

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

佐賀大学工学部 只木進一

- 1 正規表現: Regular expressions
- 2 正規表現と有限オートマトン: Regex and FA
- 3 正規表現から DFA へ: From Regex to DFA
- 4 有限オートマトンの受理言語を正規表現で構成: From FA to Regex
- 5 正規表現に対する反復補題

正規表現: Regular expressions

- 文字列の探索や置換で利用: Used for searching and replacing strings
- 柔軟にパターンを記述: Flexible pattern description
- 例: Examples
 - “000” の繰り返しを含む: Contains repetitions of “000”
 - 数字が偶数個連続する: Contains an even number of digits
 - 指定した文字列の後ろに数字が付いているファイル名: File names with numbers following a specified string
- 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: Available in many programming languages and text editors

例 1.1: ファイル名から日付部分を除く:

Remove date from file name

```
1 import re
2 fileList=[
3     '1.1 Introduction_20201010.txt',
4     '1.2 SetAndMappings_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.Pattern[str] = re.compile(r'(.*)_d{8}(\.txt)')
9 for f in fileList:
10     m: re.Match[str] | None = p.match(f)
11     if m:
12         filename: str = m.group(1)+m.group(2)
13         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: Change domain name

```
1 mail_address_list:list[str]=[
2     'brown@example.com', 'page@hoge.com',
3     't2013@example.edu', 'kate@hoge.com',
4     's20235@foo.edu',     's19220@hoge.com'
5 ]
6 pattern = r'(\S+)@(\S+)'
7 p: re.Pattern[str] = re.compile(pattern)
8 for mail in mail_address_list:
9     m: re.Match[str] | None = p.match(mail)
10    if m:
11        domain = m.group(2)
12        if domain == 'hoge.com':
13            new_mail = m.group(1)+'@hoge.hoge.com'
14            print(new_mail)
15
```

Python での正規表現: ごく一部

Regular expressions in Python: A very small part

- . 改行以外の任意の文字と一致: Matches any character except a newline
- \d 数字と一致: Matches any digit
- \D 数字以外と一致: Matches any non-digit
- \s スペースやタブ、改行などの空白文字と一致: Matches any whitespace character
- \S 空白文字以外と一致: Matches any non-whitespace character
- * 0 回以上の繰り返し: 0 or more repetitions
- + 1 回以上の繰り返し: 1 or more repetitions

正規表現の定義: 基礎

Defining Regular Expressions: Fundamentals

- $a \in \Sigma$ に対して a は正規表現であり、その言語は $\{a\}$ である:
 a is a regular expression for $\{a\}$
 - 一文字からなる言語: a language consisting of a single character
- ϵ は正規表現であり、その言語は $\{\epsilon\}$ である:
 ϵ is a regular expression for $\{\epsilon\}$
 - 長さゼロの文字列からなる言語: a language consisting of a string of length zero
- \emptyset は正規表現であり、その言語は \emptyset である:
 \emptyset is a regular expression for \emptyset
 - 要素を持たない言語: a language with no elements

正規表現の定義: 再帰

Defining Regular Expressions: Recursion

- α と β が、言語 L_α 及び L_β をそれぞれ表す正規表現のとき
Assume α and β are regular expressions representing languages L_α and L_β respectively
 - $\alpha + \beta$ は $L_\alpha \cup L_\beta$ を表す正規表現: $\alpha + \beta$ is a regular expression representing $L_\alpha \cup L_\beta$
 - $\alpha\beta$ は $L_\alpha L_\beta$ (連接: concatenation) を表す正規表現: $\alpha\beta$ is a regular expression representing $L_\alpha L_\beta$

$$L_\alpha L_\beta = \{uv \mid u \in L_\alpha, v \in L_\beta\} \quad (1.1)$$

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

$$L_\alpha^0 = \{\epsilon\}, L_\alpha^1 = L_\alpha, L_\alpha^{k+1} = L_\alpha L_\alpha^k \quad (1.2)$$

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

例 1.3: $a, b \in \Sigma$

- **a** は言語 $\{a\}$ を表す: **a** represents the language $\{a\}$
- **b** は言語 $\{b\}$ を表す: **b** represents the language $\{b\}$
- **a + b** は言語 $\{a, b\}$ を表す: **a + b** represents the language $\{a, b\}$
- **ab** は言語 $\{ab\}$ を表す: **ab** represents the language $\{ab\}$
- **a(a + b)b** は言語 $\{aab, abb\}$ を表す: **a(a + b)b** represents the language $\{aab, abb\}$
- **(a + 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: **(a + b)*** represents the language consisting of all strings of zero or more characters from a and b

例 1.4:

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

正規表現と有限オートマトン

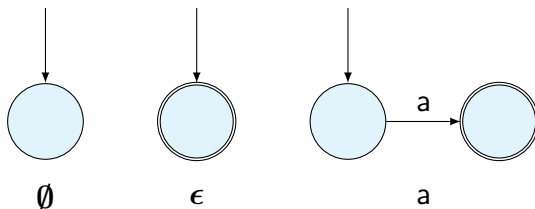
Regular Expressions and Finite State Automata

- 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: A finite automaton can be constructed for accepting the language of any regular expression
- 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: A regular expression can be constructed for representing the accepting language of any finite automaton
- 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: In other words, the accepting language of a finite automaton is represented by a regular expression

正規表現から FA へ: 基礎的表現

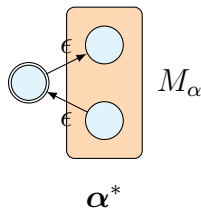
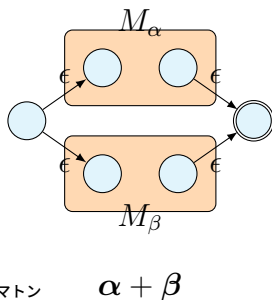
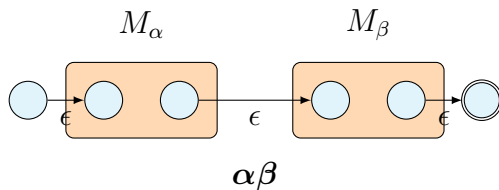
From Regex to FA: Basic Representations

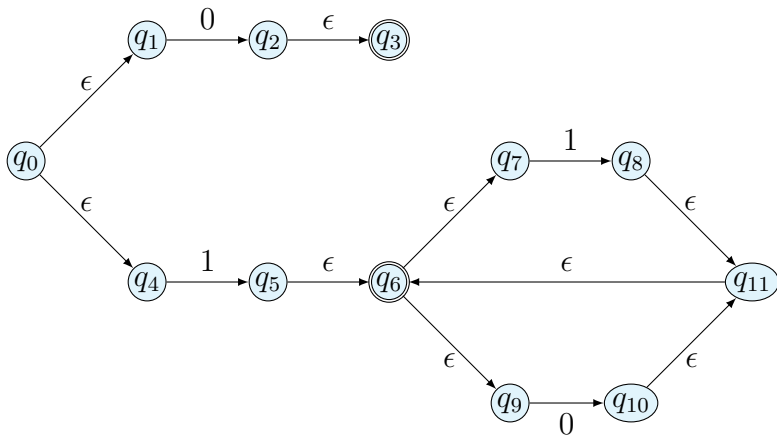
- 正規表現の構成を順を追って FA で表現: Constructing regular expressions in order to represent them in finite automata
- 基礎: Fundamentals: $a, b \in \Sigma$

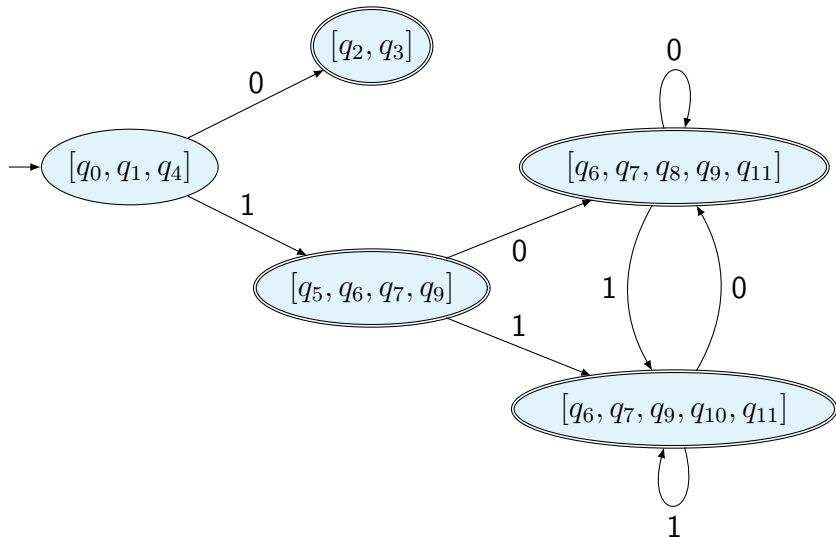


和、連接、Kleene 閉包

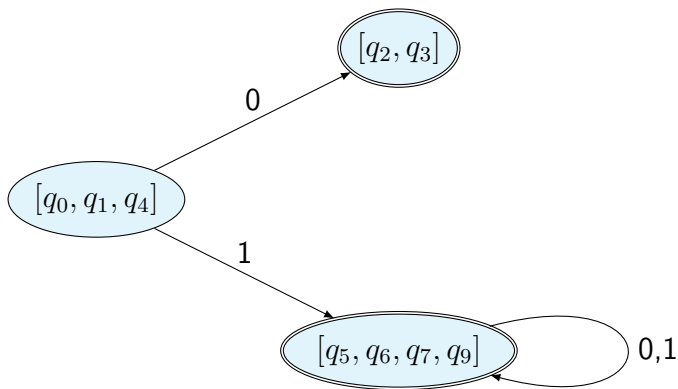
Union, Concatenation, Kleene Closure

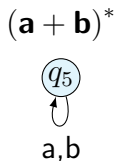
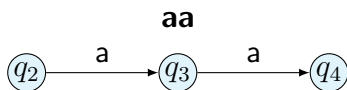
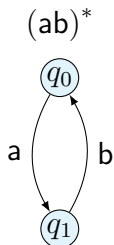


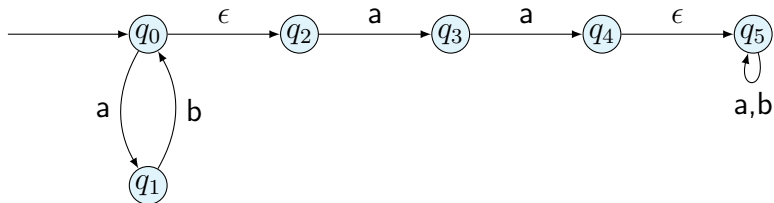
例 3.1: $0 + 1(0 + 1)^*$ 

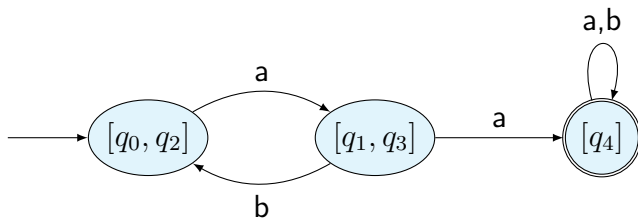
DFA \wedge 変換: NFA to DFA

DFA を最小化: Minimizing DFA



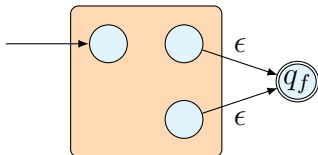
例 3.2: $(ab)^* aa (a + b)^*$ 



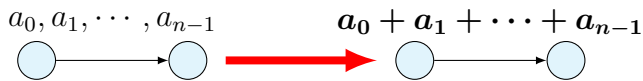


有限オートマトンの受理言語を 正規表現で構成: From FA to Regex

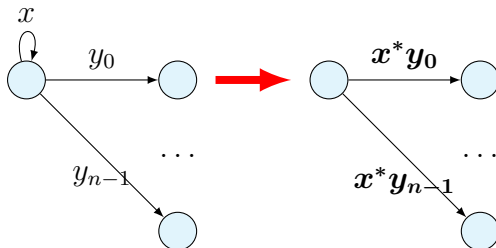
- Step1: 新たに一つの終状態 q_f を追加し、そのみが終状態とする: Add a new final state q_f and make it the only final state



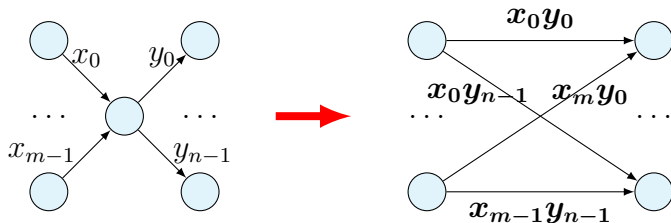
- step2: rule1,2,3 の順に適用する: Apply rule1,2,3 in order
- rule1: 同じ状態遷移を引き起こす入力に対して、正規表現の和を対応つける: Correspond a regular expression union for inputs causing the same state transition



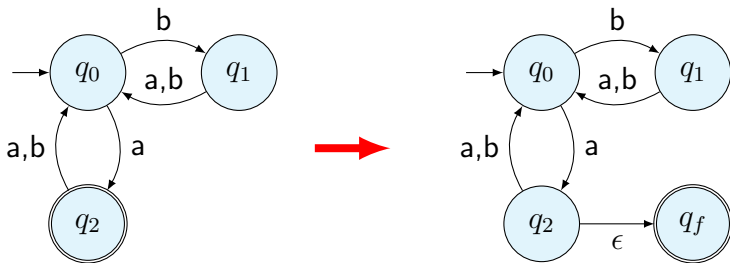
- rule2: ループの遷移を次の遷移の前に接続する: Concatenate the transition of the loop before the next transition



- rule3: 連続する遷移を、途中の状態を削除して接続とする:
Concatenate the consecutive transitions by removing intermediate states

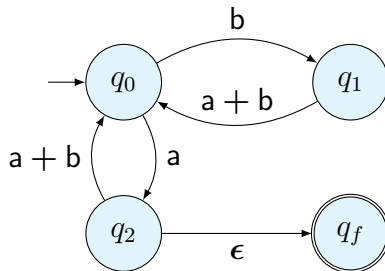


例 4.1:



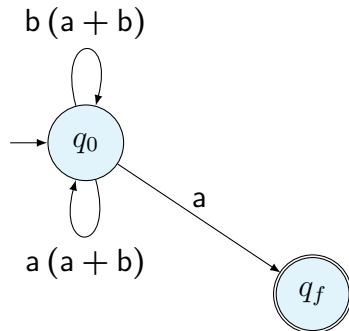
rule 1

同じ遷移を起こす入力を正規表現の和に変換: Strings that cause the same transition are combined with a plus sign.



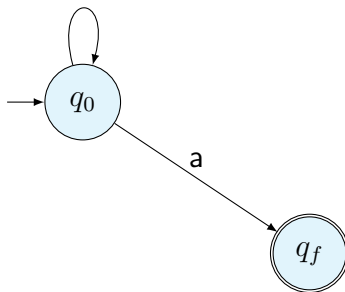
rule 3

- rule 2 に相当するループが: Loops that are equivalent to rule 2
 - q_0 から q_1 を経て、 q_0 へ戻る経路をループに: Loop from q_0 to q_1 and back to q_0
 - 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: Loop from q_0 to q_2 and back to q_0 and keep the transition to q_f



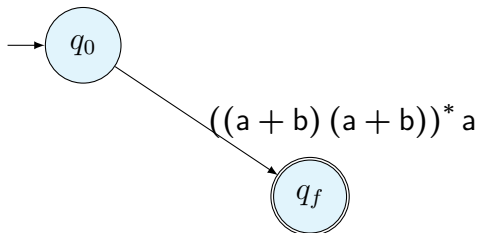
rule 1: 2回目

- q_0 の2つのループの表現の和を作る: Combine the representation of the two loops of q_0

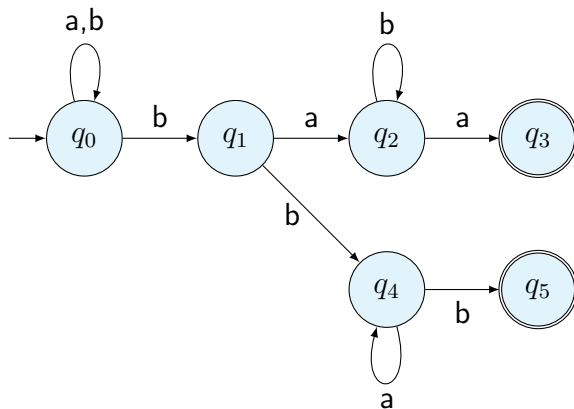
 $(a + b)(a + b)$ 

rule 2

- q_0 の2つのループの表現の和を作る: Combine the representation of the two loops of q_0

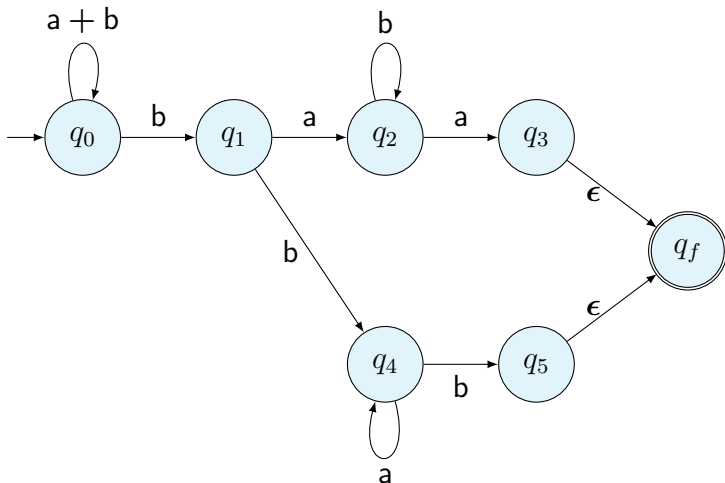


例 4.2:

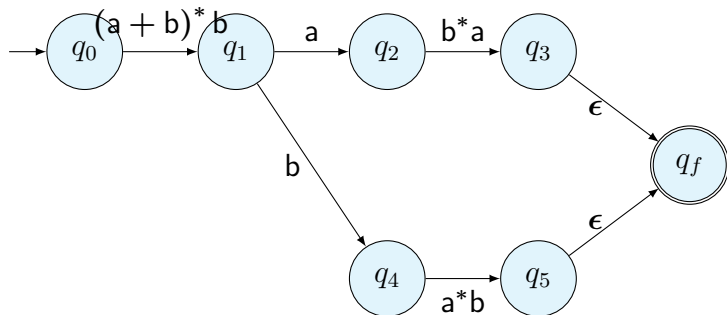


rule 1

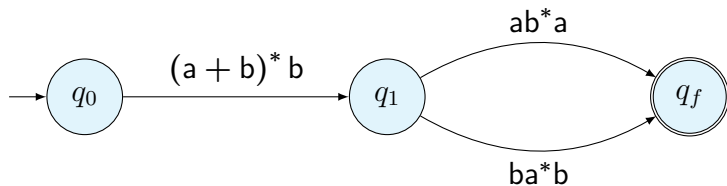
q_f を追加し、rule1 を適用



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: Let L be a regular language
- 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。: For any $w \in L$ with $|w| \geq n$, there exists a decomposition $w = xyz$ such that
 - $y \neq \epsilon$
 - $|xy| \leq n$
 - $\forall k \geq 0, xy^kz \in L$

反復補題の証明: Proof of Pumping Lemma

- L を正規言語とし、対応する DFA を M とする。: Let L be a regular language and M be the corresponding DFA
 - $L = L(M)$
 - M の状態数を n とする: Let n be the number of states in M

$$w = a_1 a_2 \cdots a_m \in L, \quad (m \geq n, a_i \in \Sigma)$$

$$p_i = \delta(q_0, a_1 a_2 \cdots a_i), \quad (p_0 = q_0)$$

- 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: At least one state appears twice in p_0, p_1, \cdots, p_n

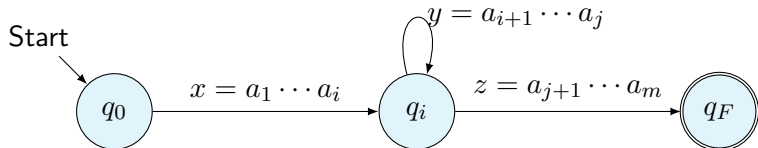
$$p_i = p_j, \quad (0 \leq i < j \leq n)$$

- $w = xyz$ と分解する: Decompose $w = xyz$

$$x = a_1 a_2 \cdots a_i, \quad y = a_{i+1} a_{i+2} \cdots a_j, \quad z = a_{j+1} a_{j+2} \cdots a_m$$

$$|x| = i \geq 0, \quad |y| = j - i > 0, \quad |z| = m - j \geq 0$$

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



反復補題の応用: Applications of Pumping Lemma

$L_{01} = \{0^m 1^m \mid m \geq 0\}$ は正規言語でない: $L_{01} = \{0^m 1^m \mid m \geq 0\}$ is not a regular language

- L_{01} が正規言語であると仮定: Assume L_{01} is a regular language
- 反復補題に現れる n に対して、 $w = 0^n 1^n$ を考える: Consider $w = 0^n 1^n$ for the n in the pumping lemma
- $w = xyz$ 、 $|xy| \leq n$ より、 $y = 0^k$ ($1 \leq k \leq n$)

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

- L_{01} は正規言語でない: L_{01} is not a regular language