$$

- Left and right **distributive laws**:

$$(R_1 | R_2) R_3 \equiv R_1 R_3 | R_2 R_3 \quad \text{and} \quad R_1 (R_2 | R_3) \equiv R_1 R_2 | R_1 R_3$$

- \emptyset and ϵ are **identity** for union and concatenation:

$$R|\emptyset \equiv \emptyset|R \equiv R \quad \text{and} \quad R\epsilon \equiv \epsilon R \equiv R$$

- \emptyset is **annihilator** for concatenation:

$$R\emptyset \equiv \emptyset R \equiv \emptyset$$

- **Idempotent Law** for union:

$$R|R \equiv R$$

- Laws involving Kleene star:

$$(R^*)^* \equiv R^* \quad \text{and} \quad \emptyset^* \equiv \epsilon \quad \text{and} \quad \epsilon^* \equiv \epsilon$$
$$\epsilon|R^* \equiv R^*|\epsilon \equiv R^* \quad \text{and} \quad R|R^* \equiv R^*|R \equiv R^*$$

We can simplify regular expressions using the equivalence laws.

For example,

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

1. Regular Expressions

Recall: Operations in Languages

Definition

Precedence Order

Language of Regular Expressions

Extended Regular Expressions

Examples

2. Regular Expressions in Practice

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("acc")    // true
re.matches("bcccc") // true
re.matches("ba")     // false
re.matches("cba")    // false
re.matches("aacc")   // false
re.matches("cccc")   // false
```

In practice, regular expressions support more syntactic sugar:

Syntax	Description
<code>^</code>	start of the line
<code>\$</code>	end of the line
<code>.</code>	any character
<code>[]</code>	any character in the set
<code>[^]</code>	any character not in the set
<code>\d</code>	any digit
<code>\w</code>	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
incidunt 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.
```

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	111
11	110011
101010	0110110
00011000	00011001100
00100110001	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.”

1. Regular Expressions

- Recall: Operations in Languages

- Definition

- Precedence Order

- Language of Regular Expressions

- Extended Regular Expressions

- Examples

2. Regular Expressions in Practice

- Equivalence of Regular Expressions and Finite Automata

Jihyeok Park

jihyeok_park@korea.ac.kr

<https://plrg.korea.ac.kr>