Formal languages and Automata
形式语言与自动机

Chapter3 REGULAR LANGUAGES AND REGULAR GRAMMARS

Beijing University of Posts and Telecommunications

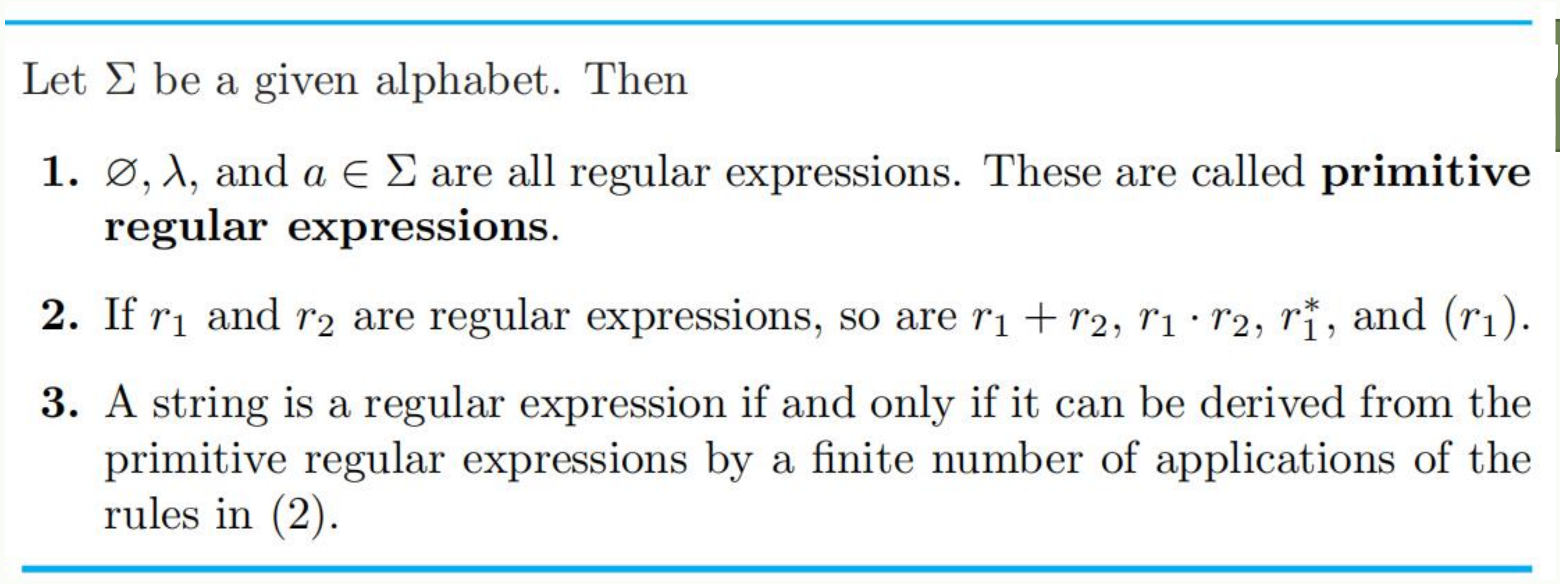
北京邮电大学

刘咏彬 liuyb@bupt.edu.cn

2023.10

Regular Expression

- Regular expression is a concise(简洁的) way to describe regular language.
- This notation involves a combination of strings of symbols
- From some alphabet Σ, parentheses（括号）, and the operators +, ·, and □.
- + union（并） · concatenation（连接）, □ star-closure（星闭包）
- DEFINITION 3.1



Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called **primitive regular expressions**.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in (2).

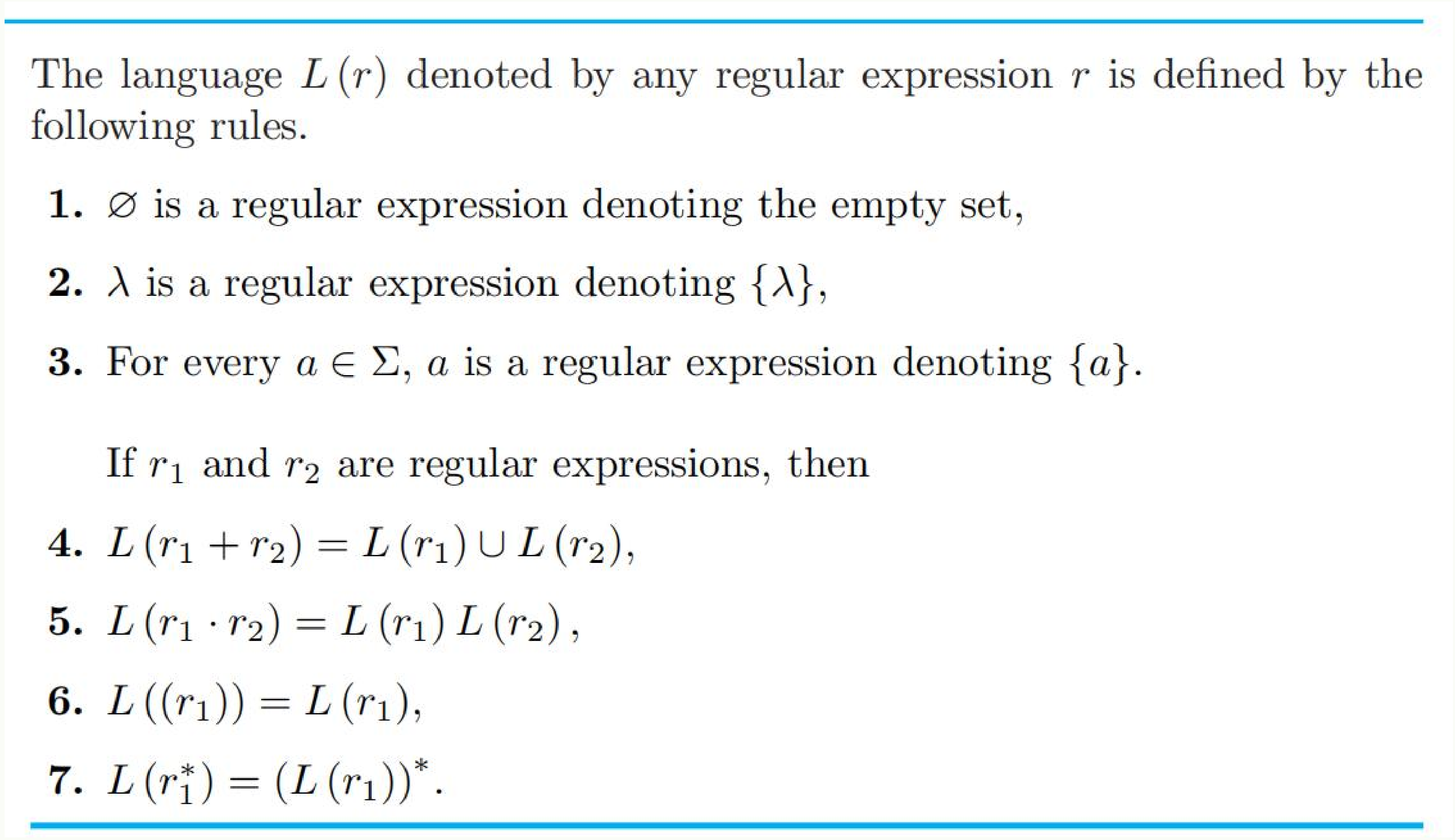
e
cy

EXAMPLE of Regular Expressions

- a stands for $\{\mathrm{a}\}$
- $\mathrm{a+b+c}$ stands for $\{\mathrm{a, b, c}\}$
- $(\mathrm{a}+(\mathrm{b}\cdot\mathrm{c}))^*$ the star-closure of $\{\mathrm{a}\}\cup\{\mathrm{bc}\}$, that is, the language $\{\lambda, \mathrm{a}, \mathrm{bc}, \mathrm{aa}, \mathrm{abc}, \mathrm{bca}, \mathrm{bcbc}, \mathrm{aaa}, \mathrm{aabc}, \ldots\}$.
- For $\Sigma = \{\mathrm{a, b, c}\}$, the string $(\mathrm{a}+\mathrm{b}\cdot\mathrm{c})^*\cdot(\mathrm{c}+\emptyset)$ is a regular expression
- $(\mathrm{a} + \mathrm{b} +)$ is not a regular expression

Languages Associated with Regular Expressions

- DEFINITION 3.2



The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,

2. λ is a regular expression denoting $\{\lambda\}$,

3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,

5. $L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$,

6. $L((r_1)) = L(r_1)$,

7. $L(r_1^*) = (L(r_1))^*$.

补充

- r is an RE on the alphabet ∑, the n-th power of r is defined as…

  ① $r^0=\lambda$

  ② $r^n=r^{n-1}r$ i.e. $(1+0)^3=(1+0)(1+0)(1+0)$

  ③ $r^+=r^*r=rr^*$ i.e. $a^+=a^*a$

EXAMPLES of RE->RL

- Precedence（优先级）rules for the operators
  - star-closure > concatenation > union.

EXAMPLE 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)\, L(a+b) \\ &= (L(a))^* \,(L(a) \cup L(b)) \\ &= \{\lambda, a, aa, aaa, \ldots\}\, \{a, b\} \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

EXAMPLE 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a+b)^*\,(a+bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

EXAMPLES of RL->RE (1)

EXAMPLE 3.5

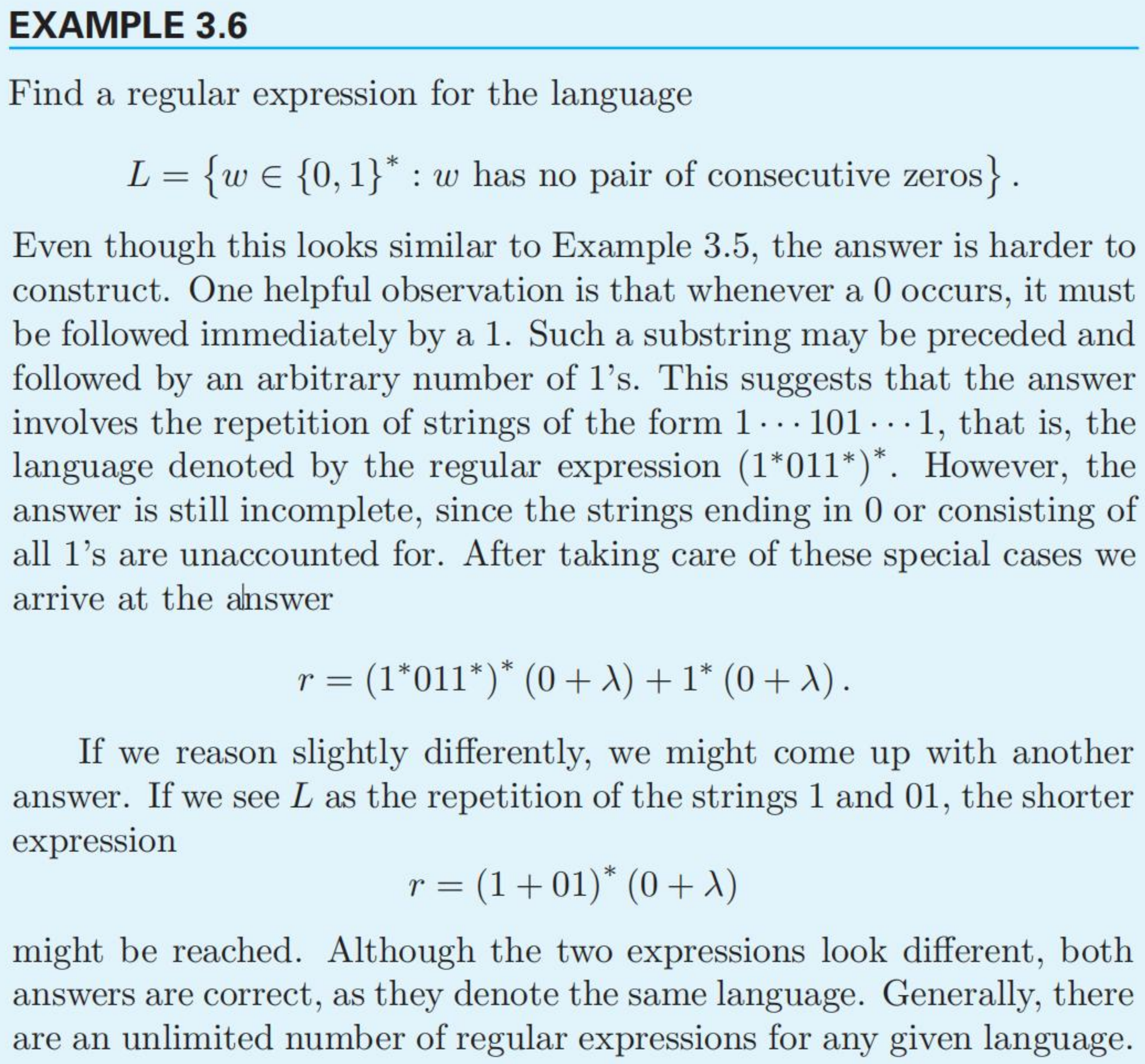
For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* \, 00 \, (0 + 1)^*.$$

EXAMPLES of RL->RE (2)



EXAMPLE 3.6

Find a regular expression for the language

$$L = \left\{ w \in \{0,1\}^* : w \text{ has no pair of consecutive zeros} \right\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever a 0 occurs, it must be followed immediately by a 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \cdots 101 \cdots 1$, that is, the language denoted by the regular expression $(1^*011^*)^*$. However, the answer is still incomplete, since the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^*011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^* (0 + \lambda)$$

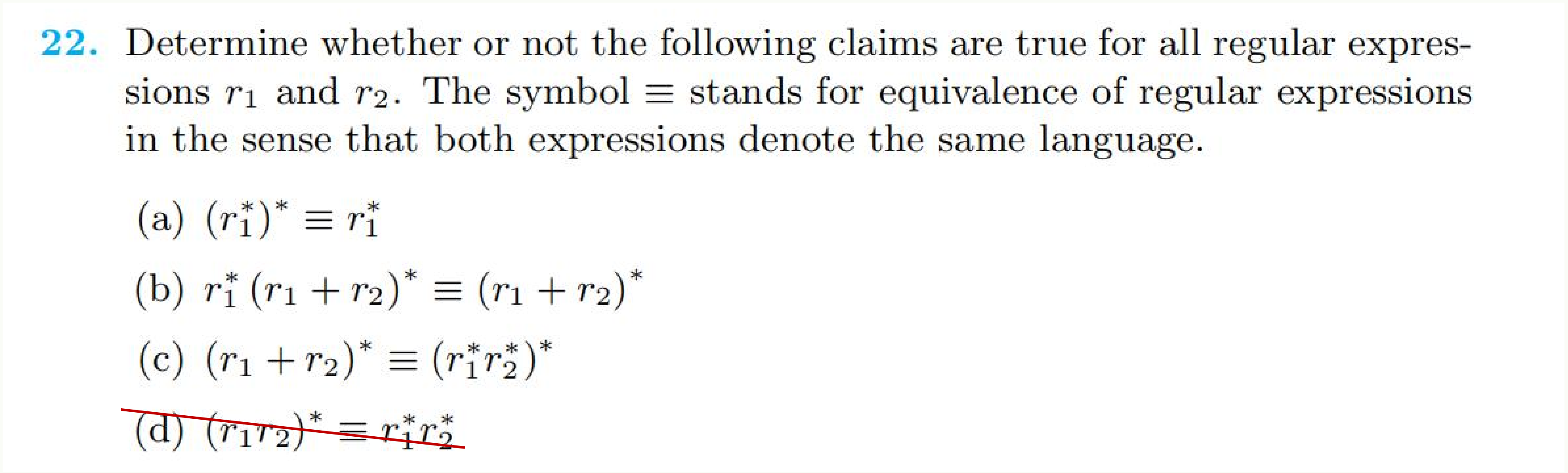
might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

EXAMPLES of RL->RE (3)

All strings of 0's and 1's	{λ,0,1,00,01,10,11,000,……}	(0+1)*
Set of all strings of 0's and 1's beginning with 0 and ending with 1	{01,001,011,0001,0011,0101,0110,00001,……}.This can be written as 0{e,0,1,00,01,10,11,000,….}1	0(0+1)*1.
Set of all strings having even number of 1's	{λ,11,1111,111111,11111111,….}	(11)*
Set of all strings having odd number of 1's	{1,111,11111,1111111,111111111,..},this can be written as {λ,11,1111,111111,11111111,……}1	1(11)* or (11)*1

Equivalence of Regular Expressions.

- Two regular expressions are equivalent if they denote the same language.
- One can derive a variety of rules for simplifying regular expressions (see Exercise 22 in the following exercise section), but since we have little need for such manipulations we will not pursue this.



22. Determine whether or not the following claims are true for all regular expressions r_1 and r_2. The symbol $\equiv$ stands for equivalence of regular expressions in the sense that both expressions denote the same language.

(a) $(r_1^*)^* \equiv r_1^*$

(b) $r_1^* (r_1 + r_2)^* \equiv (r_1 + r_2)^*$

(c) $(r_1 + r_2)^* \equiv (r_1^* r_2^*)^*$

~~(d) $(r_1 r_2)^* \equiv r_1^* r_2^*$~~

Equivalence of Regular Expressions

① $(rs)t=r(st)$

$(r+s)+t=r+(s+t)$

② $r(s+t)=rs+rt$

$(s+t)r=sr+tr$

③ $r+s=s+r$

④ $r+r=r$

⑤ $r+\Phi=r$

⑥ $r\lambda=\lambda r=r$

⑦ $r\Phi=\Phi r=\Phi$

⑧ $L(\Phi^*)=\{\lambda\}$

⑨ $L((r+\lambda)^*)=L(r^*)$

⑩ $L((r^*)^*)=L(r^*)$

⑪ $L((r^*s^*)^*)=L((r+s)^*)$

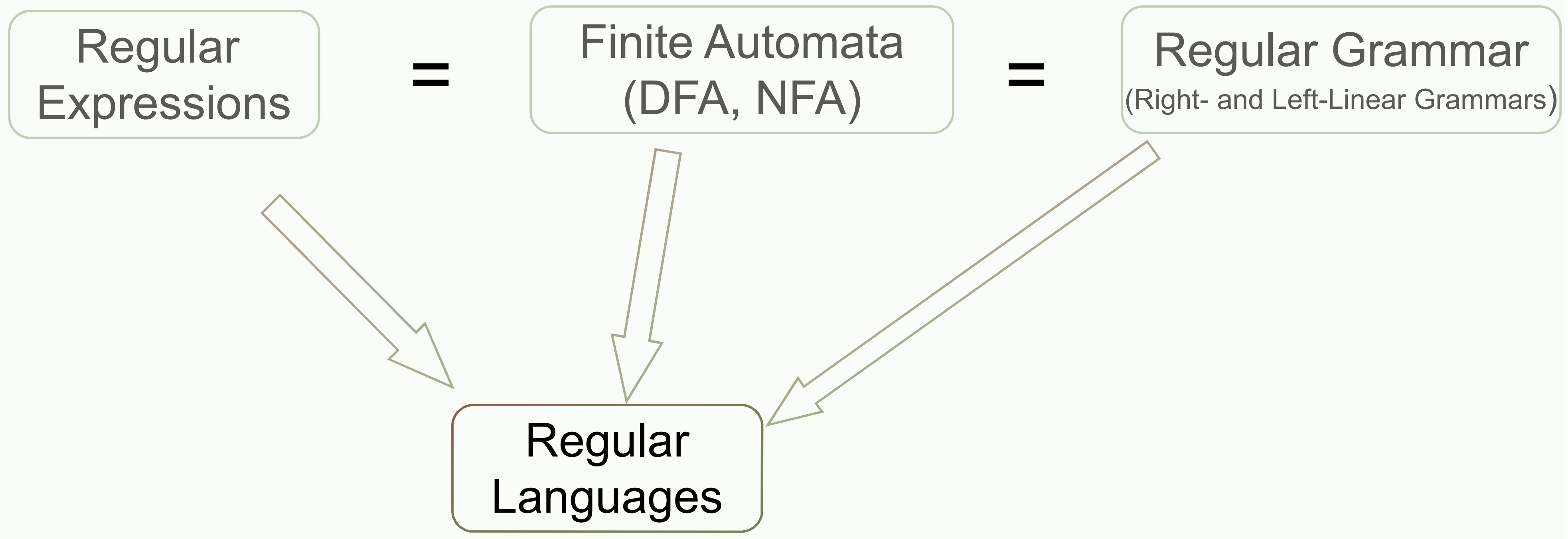
⑫ If $L(r)\subseteq L(s)$， $r+s=s$

⑬ $L(r^n)=(L(r))^n$

⑭ $r^n r^m=r^{n+m}$

Generally， $r+\lambda\neq r$， $(rs)^n\neq r^n s^n$， $rs\neq sr$

Regular Expression have Equivalent Automata



3.2 CONNECTION BETWEEN REGULAR EXPRESSIONS AND REGULAR LANGUAGES

- For every regular language there is a regular expression, and for every regular expression there is a regular language.
- Next
    - Regular expression -> nfa
    - Nfa -> regular expression

Regular Expression ->nfa

- If r is a regular expression, then L(r) is a regular language.
- How to construct an nfa that accepts L(r)
  - The construction for this relies on the recursive definition for L(r).
  - We first construct simple automata for parts (1), (2), and (3) of Definition 3.2, then show how they can be combined to implement the more complicated parts (4), (5), and (7).

THEOREM 3.1 (1)

- **Proof by induction on the number of operators**: We begin with automata that accept the languages for the simple regular expressions $\emptyset$, λ, and $\mathrm{a} \in \Sigma$. These are shown in Figure 3.1(a), (b), and (c)

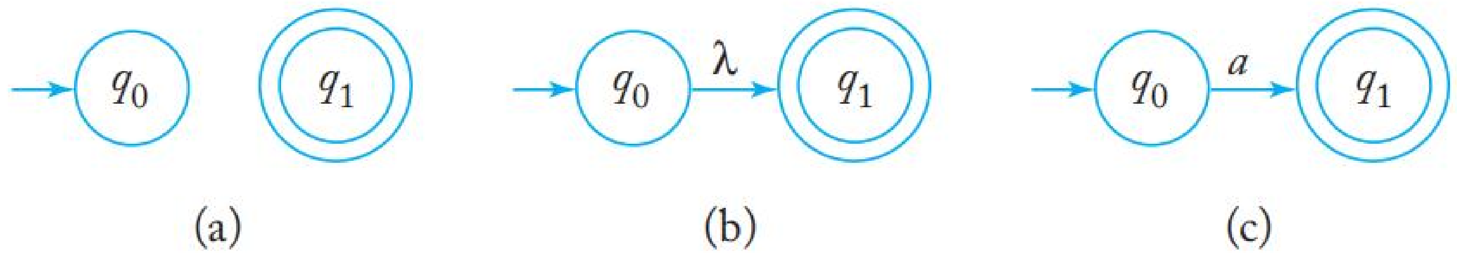


FIGURE 3.1 (a) nfa accepts $\varnothing$. (b) nfa accepts $\{\lambda\}$. (c) nfa accepts $\{a\}$.

THEOREM 3.1 (2)

- Assume that we have automata $\mathrm{M}(\mathrm{r}_1)$ and $\mathrm{M}(\mathrm{r}_2)$ that accept languages denoted by regular expressions r_1 and r_2,respectively.
- Represent them schematically, as in Figure 3.2.

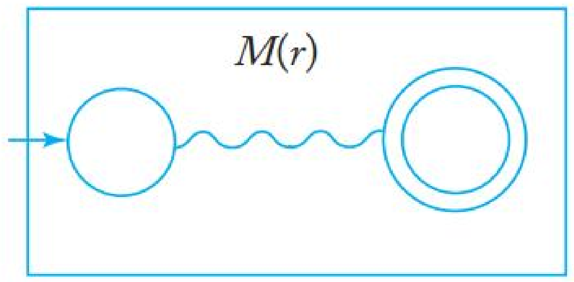


FIGURE 3.2 Schematic representation of an nfa accepting $L(r)$.

- For every nfa there is an equivalent one with a single final state.

THEOREM 3.1 (2)

- As indicated in the drawings, the initial and final states of the constituent machines lose their status and are replaced by new initial and final states.
- Prove by inductionon the number of operators that the construction yields an automaton that accepts the language denoted by any particular regular expression.

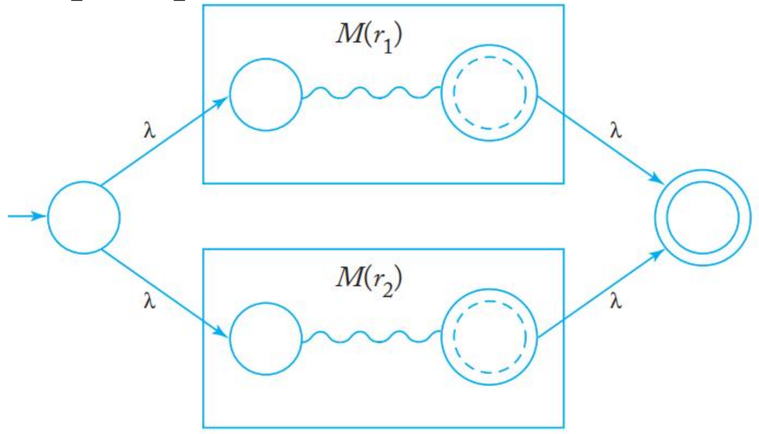


FIGURE 3.3 Automaton for $L(r_1 + r_2)$.

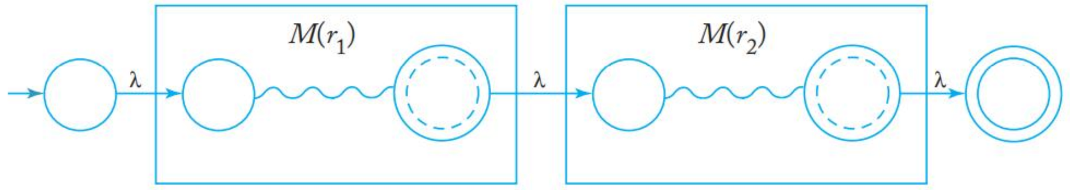


FIGURE 3.4 Automaton for $L(r_1 r_2)$.

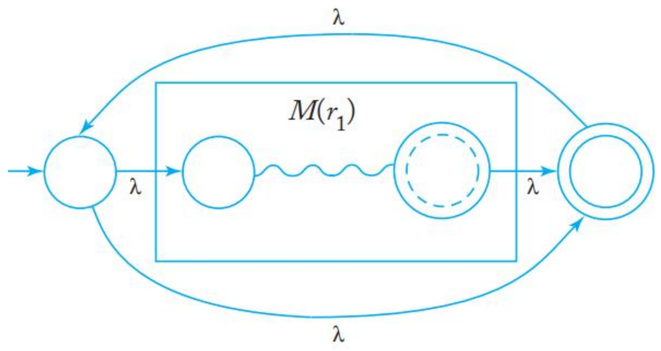


FIGURE 3.5 Automaton for $L(r_1^*)$.

EXAMPLE 3.7

Find an nfa that accepts $L(r)$, where

$$r = (a + bb)^* (ba^* + \lambda).$$

Automata for $(a + bb)$ and $(ba^* + \lambda)$, constructed directly from first principles, are given in Figure 3.6. Putting these together using the construction in Theorem 3.1, we get the solution in Figure 3.7.

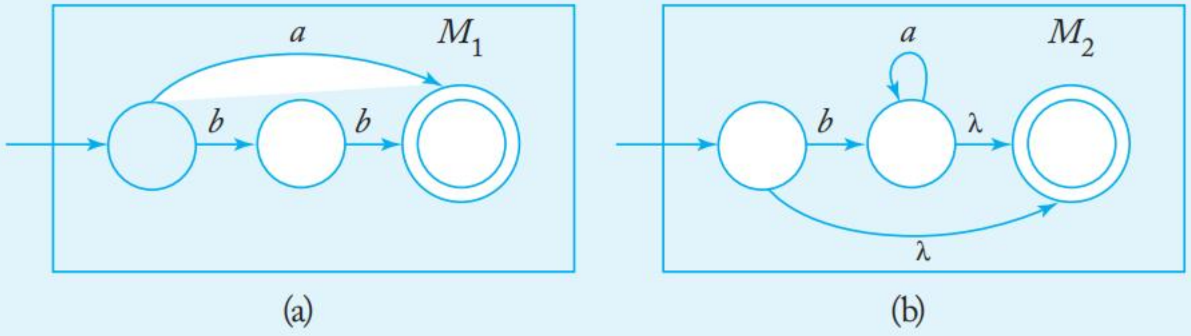


FIGURE 3.6 (a) M_1 accepts $L(a + bb)$. (b) M_2 accepts $L(ba^* + \lambda)$.

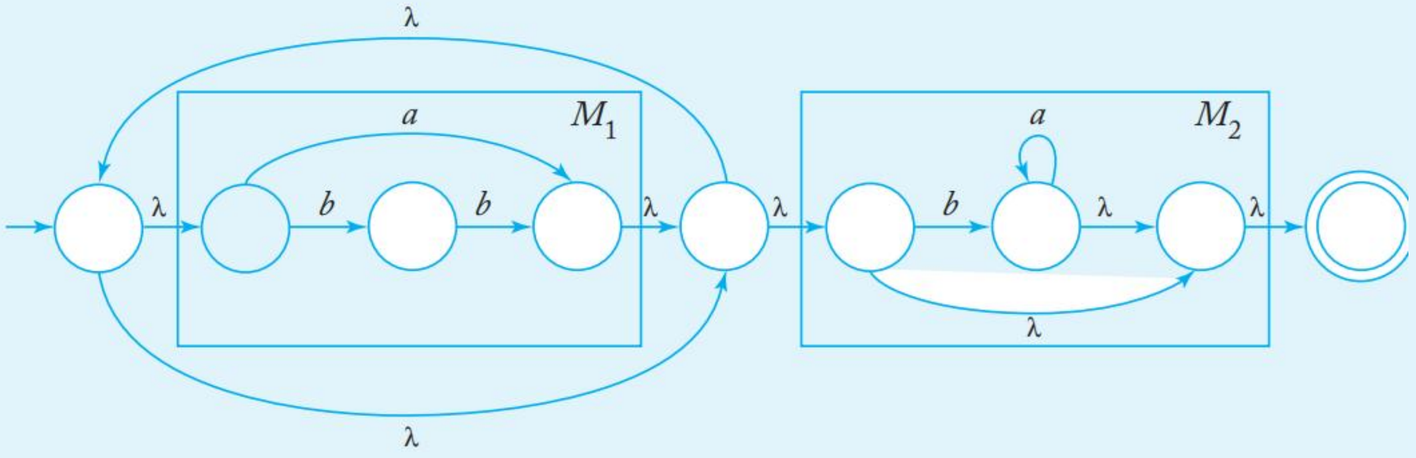


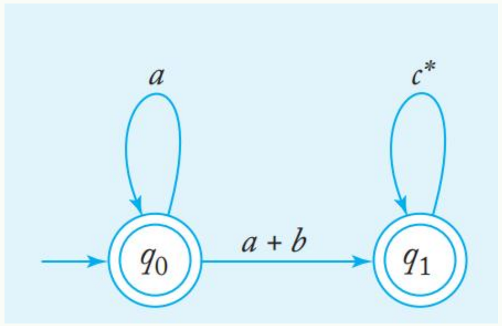
FIGURE 3.7 Automaton accepts $L((a + bb)^* \ (ba^* + \lambda))$.

补充

- For every RE, there is an accepting FA. If the FA constructed from both of the REs are the same, then we can say that two REs are equivalent.

Nfa->Regular Expression(1)

- All we need to do is to find a regular expression capable of generating the labels of all the walks from q_0 to any final state.
- A generalized transition graph(通用转移图) is a transition graph whose edges are labeled with regular expressions

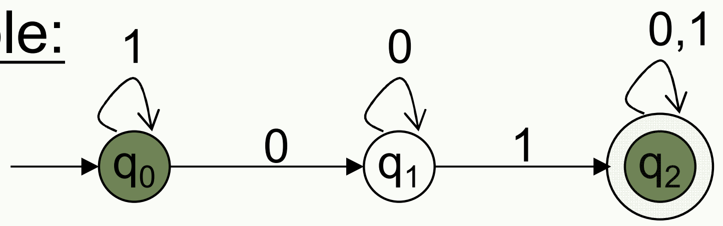


L(a□ +a□ (a+b)c□)

EXAMPLE

Informally, trace all distinct paths (traversing cycles only once)
from the start state to *each of the* final states
and enumerate all the expressions along the way

Example:



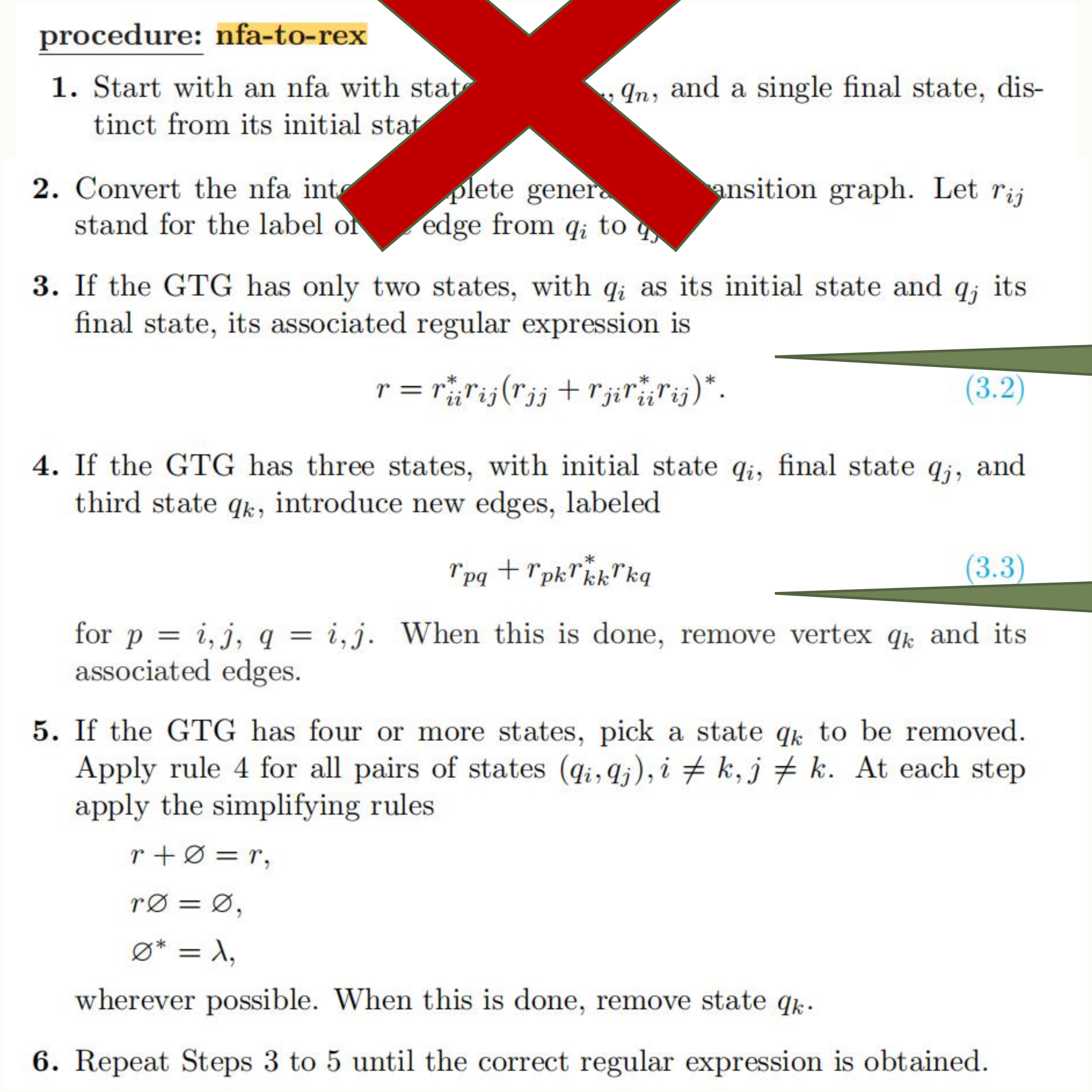
$$\underbrace{(1^*)}_{1^*}\ \underbrace{0\quad (0^*)}_{00^*}\ \underbrace{1}_{1}\ \underbrace{(0+1)^*}_{(0+1)^*}$$

$$\Downarrow$$

$$1^*00^*1(0+1)^*$$

Q) What is the language?

Nfa->RE（请使用下一页方法）



procedure: nfa-to-rex

1. Start with an nfa with stat ..., q_n, and a single final state, dis-tinct from its initial stat

2. Convert the nfa int plete genera ansition graph. Let r_{ij} stand for the label of edge from q_i to q_j

3. If the GTG has only two states, with q_i as its initial state and q_j its final state, its associated regular expression is

$$r = r_{ii}^* r_{ij} (r_{jj} + r_{ji} r_{ii}^* r_{ij})^*. \tag{3.2}$$

4. If the GTG has three states, with initial state q_i, final state q_j, and third state q_k, introduce new edges, labeled

$$r_{pq} + r_{pk} r_{kk}^* r_{kq} \tag{3.3}$$

for $p = i, j$, $q = i, j$. When this is done, remove vertex q_k and its associated edges.

5. If the GTG has four or more states, pick a state q_k to be removed. Apply rule 4 for all pairs of states $(q_i, q_j), i \neq k, j \neq k$. At each step apply the simplifying rules

$$r + \varnothing = r,$$

$$r\varnothing = \varnothing,$$

$$\varnothing^* = \lambda,$$

wherever possible. When this is done, remove state q_k.

6. Repeat Steps 3 to 5 until the correct regular expression is obtained.

此式不方便记忆

参考书此处的变量命名不清晰

RL to RE（图上作业法）

1．预处理：

① 用标记为X和Y的状态将M“括起来”：

在状态转移图中增加标记为X和Y的状态，从标记为X的状态到标记为q_0的状态引一条标记为 λ 的弧；从标记为q(q∈F)的状态到标记为Y的状态分别引一条标记为 λ 的弧。

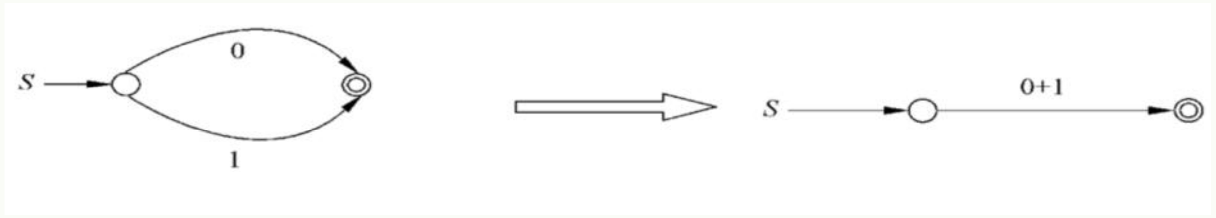
② 去掉所有的不可达状态。

RL to RE

2．对通过步骤(1)处理所得到的状态转移图重复如下操作，直到该图中不再包含除了标记为X和Y外的其他状态，并且这两个状态之间最多只有一条弧。

①并弧

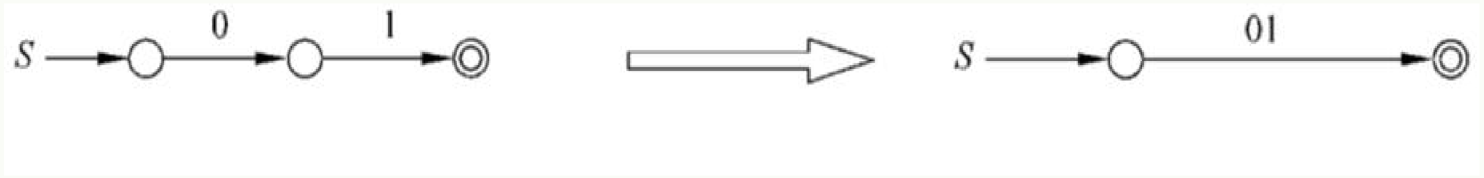
- 将从q到p的标记为r_1，r_2，…，r_g并行弧用从q到p的、标记为$r_1+r_2+\ldots+r_g$的弧取代这g个并行弧。



RL to RE （图上作业法）

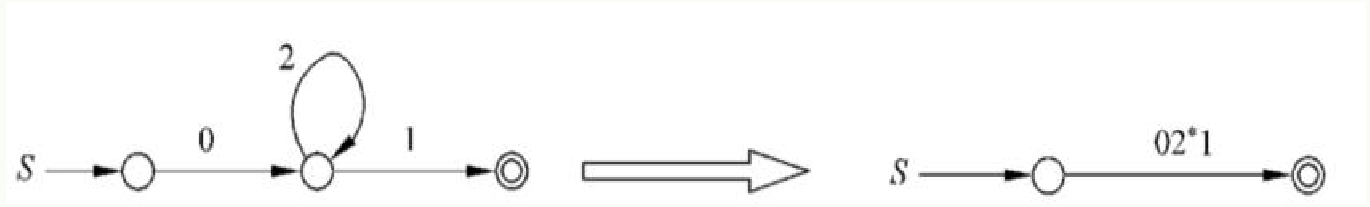
②去状态的情况1

如果从q到p有一条标记为r_1的弧，从p到t有一条标记为r_2的弧，不存在从状态p到状态p的弧，将状态p和与之关联的这两条弧去掉，用一条从q到t的标记为r_1r_2的弧代替。



③去状态的情况2

- 如果从q到p有一条标记为r_1的弧，从p到t有一条标记为r_2的弧，从状态p到状态p标记为r_3的弧，将状态p和与之关联的这三条弧去掉，用一条从q到t的标记为$r_1r_3{}^*r_2$的弧代替。



RL to RE （图上作业法）

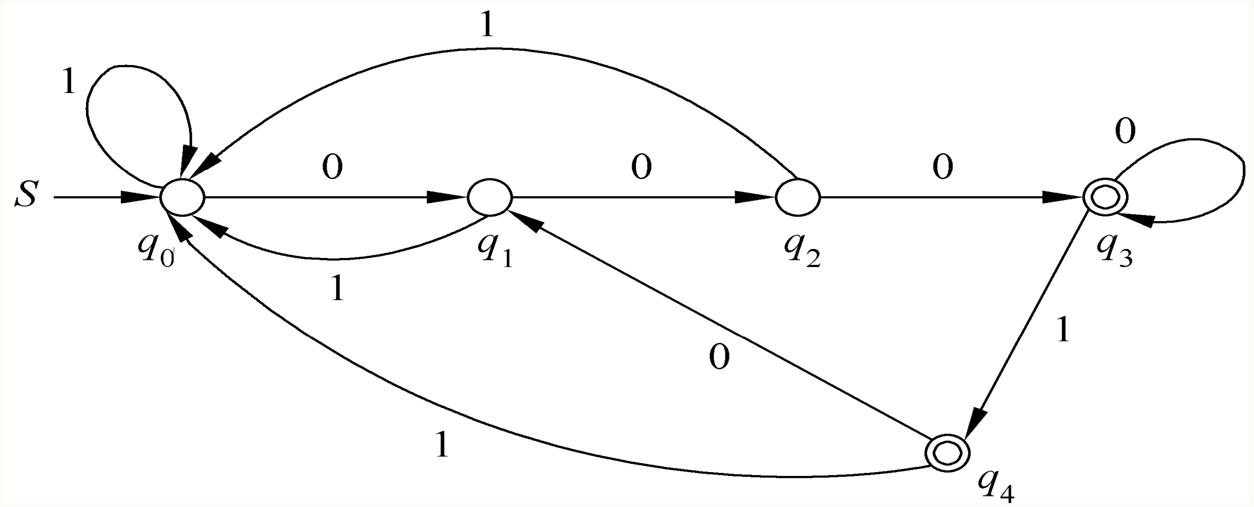
④去状态的情况3

如果图中只有三个状态，而且不存在从标记为X的状态到达标记为Y的状态的路，则将除标记为X的状态和标记为Y的状态之外的第3个状态及其相关的弧全部删除。

3．从标记为X的状态到标记为Y的状态的弧的标记为所求的正则表达式。如果此弧不存在，则所求的正则表达式为 Φ 。

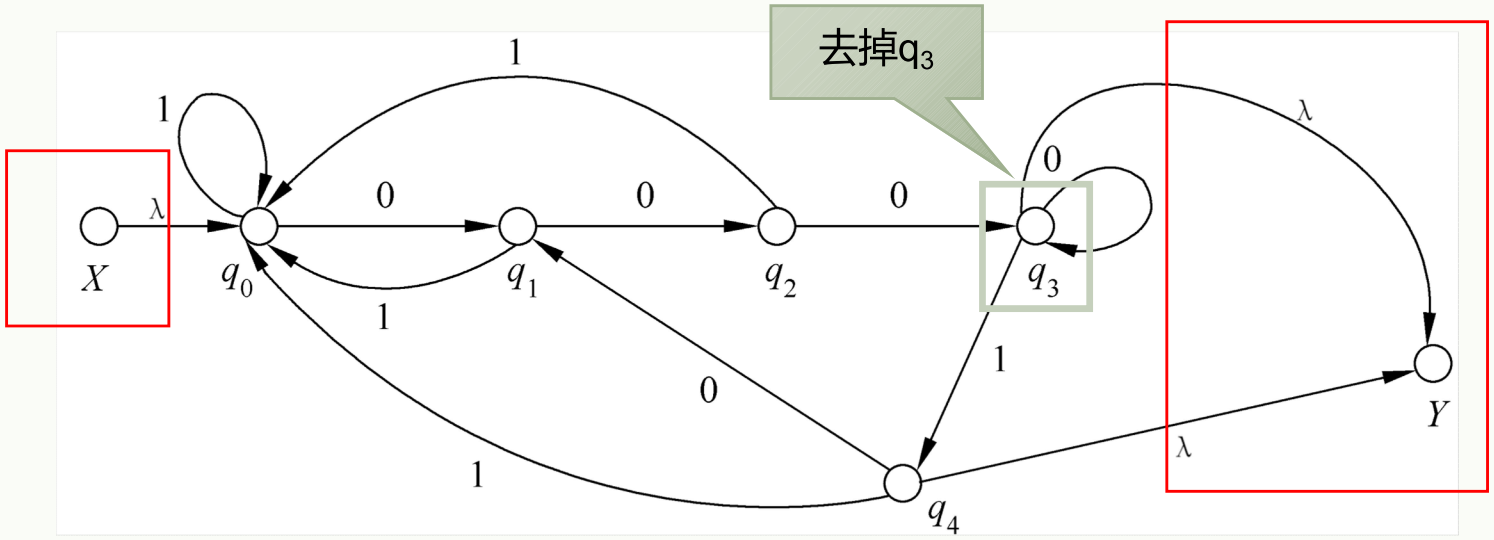
RL to RE （图上作业法）

- 例求如图所示的FA等价的RE 。

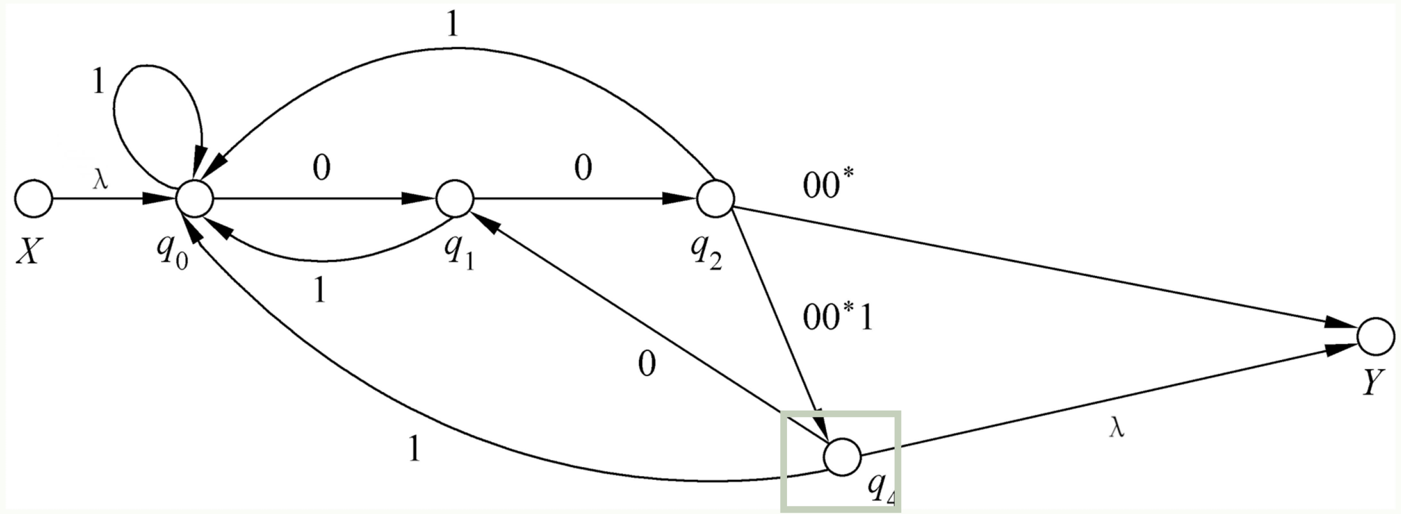


RL to RE （图上作业法）

(1)预处理。

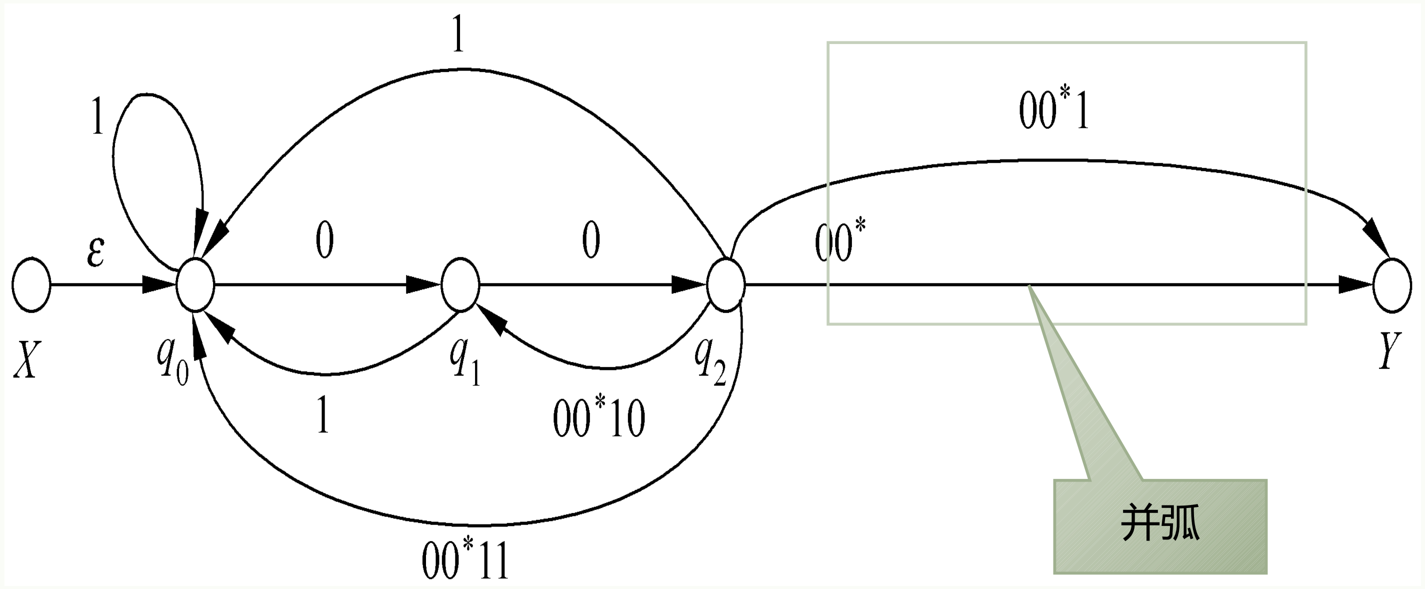


RL to RE （图上作业法）

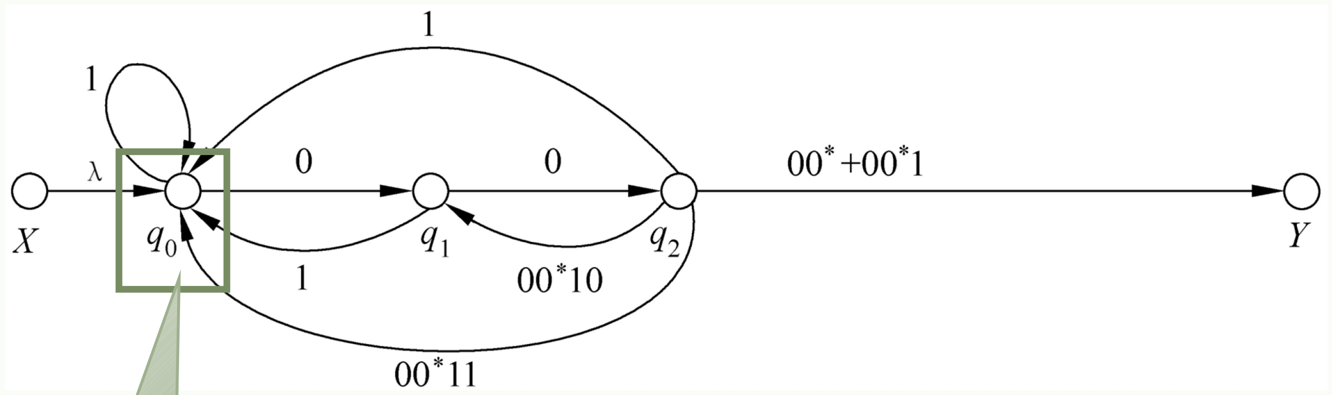


去掉q_4

RL to RE （图上作业法）

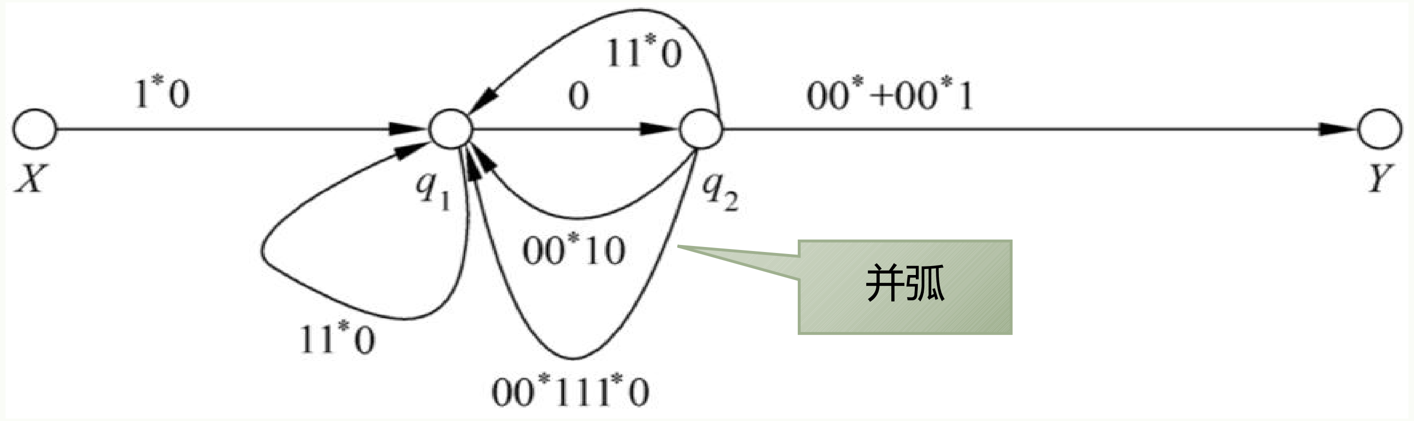


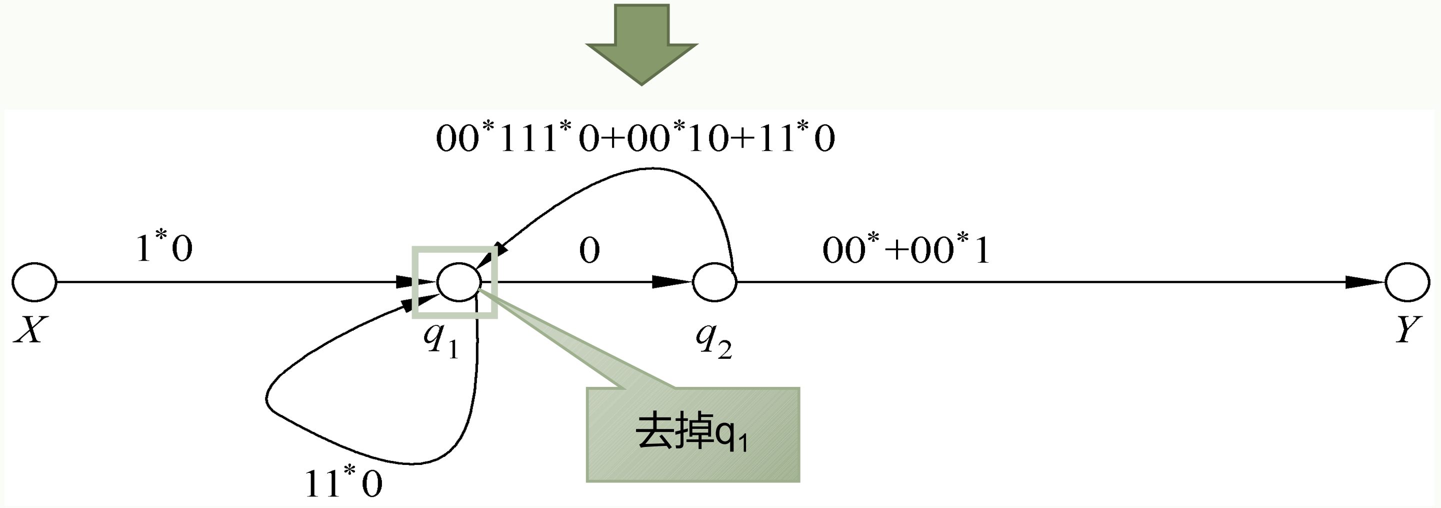
RL to RE （图上作业法）



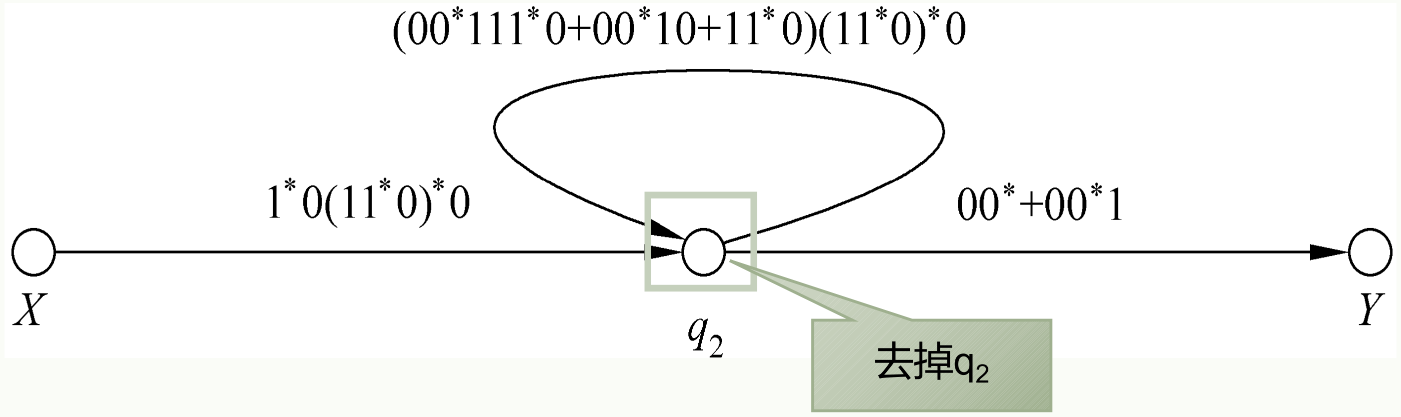
去掉q_0

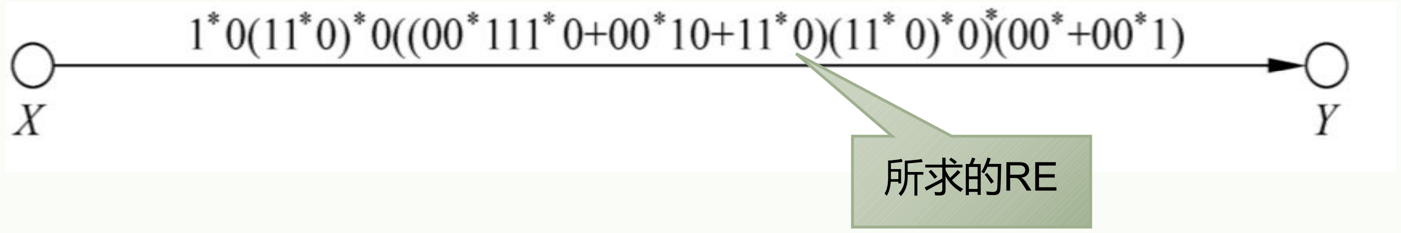
RL to RE （图上作业法）





RL to RE （图上作业法）





Regular Expressions for Describing Simple Patterns

- In many programming languages the set of integer constants is defined by the regular expression

  $$\mathrm{sdd}^*,$$

  where s stands for the sign, with possible values from $\{+, -, \lambda\}$, and d standsfor the digits 0 to 9.
- The *vi* editor in the *UNIX* operating system recognizes the command $/\mathrm{aba}^*\mathrm{c}/$ as an instruction to search the file for the first occurrence of the string ab, followed by an arbitrary number of a' s, followed by a c.

END