有限オートマトンと正規表現

離散数学・オートマトン
2024 年後期
佐賀大学理工学部　只木進一

正規表現: Regular expressions

- 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
- 多くのプログラミング言語やテキストエディタで利用できる

例 1.1: ファイル名から日付部分を除く

```
import re
fileList=[
    '1.1 Introduction_20201010.txt',
    '1.2 SetAndMappings_20211013.txt',
    '1.3 Relations_20100401.txt'
]
#日付以外の部分を取り出す
p: re.Pattern[str] = re.compile(r'(.*)_\d{8}(\.txt)')
for f in fileList:
    m: re.Match[str] | None = p.match(f)
    if m:
        filename: str = m.group(1)+m.group(2)
        print(filename)
```

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

例 1.2: ドメイン名を変更

```
mail_address_list:list[str]=[
    'brown@example.com', 'page@hoge.com',
    't2013@example.edu', 'kate@hoge.com',
    's20235@foo.edu',    's19220@hoge.com'
]
pattern = r'(\S+)@(\S+)'
p: re.Pattern[str] = re.compile(pattern)
for mail in mail_address_list:
    m: re.Match[str] | None = p.match(mail)
    if m:
        domain = m.group(2)
        if domain == 'hoge.com':
            new_mail = m.group(1)+'@hoge.com'
            print(new_mail)

```

Python での正規表現: ごく一部

.	任意の文字と一致
\d	数字と一致
\D	数字以外と一致
\s	スペースやタブ、改行などの空白文字と一致
\S	空白文字以外と一致
*	0 回以上の繰り返し
+	1 回以上の繰り返し

正規表現の定義: 基礎

- $\mathrm{a} \in \Sigma$ に対して **a** は正規表現であり、その言語は $\{\mathrm{a}\}$ 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
  - 要素を持たない言語

正規表現の定義: 再帰

- $\boldsymbol{\alpha}$ と $\boldsymbol{\beta}$ が、言語 L_α 及び L_β をそれぞれ表す正規表現のとき
  - $\boldsymbol{\alpha} + \boldsymbol{\beta}$ は $L_\alpha \cup L_\beta$ を表す正規表現
  - $\boldsymbol{\alpha\beta}$ は $L_\alpha L_\beta$(連接:concatenation) を表す正規表現

$$L_\alpha L_\beta = \{uv | u \in L_\alpha, v \in L_\beta\} \tag{1.1}$$

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

$$L_\alpha^0 = \{\epsilon\}, L_\alpha^1 = L_\alpha, L_\alpha^{k+1} = L_\alpha L_\alpha^k \tag{1.2}$$

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

例 1.3: $\mathrm{a}, \mathrm{b} \in \Sigma$

- **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

例 1.4:

$$\mathbf{(0+1)1(0+1)}$$

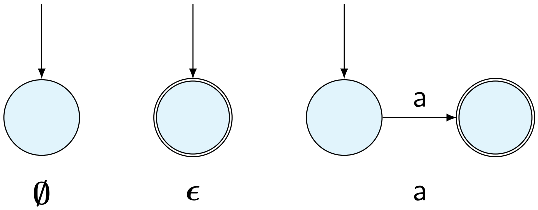
$$\mathbf{0(0+1)^{*}0}$$

正規表現と有限オートマトン Regular Expressions and Finite State Automata

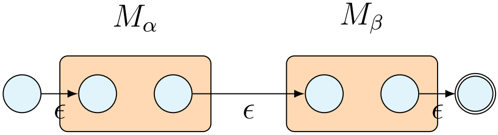
- 任意の正規表現を受理言語とする有限オートマトンを構成することができる
- 任意の有限オートマトンの受理言語を表す正規表現を構成することができる
- つまり、有限オートマトンの受理言語は正規表現で表される

正規表現から FA へ: 基礎的表現

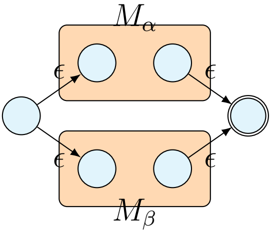
- 正規表現の構成を順を追って FA で表現
- 基礎:a, b $\in \Sigma$



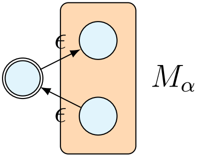
和、連接、Kleene 閉包



$\boldsymbol{\alpha\beta}$

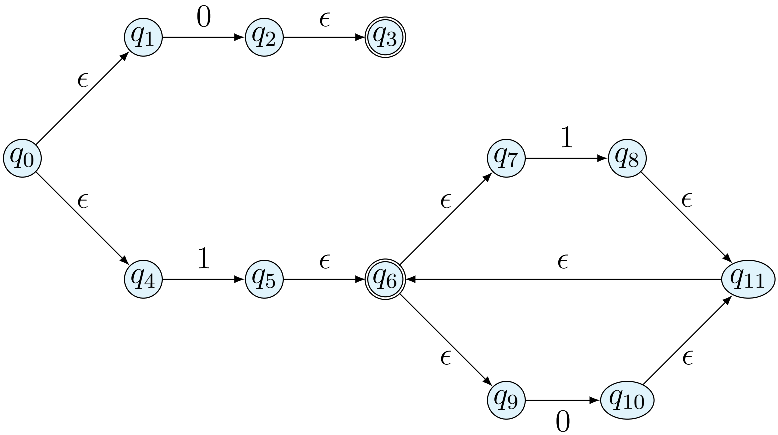


$\boldsymbol{\alpha} + \boldsymbol{\beta}$

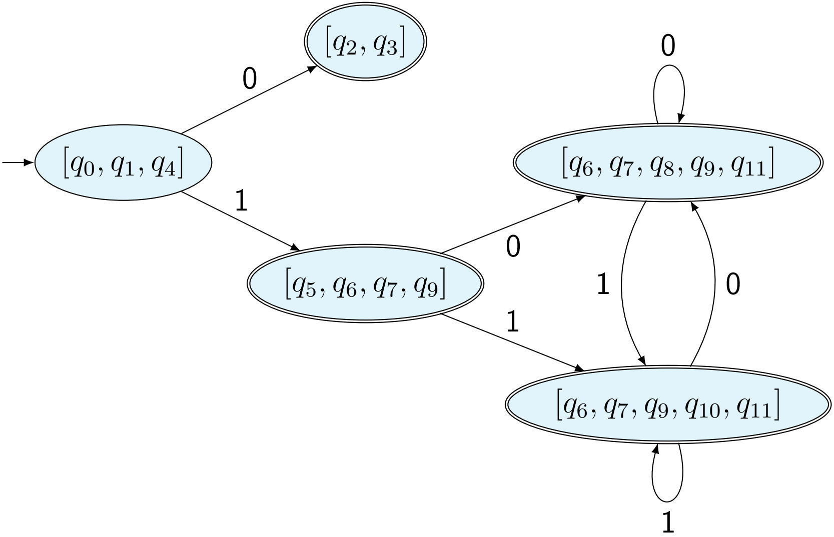


$\boldsymbol{\alpha}^*$

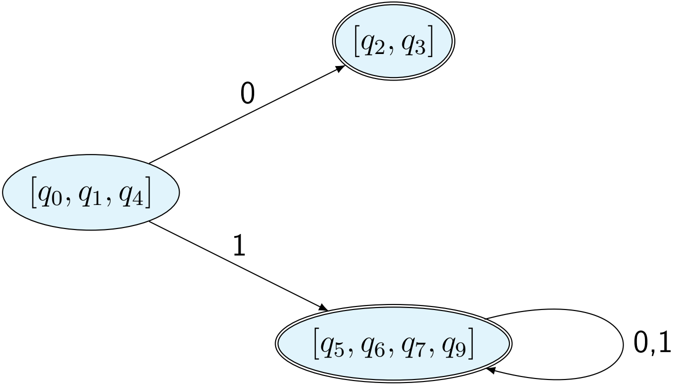
例 3.1: $\mathbf{0} + \mathbf{1}\,(\mathbf{0} + \mathbf{1})^*$



DFA へ変換

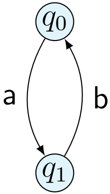


DFA を最小化

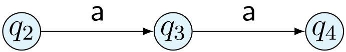


例 3.2: $(\mathbf{ab})^* \, \mathbf{aa} \, (\mathbf{a}+\mathbf{b})^*$

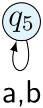
$(\mathrm{ab})^*$

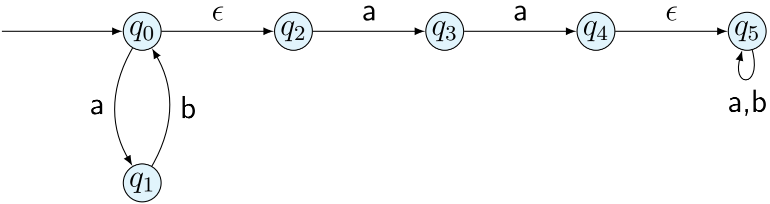


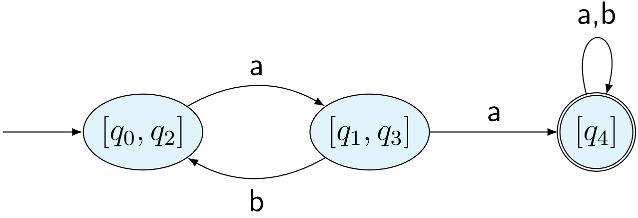
$\mathbf{aa}$



$(\mathbf{a}+\mathbf{b})^*$

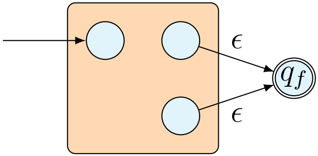



q_0
q_1
q_2
q_3
q_4
q_5
a
b
ϵ
a
a
ϵ
a,b

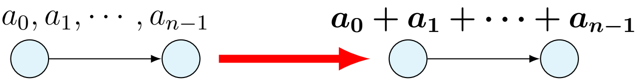

a,b
a
a
$[q_0, q_2]$
$[q_1, q_3]$
$[q_4]$
b

有限オートマトンの受理言語を正規表現で構成

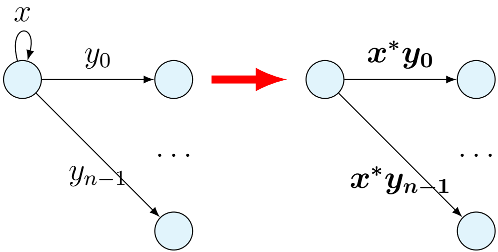
- Step1: 新たに一つの終状態 q_f を追加し、それのみが終状態とする



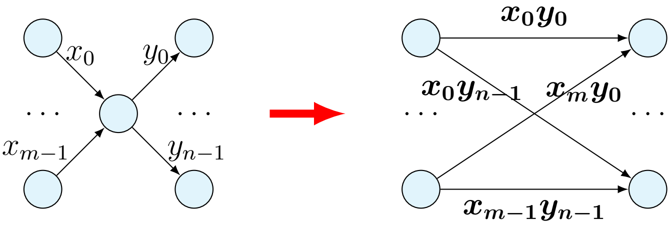
- step2: rule1,2,3 の順に適用する
- rule1: 同じ状態遷移を引き起こす入力に対して、正規表現の和を対応つける



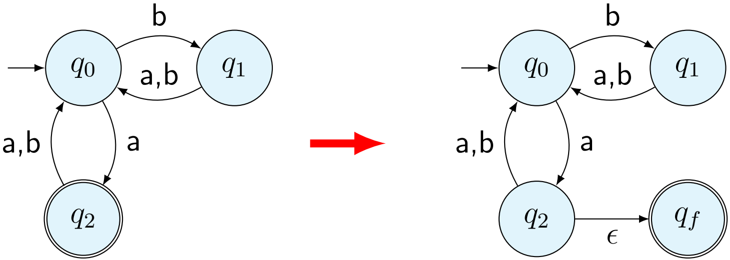
- rule2: ループの遷移を次の遷移の前に連接する



- rule3: 連続する遷移を、途中の状態を削除して連接とする

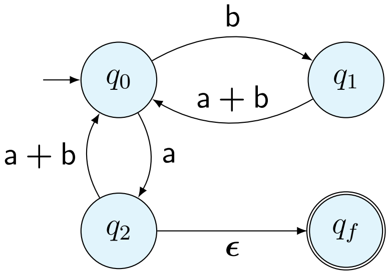


例 4.1:



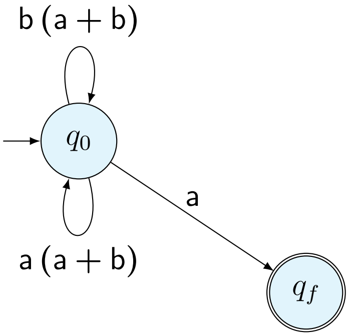
rule 1

同じ遷移を起こす入力を正規表現の和に変換



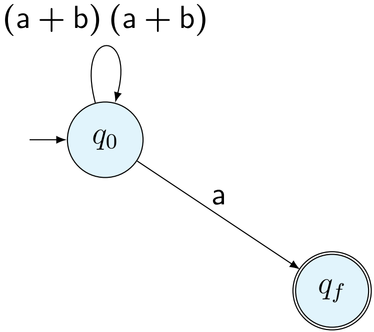
rule 3

- 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



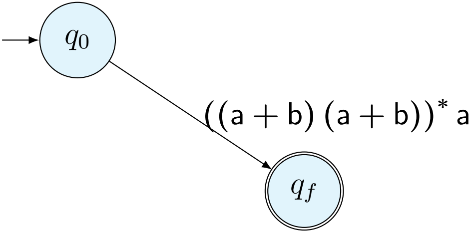
rule 1: 2回目

- q_0 の2つのループの表現の和を作る

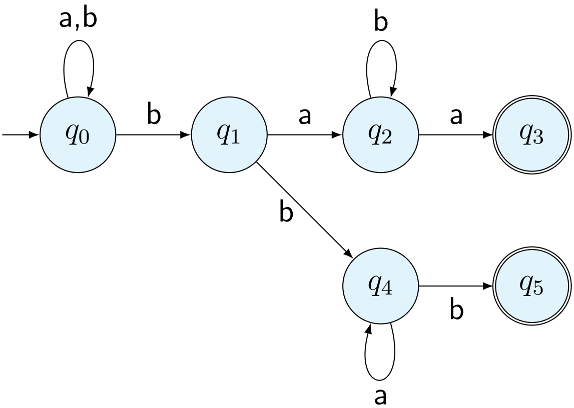


rule 2

- q_0 の２つのループの表現の和を作る

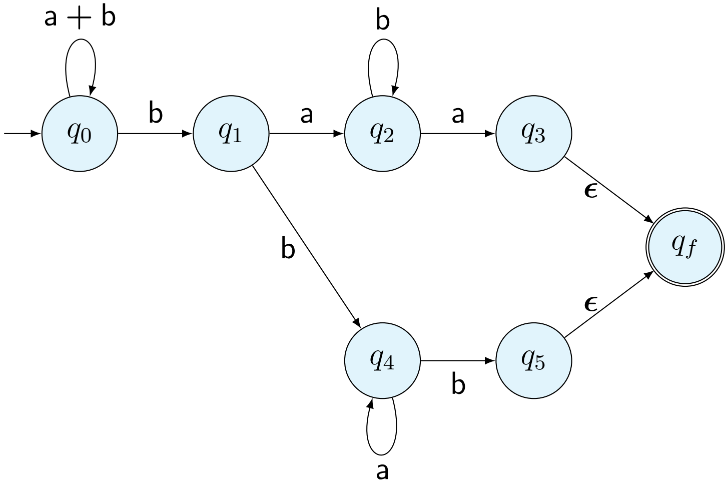


例 4.2:

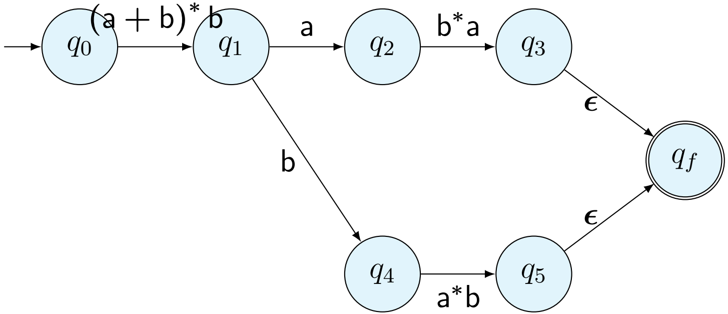


rule 1

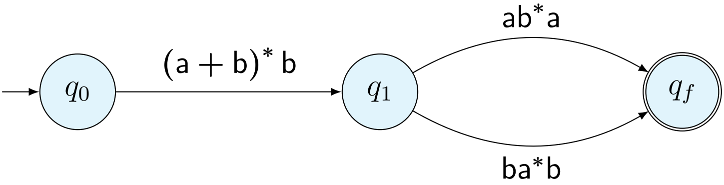
q_f を追加し、rule1 を適用



rule 2



rule 3



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

正規表現に対する反復補題: Pumping Lemma

- L を正規言語とする
- L に対して、ある定数 n が存在し、$|w| \geq n$ となる任意の $w \in L$ に対し、以下のように $w = xyz$ と分解できる。
  - $y \neq \epsilon$
  - $|xy| \leq n$
  - $\forall k \geq 0,\ xy^k z \in L$

反復補題の証明

- L を正規言語とし、対応する DFA を M とする。
  - $L = L(M)$
  - M の状態数を n とする

$$w = a_1 a_2 \cdots a_m \in L,\ (m \geq n,\ a_i \in \Sigma)$$
$$p_i = \delta(q_0, a_1 a_2 \cdots a_i),\ (p_0 = q_0)$$

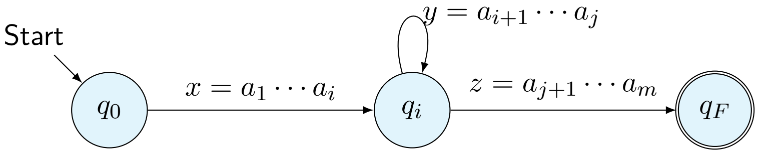
- $p_0, p_1, \cdots, p_n$ には少なくとも一つの状態が 2 回以上現れる

$$p_i = p_j,\ (0 \leq i < j \leq n)$$

- $w = xyz$ と分解する

$$x = a_1 a_2 \cdots a_i, \quad y = a_{i+1} a_{i+2} \cdots a_j, \quad z = a_{j+1} a_{j+2} \cdots a_m$$
$$|x| = i \geq 0, \quad |y| = j - i > 0, \quad |z| = m - j \geq 0$$

- $xy^k z \in L \ (0 \leq k)$



反復補題の応用

$L_{01} = \{0^m 1^m | m \geq 0\}$ は正規言語でない

- L_{01} が正規言語であると仮定
- 反復補題に現れる n に対して、$w = 0^n 1^n$ を考える
- $w = xyz$、$|xy| \leq n$ より、$y = 0^k$ $(1 \leq k \leq n)$

$$xy^0 z = xz = 0^{n-k} 1^n \notin L_{01}$$

- L_{01} は正規言語でない