有限オートマトンと正規表現

離散数学・オートマトン
2021 年後期
佐賀大学理工学部　只木進一

1. 正規表現: Regular expression
2. 正規表現と有限オートマトン
3. 正規表現から DFA へ
4. 有限オートマトンの受理言語を正規表現で構成

正規表現: Regular expression

- 文字列の探索や置換で利用
- 柔軟にパターンを記述できる
- 例
  - “000”の繰り返しを含む
  - 数字が偶数個連続する
  - 指定した文字列の後ろに数字が付いているファイル名

例 1

ソースコード 1: ファイル名から日付部分を除く

```
1  import re
2  fileList=[
3      '1.1 Introduction_20201010.txt',
4      '1.2 SetAndMapppings_20211013.txt',
5      '1.3 Relations_20100401.txt'
6  ]
7  #日付以外の部分を取り出す
8  p = re.compile(r'(.*)_\d{8}(\.txt)')
9  for f in fileList:
10     m = p.match(f)
11     if m:
12         filename = m.group(1)+m.group(2)
13         print(filename)
```

出力結果

```
1.1 Introduction.txt
1.2 SetAndMapppings.txt
1.3 Relations.txt
```

正規表現の定義: 基礎

- $a \in \Sigma$ に対して a は正規表現であり、その言語は $\{a\}$ である
  - 一文字からなる言語
- ϵ は正規表現であり、その言語は $\{\epsilon\}$ である
  - 長さゼロの文字列からなる言語
- $\emptyset$ は正規表現であり、その言語は $\emptyset$ である

正規表現の定義: 再帰

- α と β が L_α 及び L_β をそれぞれ表す正規表現のとき
  - $(\alpha + \beta)$ は $L_\alpha \cup L_\beta$ を表す正規表現
  - $(\alpha\beta)$ は $L_\alpha L_\beta$(連接) を表す正規表現

$$L_\alpha L_\beta = \{uv | u \in L_\alpha, v \in L_\beta\} \tag{1}$$

  - α^* は Kleene 閉包: $L_\alpha^* = \bigcup_{k=0}^{\infty} L_\alpha^k$

$$L_\alpha^0 = \{\epsilon\}, L_\alpha^1 = L_\alpha, L_\alpha^{k+1} = L_\alpha L_\alpha^k \tag{2}$$

  - α^+ は正閉包：$L_\alpha^+ = \bigcup_{k=1}^{\infty} L_\alpha^k$

例 2

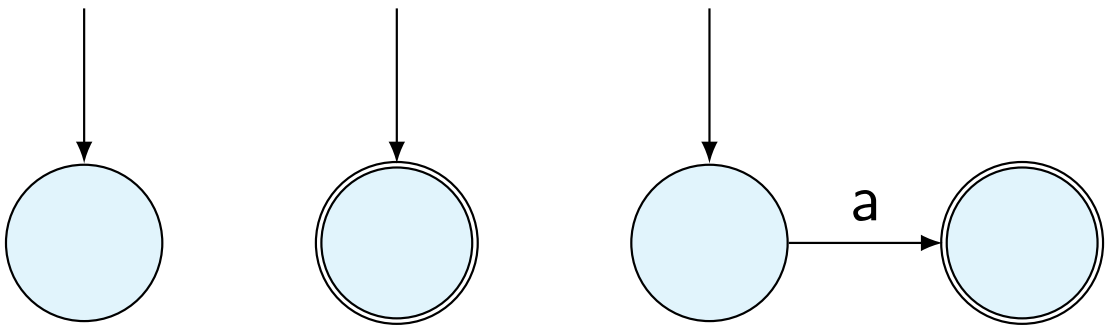
- $\mathrm{a}, \mathrm{b} \in \Sigma$
- **a** は言語 $\{\mathrm{a}\}$ を表す
- **b** は言語 $\{\mathrm{b}\}$ を表す
- $\mathbf{a} + \mathbf{b}$ は言語 $\{\mathrm{a}, \mathrm{b}\}$ を表す
- **ab** は言語 $\{\mathrm{ab}\}$ を表す
- $\mathbf{a}\,(\mathbf{a} + \mathbf{b})\,\mathbf{b}$ は言語 $\{\mathrm{aab}, \mathrm{abb}\}$ を表す
- $(\mathbf{a} + \mathbf{b})^*$ は、a と b からなる、長さゼロ以上の文字列全体からなる言語を表す

正規表現と有限オートマトン

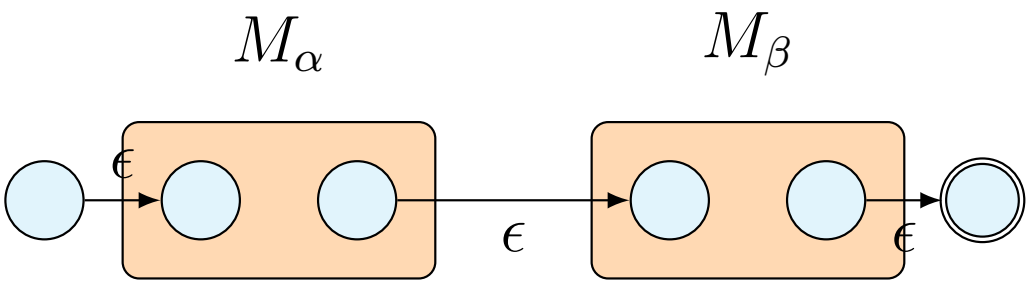
- 任意の正規表現を受理言語とする有限オートマトンを構成することができる
- 任意の有限オートマトンの受理言語を表す正規表現を構成することができる
- つまり、有限オートマトンの受理言語は正規表現で表される。

正規表現を受理する有限オートマトン

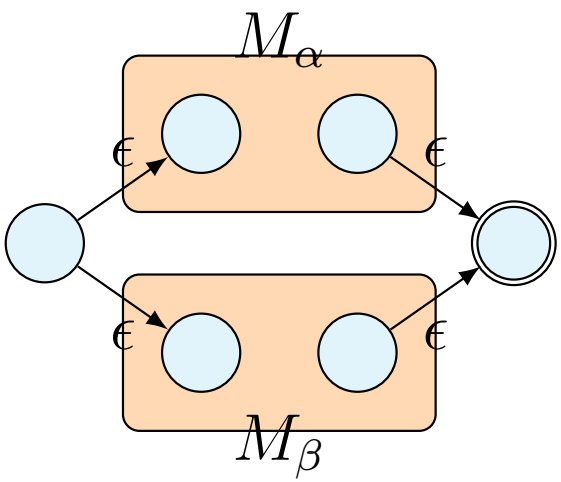
- 正規表現の構成を順を追って FA で表現
- 基礎:$a, b \in \Sigma$



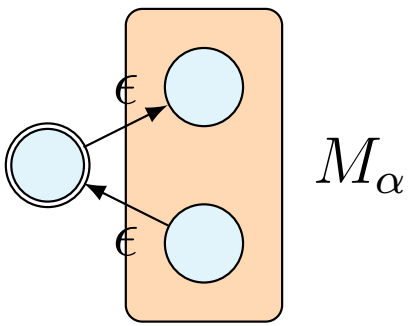
和、連接、Kleene 閉包



$\alpha\beta$

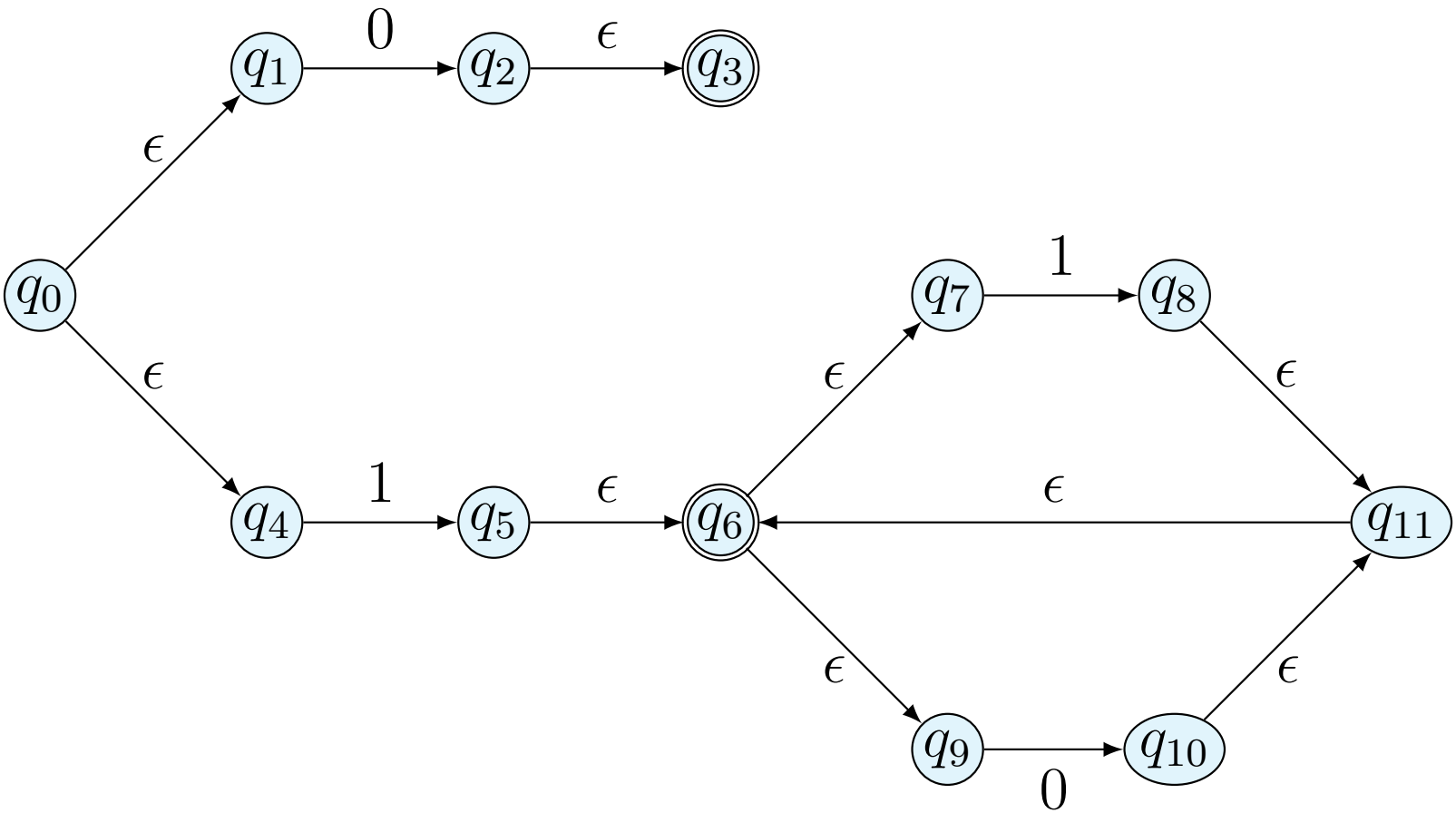


$\alpha + \beta$

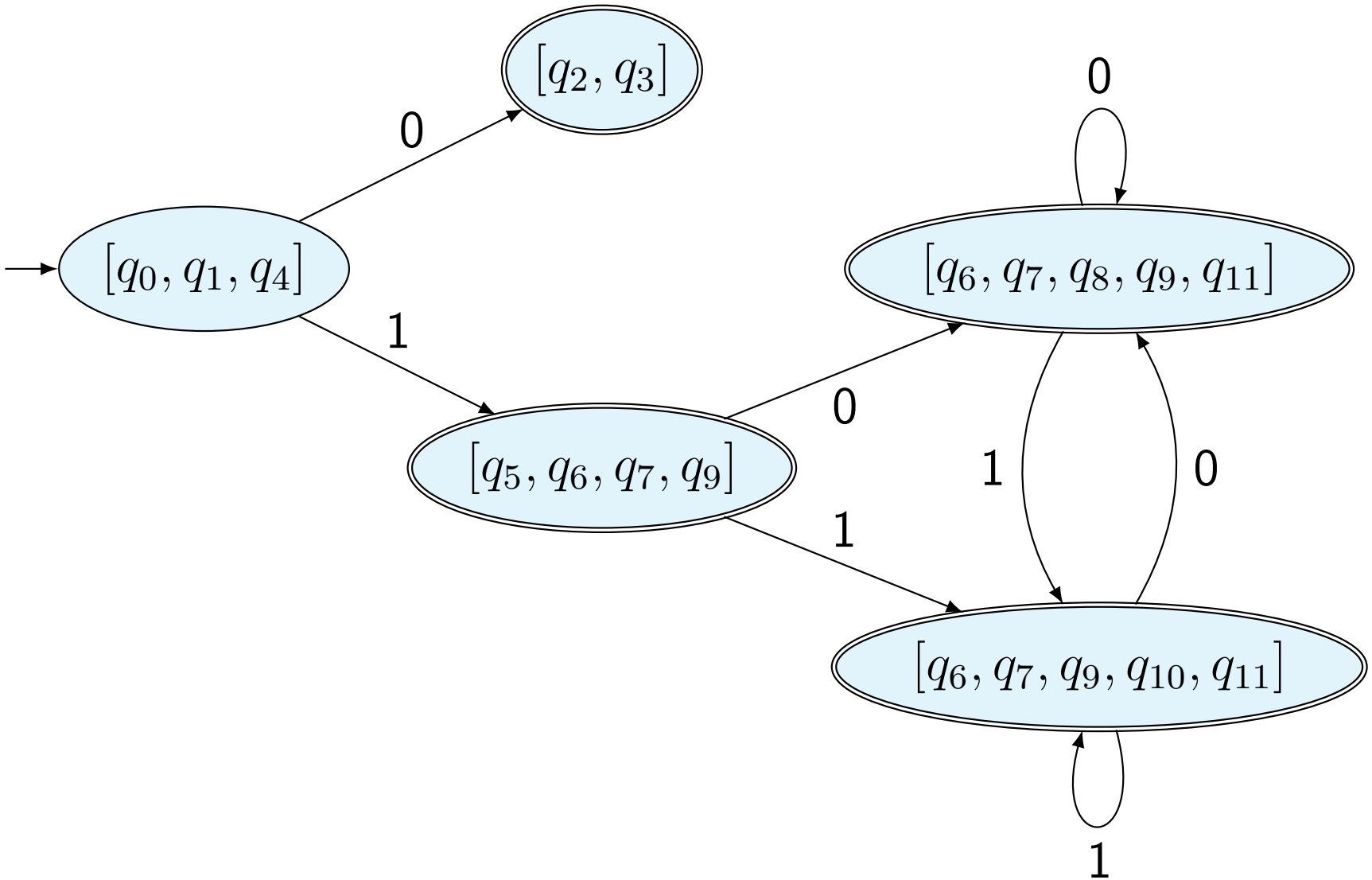


α^*

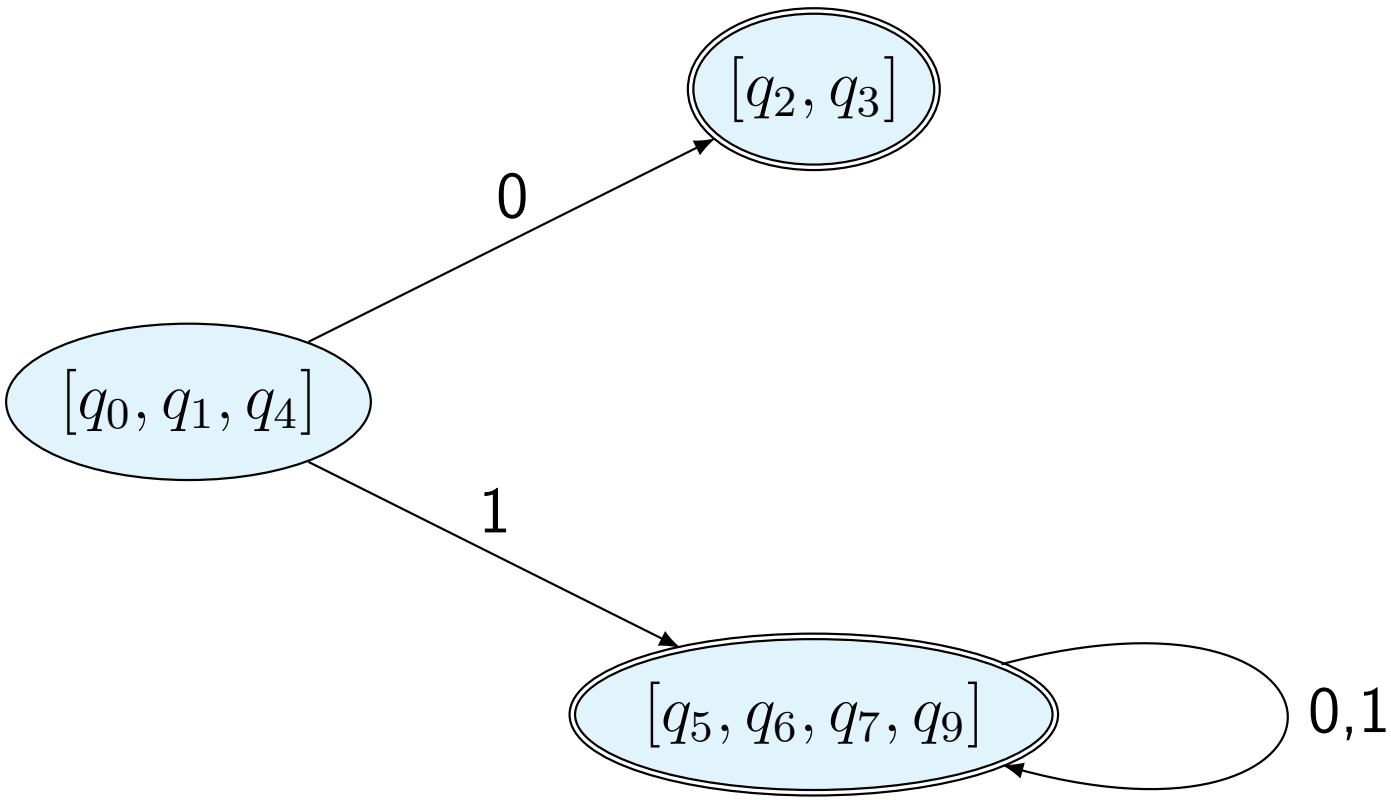
例 $1: 0 + 1(0 + 1)^*$



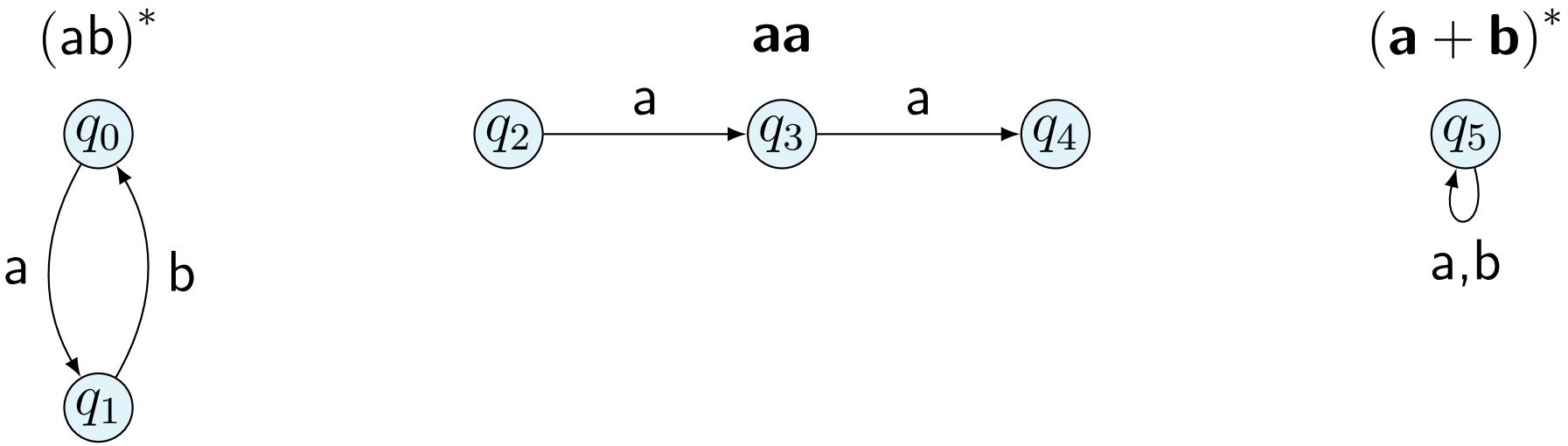
DFA へ変換

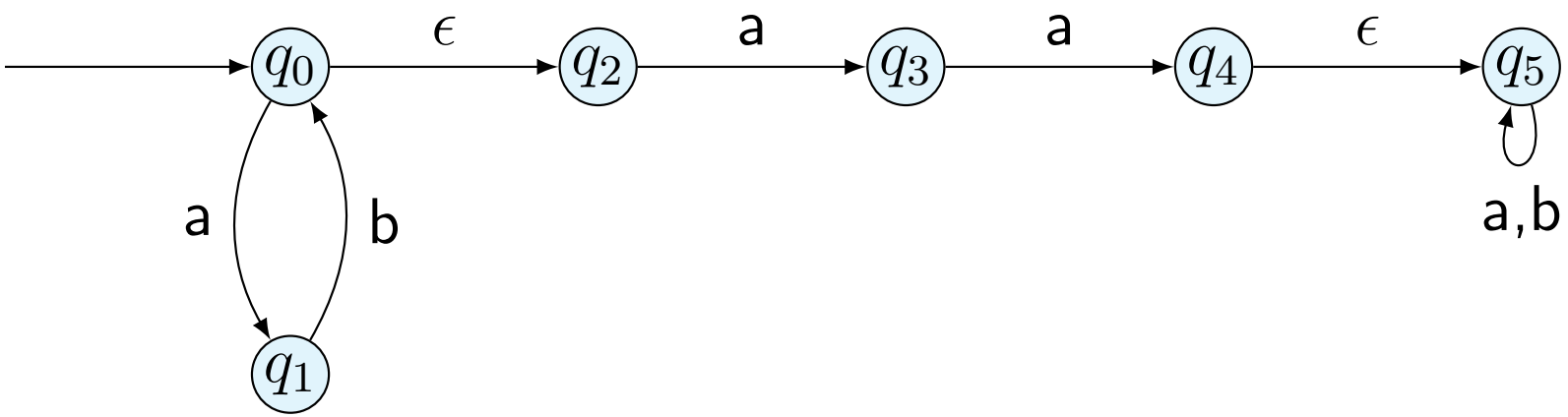


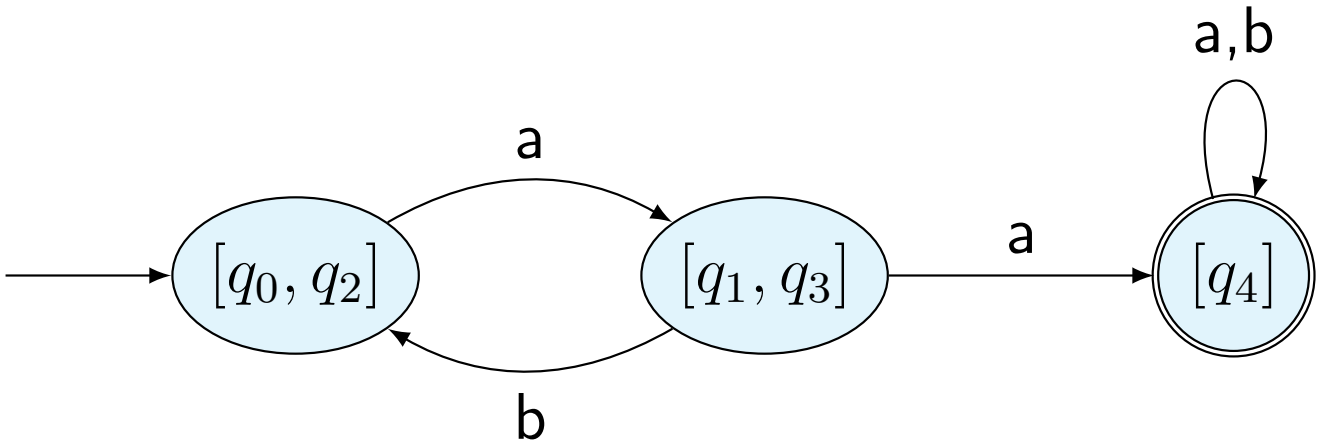
DFA を最小化



例 2: $(\mathbf{ab})^* \mathbf{aa} (\mathbf{a} + \mathbf{b})^*$

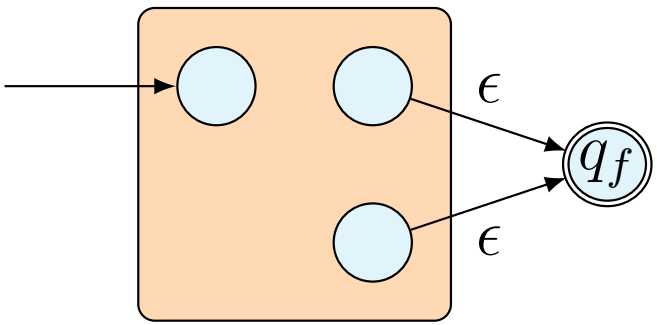



q_0
q_1
q_2
q_3
q_4
q_5
ϵ
a
a
ϵ
a
b
a,b

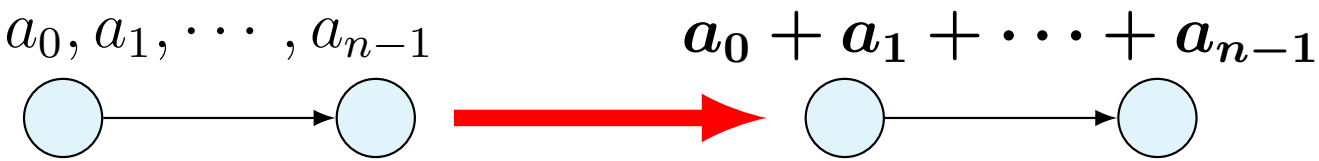

a,b
a
a
$[q_0, q_2]$
$[q_1, q_3]$
$[q_4]$
b

有限オートマトンの受理言語を正規表現で構成

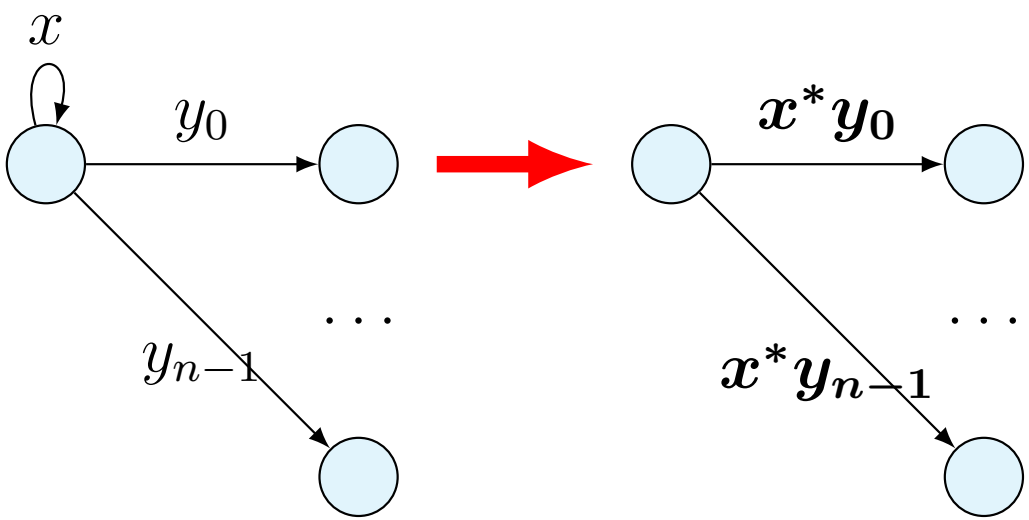
- Step1: 新たに一つの終状態 q_f を追加し、それのみが終状態とする



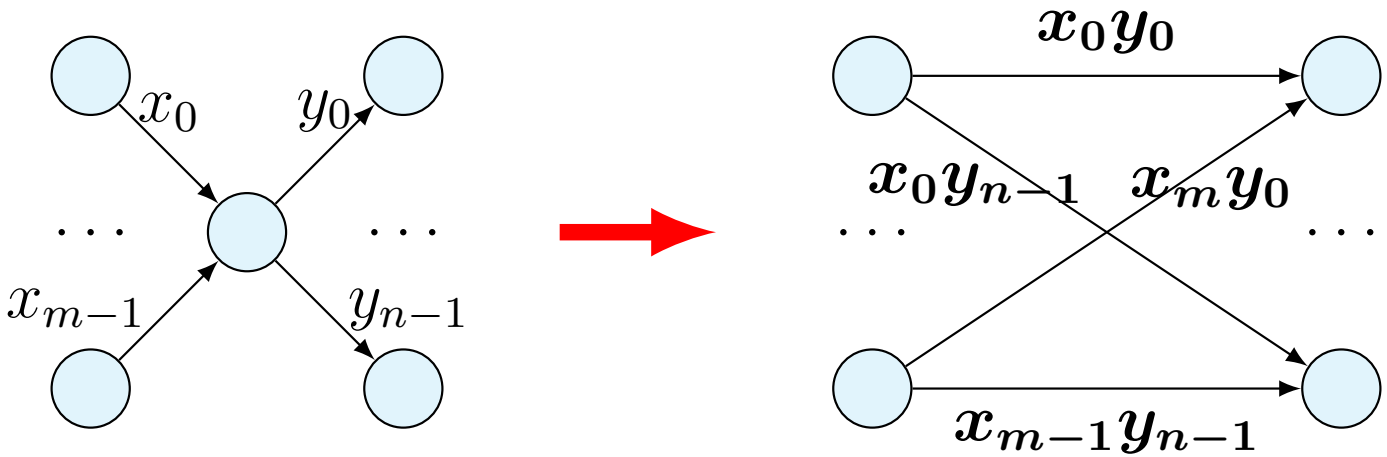
- step2: rule1,2,3 の順に適用する
- rule1: 同じ状態遷移を引き起こす入力に対して、正規表現の和を対応つける



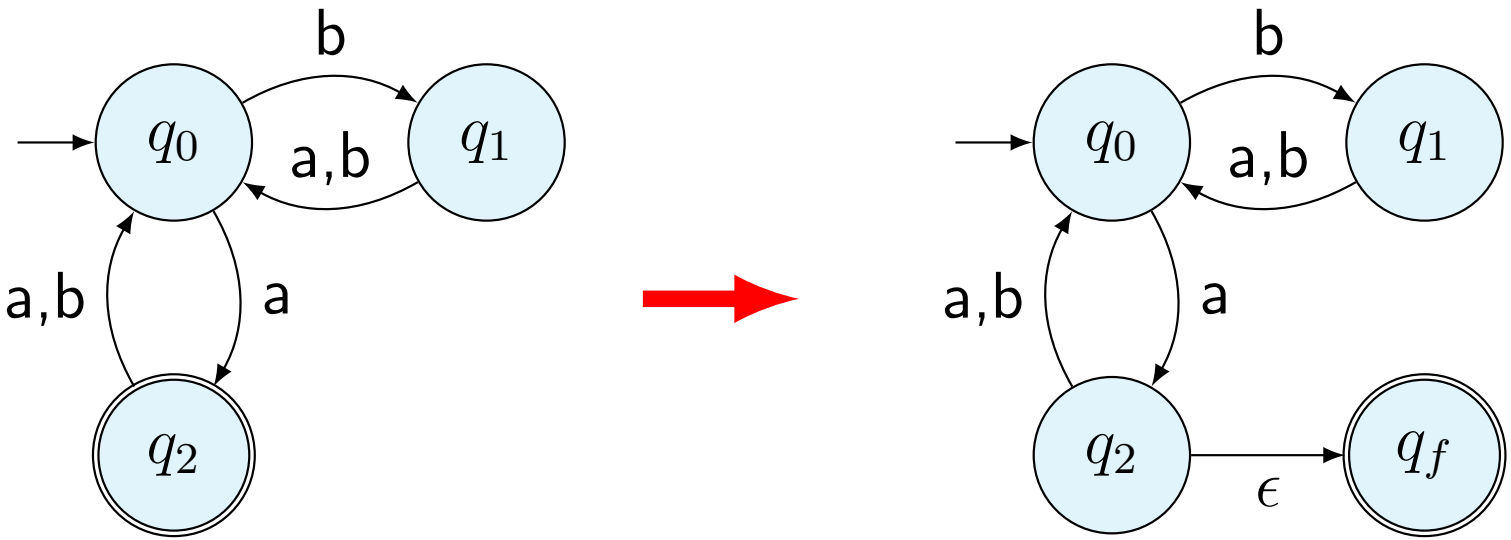
- rule2: **ループの遷移を次の遷移の前に連接する**



- rule3: 連続する遷移を、途中の状態を削除して連接とする

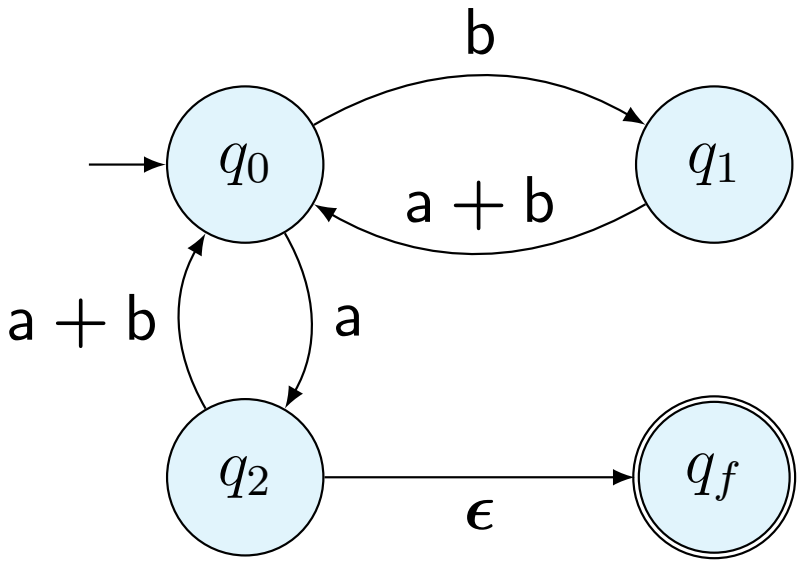


例 1



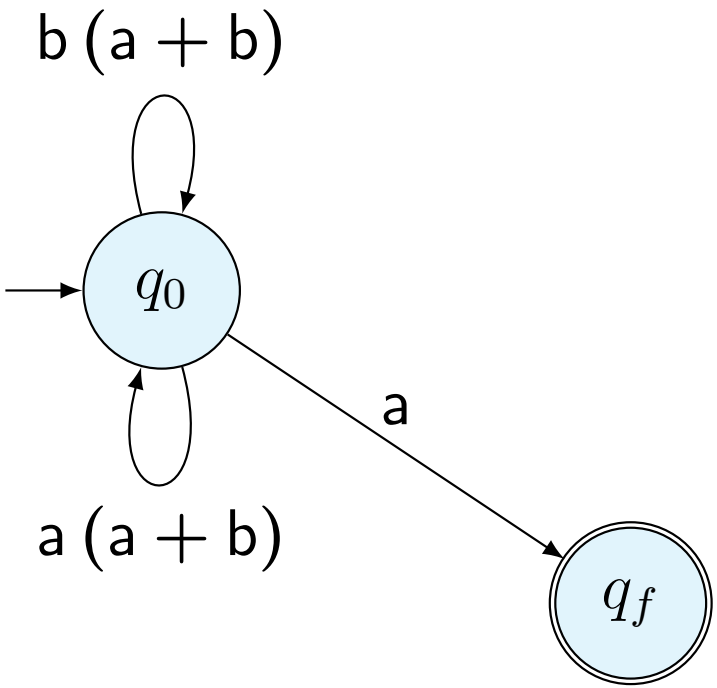
rule 1

同じ繊維を起こす入力を正規表現の和に変換



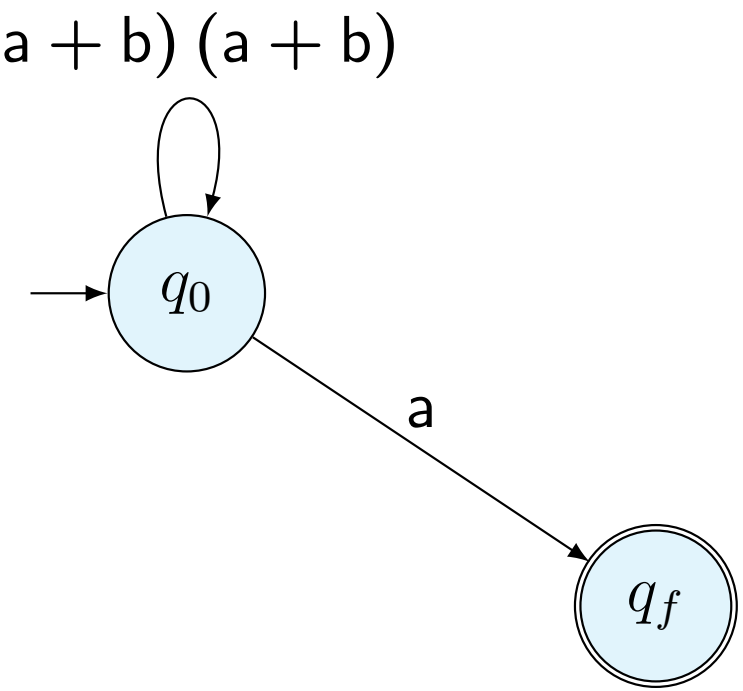
rule 3

- rule 2 に相当するループが
- q_0 から q_1 を経て、q_0 へ戻る経路をループに
-
- q_0 から q_2 を経て、q_0 へ戻る経路をループにするとともに、a_f への遷移を残す



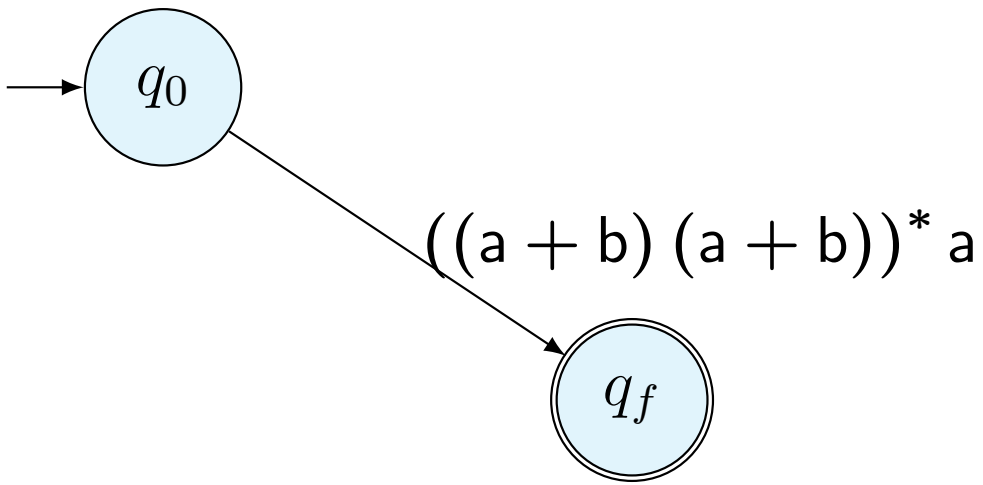
rule 1: 2 回目

- q_0 の 2 つのループの表現の和を作る

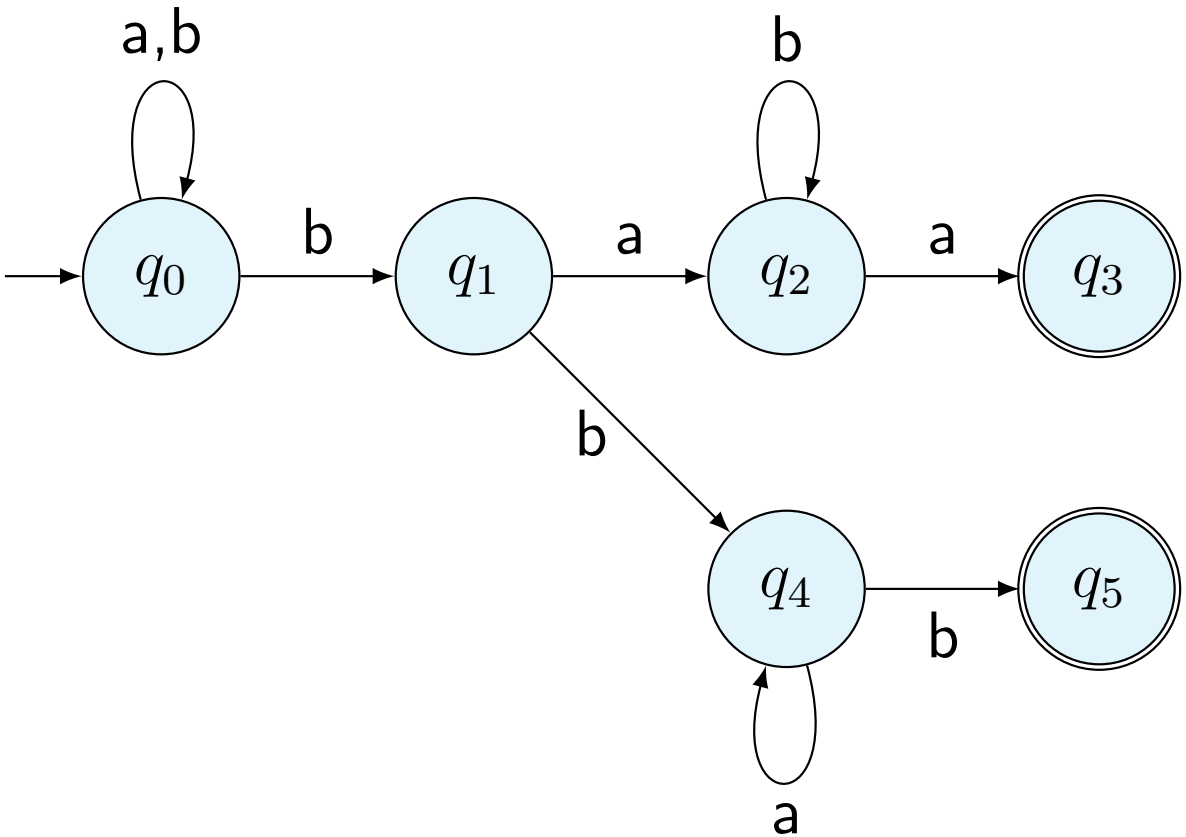


rule 2

- q_0 の２つのループの表現の和を作る

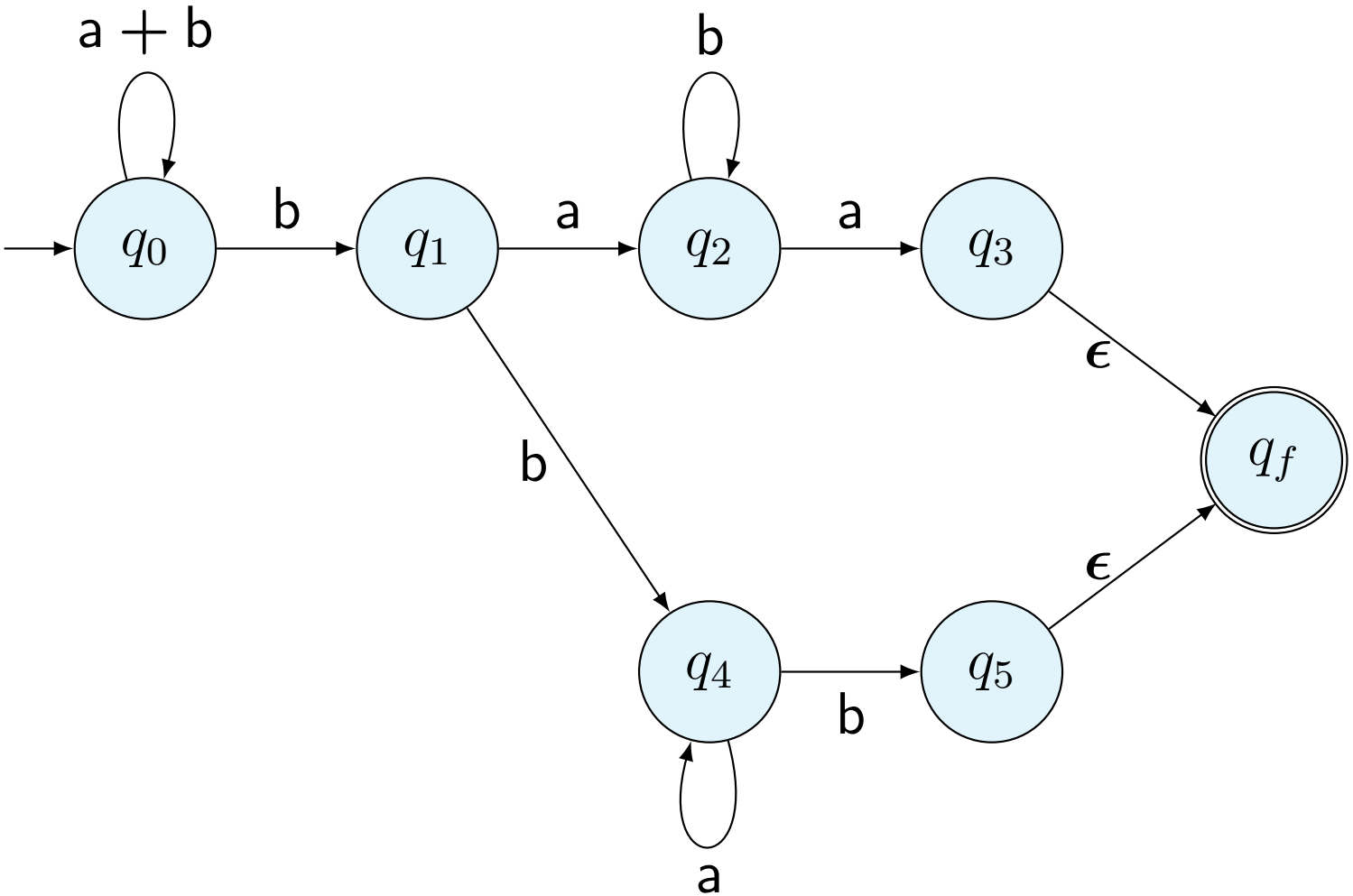


例 2

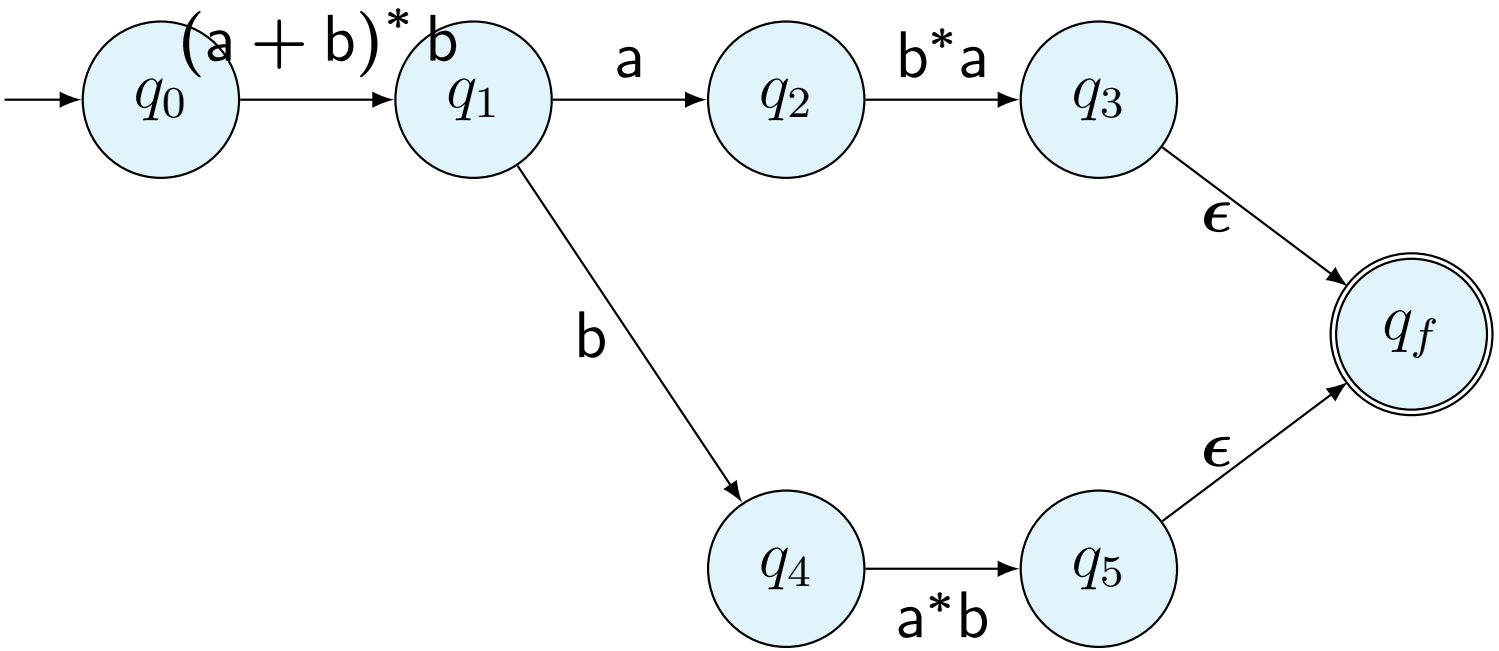


rule 1

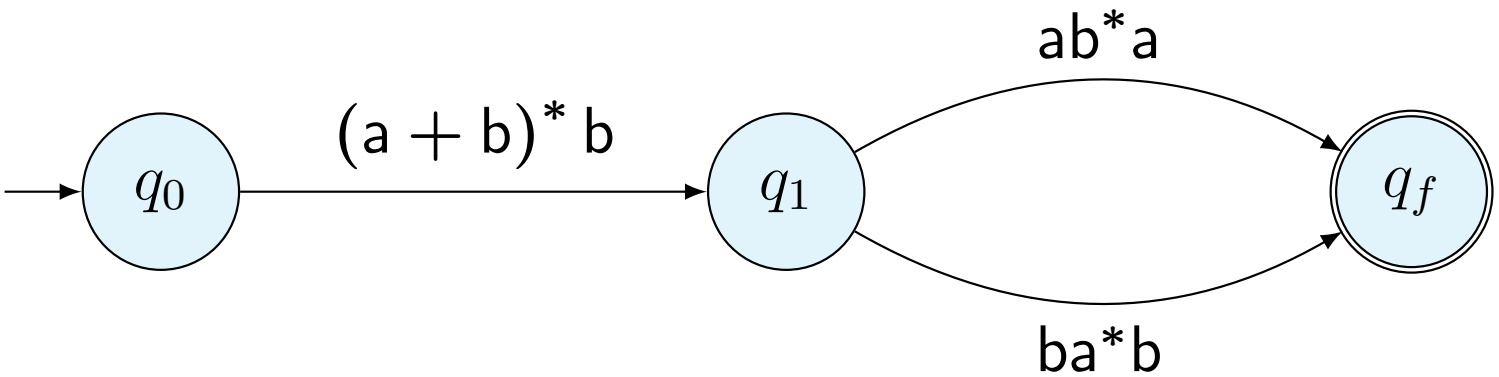
q_f を追加し、rule1 を適用



rule 2



rule 3



$$(\mathrm{a}+\mathrm{b})^{*}\,\mathrm{b}\,(\mathrm{ab}^{*}\mathrm{a}+\mathrm{ba}^{*}\mathrm{b})$$