オートマトンと言語理論

水谷 正大

2020 年 7 月 22 日 version 2.3

目次

第 1 章

文字列学

1.1　文字列

有限個の記号集合からなるアルファベット (alphabet)Σ の要素 $s_1, s_2, \ldots, s_n$ を並べた $s_1 s_2 \ldots s_n$ を文字列または語 (word) とする。語 w の長さ (length) を $|w|$ で表す。長さ n の語全体を

$$\Sigma^n = \{s_1 s_2 \ldots s_n \,|\, s_i \in \Sigma, i = 1, \ldots, n\}$$

と記す。語 $x = a_1 a_2 \ldots a_m$ と $y = b_1 b_2 \ldots b_n$ の連接 (concatenation) を $a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$ として xy と書く。このとき

$$|xy| = |x| + |y|$$

が成り立つ。

長さ 0 の文字列を空語 (empty word) といい ε を記す。このとき

$$\Sigma^0 = \{\varepsilon\}$$

である。Σ 上の有限長の語全体の集合を Σ^* で表わそう（Σ 上の **Kleene** 閉包ということもある)。

$$\Sigma^* = \bigcup_{k=0} \Sigma^k = \{\varepsilon\} \cup \Sigma \cup \Sigma^2 \cup \cdots \cup \Sigma^n \cup \ldots$$

語 w が語 x の接頭辞 (prefix) であるとは、ある語 $y \in \Sigma^*$ が存在して $x = wy$ であるときで $w \sqsubset x$ と記す。とくに $|y| > 0$ のとき、w を真の接頭辞 (proper prefix) という(煩雑さをさけるために記号 $\sqsubset$ と $\sqsubseteq$ を区別して用いない)。また、語 w が語 x の接尾辞

(suffix) であるとは、ある語 $z \in \Sigma^*$ が存在して $x = zw$ であるときで $w \sqsupset x$ と記す。とくに $|z| > 0$ のとき、w を**真の接尾辞** (proper suffix) という（煩雑さをさけるために記号 $\sqsupset$ と $\sqsupseteq$ を区別して用いない)。

性質 1.1 接頭辞および接尾辞については次が成立する。

(1) 任意の語 x について、$\varepsilon \sqsubset x$ かつ $\varepsilon \sqsupset x$.
(2) $w \sqsubset x$ または $w \sqsupset x$ のとき、$|w| \leqq |x|$.

$$x \sqsubset y \to ax \sqsubset ay, \quad x \sqsupset y \to xa \sqsupset ya.$$

(3) 関係 $\sqsubset$ および $\sqsupset$ は推移的である。

$$\begin{aligned} &x \sqsubset y \text{ かつ } y \sqsubset z \quad \to \quad x \sqsubset z \\ &z \sqsupset y \text{ かつ } y \sqsupset x \quad \to \quad z \sqsubset x. \end{aligned}$$

補題 1.1 語 x, y および z が $x \sqsupset z$ および $y \sqsupset z$ であるとき、つぎが成立する。

(1) $|x| \leqq |y|$ のとき、$x \sqsupset y$,
(2) $|x| \geqq |y|$ のとき、$y \sqsupset x$,
(3) $|x| = |y|$ のとき、$x = y$.

第 2 章

有限オートマトン

有限オートマトン (finite automaton) を **FA** と略記する。FA には以下でみるように、決定性有限オートマトン（Deterministic Finite Automaton, DFA と略記)、非決定性有限オートマトン（Non-deterministic Finite Automaton, NFA と略記)、また **ε-動作する NFA**（ε-NFA と略記）がある。

実は DFA NFA ε-NFA、さらには節 3.1 の正規表現はどれも同等である（図 2.1)。定理 2.2 で NFA から DFA が構成できることを、定理 2.3 で ε-NFA から NFA が構成できることを示す。また、節 3.5 の定理 3.3 で正規表現を受理する ε-NFA か構成できること、定理 3.4 で DFA から正規表現を書き下すことができることが示される。

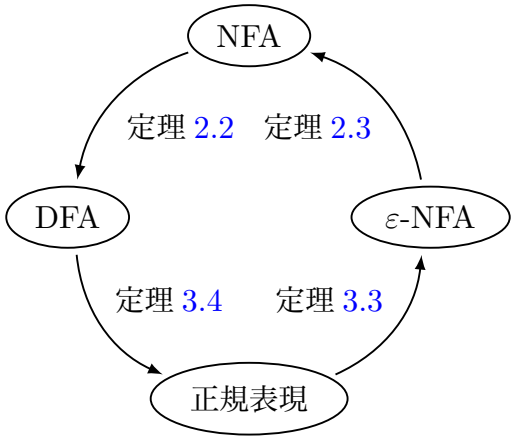


図 2.1　DFA, NFA, ε-NFA および正規表現の 4 つはどれも同等である

2.1　決定性有限オートマトン (DFA)

決定有限オートマトン (Determinisitic Dinite Automaton) M とは図 2.2 が示すように、有限個の内部状態 (states) を有する機械で、ヘッド (head) が入力文字列 $a_1a_2\ldots a_n$ を先頭から 1 文字づつ読み取るごとにその時の内部状態に応じて次の状態が定まる状態遷移機構を持つ。入力文字列を読み切った際の最終状態が受理集合のどれかの状態に到達していれば入力文字列を受理 (accept) し、そうでなければ受理しないという文字列の識別 (recoginition) を行うことができる。

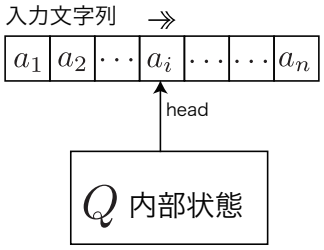


図 2.2　有限オートマトン M。ヘッドが入力文字列 $a_1a_2\ldots a_n$ を 1 文字づつ読み取るごとにその時の内部状態に応じて状態遷移する。

決定有限オートマトンを単に有限オートマトンと称し、しばしば **DFA** と略記する。形式的には次のように DFA を定義する。

定義 2.1 (決定有限オートマトン) 5 つ組 $M = (Q, \Sigma, \delta, q_0, F)$ で定まる状態遷移機械 M を決定有限オートマトン (DFA: Deterministic Dinite Automaton) という。Q は有限集合で、その要素を M の状態とし、$q_0 \in Q$ を初期状態 (initila state)、$F \subseteq Q$ が受理状態の集合である。δ は状態 p にあるとき文字 $a \in \Sigma$ を読んだときに状態 $q = \delta(p, a)$ への遷移を定める状態遷移関数 $\delta : Q \times \Sigma \to Q$ である。状態と入力記号の対が定まれば、次の状態が一意に定まるという意味で '決定' という。

δ を次のように再帰的に $\delta : Q \times \Sigma^*$ 上の関数

$$\delta(p, \varepsilon) = p \qquad p \in Q,$$
$$\delta(p, ax) = \delta(\delta(p, a), x) \qquad p \in Q, a \in \Sigma, x \in \Sigma^*$$

に拡張できる。M は最初に初期状態 q_0 にあり、アルファベット Σ 上の入力文字列 $x = a_1a_2\ldots a_n \in \Sigma^*$ を先頭から読み込みながら δ によって状態遷移を行い、最終状態

$\delta(q_0, x)$ が F に属したときに入力文字列を受理 (accept) する。空語 ε が受理される必要十分条件は初期状態 $q_0 \in F$ である。

表 2.1 は状態 p が入力 $a \in \Sigma$ に応じて q へと遷移する状態遷移表の一部で、DFA の状態遷移関数が定まる。

δ	a
p	q

表 2.1　状態 p が入力 $a \in \Sigma$ に応じて q へと遷移する様子を表す状態遷移表の一部。

DFA を、その状態を丸で囲んで、初期状態を矢印で指定し、受理状態を二重丸で囲んで図示する方法がしばしば用いられる。図 2.3 は、同じく DFA において状態 p が入力 $a \in \Sigma$ に応じて q へと遷移する様子を図示している。

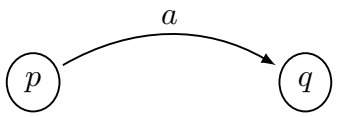


図 2.3　有限決定オートマトンの状態遷移図。状態 p から入力 $a \in \Sigma$ に応じて q へと遷移する様子

定義 2.2 DFA $M = (Q, \Sigma, \delta, q_0, F)$ によって受理される Σ 上の語全体

$$L(M) = \left\{ w \in \Sigma^* \,|\, \delta(q_0, w) \in F \right\}$$

を M によって受理される言語 $L(M)$ という。言語 $L \subseteq \Sigma^*$ に対して、$L = L(M)$ である決定有限オートマトン M が存在するとき、L を**正規言語** (regular language) という。

演習 2.1 正規言語 L が与えられたとき、L を受理する機械は一意に定まるだろうか。また、DFA M が与えられたとき、M で受理される正規言語をどのように表せばよいだろうか。

例 2.1 アルファベット $\Sigma = \{0, 1\}$、内部状態集合 $Q = \{q_0, q_1\}$ のとき、q_0 を初期状態かつ受理状態 $F = \{q_0\}$ とし、

$$\delta(q_0, 0) = q_0, \qquad \delta(q_0, 1) = q_1,$$
$$\delta(q_1, 0) = q_1, \qquad \delta(q_1, 1) = q_0$$

で決定有限オートマトン $M_1 = (Q, \Sigma, \delta, q_0, F = \{q_0\})$ を定める。このとき、M は図 2.4 のように図示できる。その状態遷移関数は表 2.2 で与えられる。記号 $\rightarrow$ は初期状態を、F は受理状態を表している。

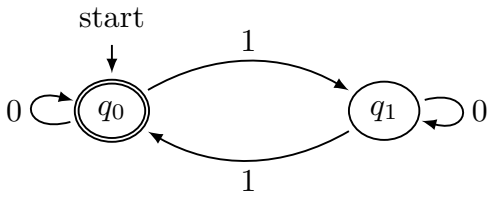


図 2.4　0,1 列のパリティを識別する機械 M_1

	δ	0	1
$\rightarrow, F$	q_0	q_0	q_1
	q_1	q_1	q_0

表 2.2　機械 M_1 の状態遷移表

M_1 に文字列 '010110' を入力してみよう。拡張した状態遷移関数を使うと、次のように最終状態として q_1 に到達することがわかる。

$$\begin{aligned}\delta(q_0, 010110) &= \delta(q_0, 10110) = \delta(q_1, 0110) \\ &= \delta(q_1, 110) = \delta(q_0, 10) = \delta(q_1, 0) \\ &= q_1\end{aligned}$$

演習 2.2 例 2.1 の機械 M_1 が受理する $\Sigma = \{0, 1\}$ 上の正規言語 $L(M_1)$ はどのようなものかを説明しなさい（正規表現（節 3.3）で表しなさい）。

演習 2.3 例 2.1 で定義される正規言語 $L(M_1)$ の補集合 $(L(M_1))^c = \Sigma^* - L(M_1)$ を受理する機械を構成しなさい。

演習 2.4 図 2.5 で図示される DFA の状態遷移表を書き、機械の動作を述べなさい。q_2 はどのような状態かを説明しなさい。この DFA が受理する言語はどのようなものかを説明しなさい（正規表現（節 3.3）で表しなさい）。

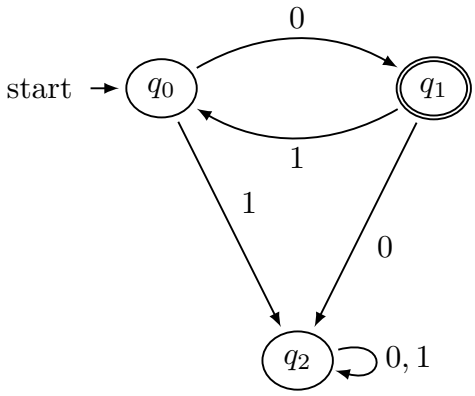


図 2.5　3 状態機械の例

演習 2.5 $\Sigma = \{a, b\}$ 上の次の言語を受理する FA を構成して図示しなさい。

(1) $L_1 = \{a^k \mid k$ は 3 の倍数（0 を含む）$\}$.
(2) $L_2 = \{a, b\}^* - \{a\}\{a\}^*\{b\}\{b\}^*$.
(3) $L_3 = \{a^j b^k \mid j$ は奇数、k は 3 の倍数 (0 を含む)$\}$.

演習 2.6 $\Sigma = \{a, b\}$ 上の次の言語 $L = \{a^n b^n \mid n \geqq 1\}$ を考える（L は a が 1 つ以上並んだあとに同じ数だけ b が並んだ語全体からなる言語)。このとき、図 2.6 で表される機械 M は言語 L の任意の語 $x \in L$ を受理することを説明しなさい。

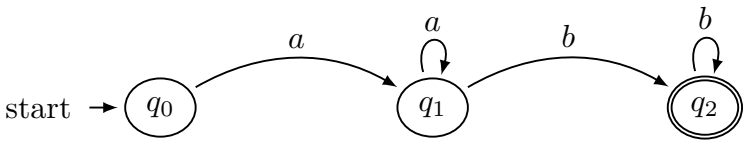


図 2.6　言語 $L = \{a^n b^n \mid n \geqq 1\}$ の任意の語 $x \in L$ を受理する FA M。

演習 2.7 図 2.6 で表される機械 M は言語 $L = \{a^n b^n \mid n \geqq 1\}$ に属する語以外の語もまた受理することを説明しなさい。言語 L に属する語「だけ」を受理する（つまり言語 L を受理する）有限状態機械を構成することができるだろうか？（実は、言語 L は正規言語でなく、文脈自由言語のクラスに属する。）　ヒント) 言語 L を受理するために必要な要件とは何かを考えてみる。

2.1.1　数論に関係する DFA

次のような言語 L_{3mul}

$$\{x \in \{0,1\}^* \mid x \text{ は } 3 \text{ の倍数の } 2 \text{ 進数表示} = [3n]_2, n \in \mathbb{N}\}$$

を考えよう。ただし、空語 $\varepsilon \in L_{3mul}$ は数 0 を表す（0 の 2 進表示）と考えておく。次の表 2.3 は L_{3mul} に属する語の一部を示している。

2 進表示	10 進表示
ε	0
0	0
11	3
110	6
1001	9
1100	12
1111	15
$\vdots$	$\vdots$

表 2.3　3 の倍数 $3n \in \mathbb{N}$ の 2 進数表示を語とする言語 L_{3mul}

言語 L_{3mul} を受理する機械 M_{3mul} を考えよう。結論をいうと表 2.4 がその状態遷移表、図 2.7 が機械 M_{3mul} である。

	δ	0	1
$\to, F$	q_0	q_0	q_1
	q_1	q_2	q_0
	q_2	q_1	q_2

表 2.4　3 の倍数 $3n, n \in \mathbb{N}$ の 2 進表現 L_{3mul} を受理する DFA の状態遷移表

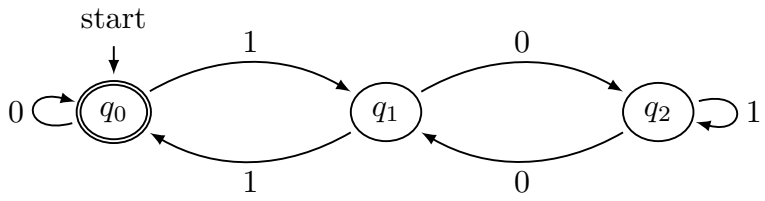


図 2.7　3 の倍数 $3n, n \in \mathbb{N}$ の 2 進表現 L_{3mul} を受理する DFA

図 2.7 の DFA が 3 の倍数 $3n(n \in \mathbb{N})$ の 2 進表現 L_{3mul} を受理することを証明することが残っている。文字列 $\boldsymbol{x} \in \{0,1\}^*$ に対して、記法 $[\boldsymbol{x}]_2$ は 2 進表示 $\boldsymbol{x}$ を持つ数

$$[x_{n-1} \dots x_0]_2 = x_{n-1}2^{n-1} + \cdots + x_0$$

を表すとする。図 2.7 を眺めて直ぐに気づくことは、以下の事実である。

入力	最終到達状態
$[\boldsymbol{x}]_2 \equiv 0 \mod 3$	$\delta(q_0, \boldsymbol{x}) = q_0$
$[\boldsymbol{x}]_2 \equiv 1 \mod 3$	$\delta(q_0, \boldsymbol{x}) = q_1$
$[\boldsymbol{x}]_2 \equiv 2 \mod 3$	$\delta(q_0, \boldsymbol{x}) = q_2$

これだけでは、機械 M_{3mul} の構成の見通しが立ちにくい。数の 2 進数表示に立ち戻って考えればよい。まず、次の等式が成立する

$$\begin{aligned}
&[\boldsymbol{x}0]_2 = 2[\boldsymbol{x}]_2 + 0 \\
&[\boldsymbol{x}1]_2 = 2[\boldsymbol{x}]_2 + 1 \\
&[\boldsymbol{x}i]_2 = 2[\boldsymbol{x}]_2 + i, \quad i \in \{0,1\}
\end{aligned}$$

したがって、この機械機械 M_{3mul} においては、状態 $q_k \in \{q_0, q_1, q_2\}$ に対する入力 $i \in \{0,1\}$ について

$$\delta(q_k, i) = q_{(2k+i) \bmod 3}$$

である。この状態遷移関数は入力 $\boldsymbol{x} \in \{0,1\}^*$ の長さに関して次のように拡張できる。$x = \varepsilon$ について

$$\begin{aligned}
\delta(q_0, \varepsilon) &= 0 \qquad (\text{定義より}) \\
&= [\varepsilon]_2 \qquad ([\varepsilon]_2 = 0 \text{ から}) \\
&= [\varepsilon]_{2 \bmod 3}
\end{aligned}$$

とするとき、帰納的に次のように定義すればよい。

$$\begin{aligned}
\delta(q_0, \boldsymbol{x}i) &= \delta(\delta(q_0, \boldsymbol{x}), i) \qquad i \in \{0,1\} \\
&= \delta(q_{[\boldsymbol{x}]_2 \bmod 3}, i) \qquad \text{帰納法の仮定から} \\
&= q_{(2([\boldsymbol{x}]_2 \bmod 3)+i) \bmod 2} \qquad \text{定義から} \\
&= q_{(2[\boldsymbol{x}]_2+i) \bmod 3} \qquad \text{初等整数論より} \\
&= q_{[\boldsymbol{x}i]_2 \bmod 3}
\end{aligned}$$

2.1.2 De Bruijn 系列

円盤の周囲に 0 と 1 からなる記号が配置されていて、長さ n のすべての 01-列が円周上の記号の連続部分として 1 回ずつ現れるようにしたい（回転ドラム問題)。そのような盤配置は存在するのか、またあるとすれば何通り有るだろうか。1946 年に De Bruijn が考えた問題である。

定義 2.3 (De Bruijn 系列) 長さ 2^n の 0-1 列 $B_n = b_1, b_2, \ldots, b_{2^n}, (b_k \in \{0,1\})$ が n-次の De Bruijn 系列とは、長さ n の全ての 01-列 $(a_1, \ldots, a_n) \in \{0,1\}^n$ が B_n 内に唯 1 回だけ連続部分列として表れる、すなわち、唯一の k があって

$$b_k = a_1, b_{k+1} = a_2, \ldots, b_{k+n-1} = a_n$$

となることである。ここで、添字番号は巡回的（b_{2^n} のあとに b_1 が続く）とする。

例 2.2 3 次の de Bruijn 列として $B_3 = 00010111$ は長さ 3 の 0-1 列が全て登場。
4 次の de Bruijn 列として $B_4 = 0100110101111000$ には長さ 4 の 0-1 列が全て登場。

演習 2.8 2 次の de Bruin 列を求めなさい。

定義 2.4 (n 次の De Bruijn グラフ) n 次の De Bruijn グラフ $\mathrm{DB}_n = (V, E)$. 長さ $n-1$ の 01-列からなる 2^{n-1} 個の頂点 $V = \{[x_1 x_2 \ldots x_{n-1}] \,|\, x)i \in \{0,1\}^{n-1}\}$ の間のラベル付き有向グラフで、頂点 $[x_1 \ldots x_{n-1}]$ から出る 2 頂点に向かう有向辺、すなわち、頂点 $[x_2 \ldots x_{n-1}0]$ に向かう有向辺 $x_1 \ldots x_{n-1}0$ と頂点 $[x_2 \ldots v_{n-1}1]$ に向かう有向辺 $x_1 \ldots x_{n-1}1$ からなる（図 2.8)。

De Bruijn グラフの各頂点には出入する辺がそれぞれ 2 本で各頂点の出入り次数は偶数の Euler グラフであることに注意しよう。

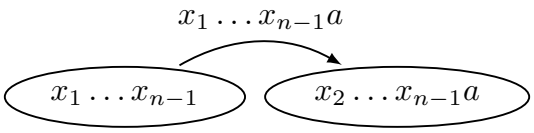


図 2.8　De Bruijn グラフ DB_n の一部。頂点 $[x_1 \ldots x_{n-1}]$ から頂点 $[x_2 \ldots x_{n-1}a]$ ($a \in \{0,1\}$) に向かう長さ n の有向辺 $ex_1 \ldots x_{n-1}a$ は De Bruijn 系列をなす。

定理 2.1 n 次の De Bruijn 系列 B_n はすべての n について存在し、De Bruijn 系列 B_n の個数は $2^{2^{n-1}}$ である（De Bruijn グラフの一筆書きの個数)。

演習 2.9 2 次、3 次、4 次の De Bruijn グラフを書き、その Euler 一筆書きから得られるラベル列（図 2.8 の有向辺ラベル末尾の記号 $a \in \{0,1\}$ からなる列）として De Bruijn 系列を書き出しなさい。

証明　De Bruijn 列が Euler グラフである De Bruijn グラフを一筆書したときに辺ラベル列として得られることを示すことで、定理の前半を示すことができる（自ら試みてみなさい)。

定理の後半。Euler グラフの一筆書き（出発点に戻る）ことから出発点に違いは 1 つのサイクルとして同一視すると、一筆書き可能なグラフに対する Euler 巡回のパターン数が De Bruijn 列の個数となる。

Euler グラフに一筆書き巡回数はグラフ内の注目する頂点に関する広域木 (spanning tree) に関係していることが知られている。その個数はグラフの Laplacian 行列の固有値から求められる[*1] [*2] [*3]。そのような立場でグラフ（のたとえば隣接行列）の性質を研究する分野が代数的グラフ理論である。 ■

2.2　非決定性有限オートマトン (NFA)

入力アルファベット Σ 、機械の有限個の内部状態の集合 Q、受理状態集合 $F \subseteq Q$ は DFA と同じとして、5 つ組 $M = (Q, \Sigma, \delta_{NFA}, Q_0, F)$ で定まる非決定有限オートマトン (NFA: Non-deteministic Finite Automaton) を定義することができる。$q_0 \in Q$ を初期

[*1] C.Godsil and G. Royle,*Algebraic Graph Theory*, Springer GTM 207(2001), p.281
[*2] N. Biggs, *Algebraic Graph Theory*, Cambridge Math. Library(1993), p.38.
[*3] 竹中淑子『線形代数的グラフ理論』培風館 (1989) , p.55.

状態集合、図 2.9 のように任意の $(q,a) \in Q \times \Sigma$ に対して、次の状態を定める状態遷移関数 $\delta_{NFA} : Q \times \Sigma \to 2^Q$ の値が集合値、つまり状態が Q の部分集合 $\delta(q,a) \in 2^Q$ として定まる。ここで 2^Q は $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ の部分集合全体

$$2^Q = \Big\{\phi, \{q_0\}, \ldots, \{q_{n-1}\}, \{q_0, a_1\}, \ldots, \{q_{n-2}, q_{n-1}\}, \{q_0, q_1, q_2\}, \ldots, \{q_{n-3}, q_{n-2}, q_{n-1}\},$$
$$\ldots, \{q_1, q_2, \ldots, q_{n-1}\}, \ldots, \{q_0, q_1, \ldots, q_{n-1}\}\Big\}, \qquad |2^Q| = 2^{|Q|}.$$

NFA の ‘状態’ は DFA と違って Q 内の唯一つの状態だけ決定されず、複数の状態となり得る（同時並行進行を許す）ように Q の部分集合値が定まると考える。この意味で状態遷移機械として次の状態が一意に定まらないことをもって ‘非決定’ というのである（確率論的な意味合いは全くない)。

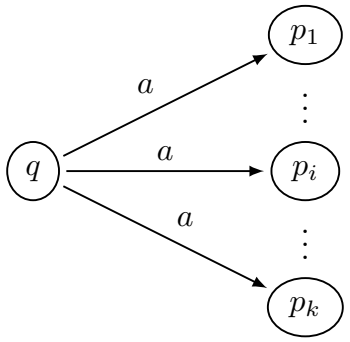


図 2.9　非決定有限オートマトン NFA では、状態 q で文字 $a \in \Sigma$ が入力されると、同時に状態 $q_1, \ldots, q_k$ に遷移する $\delta(q,a) = \{p_1, \ldots, p_m\} \in 2^Q$

定義 2.5 $M = (Q, \Sigma, \delta_{NFA}, Q_0, F)$ の 5 つ組で定まる Σ 上の状態遷移機械 M を**非決定有限オートマトン** (NFA) という。ここで、Q は有限個の内部状態の集合、Σ は入力アルファベット、$\delta_{NFA} : Q \times \Sigma \to 2^Q$ は状態遷移関数、$Q_0 \subseteq Q$ は初期状態の集合、$F \subseteq Q$ は受理状態である。初期状態集合が 1 つの要素からなる場合 $Q_0 = \{q_0\}$、記法 Q_0 の代わりに初期状態 q_0 と記すことがある。

図 2.9 のように、NFA では状態 $q \in Q$ に文字 $a \in \Sigma$ が入力されると複数の状態集合に遷移する：

$$\delta(q,a) = \{p_1, \ldots, p_k\}, \qquad p_i \in Q.$$

また、状態 $q \in Q$ に文字列 $ab \in \Sigma^*$ が入力されると、$a \in \Sigma$ の入力により遷移した状態

集合のそれぞれ状態に $b \in \Sigma$ の入力によって遷移した状態集合の和に遷移する：

$$\delta(q, ab) = \delta(\delta(q, a), b) = \delta(p_1, b) \cup \ldots \cup \delta(p_k, a).$$

このような Σ 上の文字列入力に対する NFA の状態遷移の挙動を記述できるように、状態遷移関数を次のように拡張しよう。

定義 2.6 (NFA の動作) NFA を定める 5 つの組 $M = (Q, \Sigma, \delta_{NFA}, q_0, F)$ の状態遷移関数 $\delta_{NFA} : Q \times \Sigma \to 2^Q$ を Σ 上の文字列に対して、次のように $\delta^*_{NFA} : Q \times \Sigma^* \to 2^Q$ として動作するように拡張する：

任意の状態 $p \in Q$ に対して

$$\delta^*_{NFA}(p, \varepsilon) = \{p\},$$

入力文字列 $x \in \Sigma^*$ と記号 $a \in \Sigma$ に対して

$$\delta^*(p, x) = \{q_1, \ldots, q_m\} \in 2^Q, \quad \delta_{NFA}(q_j, a) = Q_j \subseteq Q$$

であるとき

$$\begin{aligned} \delta^*_{NFA}(p, xa) &= \delta_{NFA}(q_1, a) \cup \ldots \cup \delta_{NFA}(q_m, a) \\ &= Q_1 \cup \ldots \cup Q_m. \end{aligned}$$

$a \in \Sigma$ に対して $\delta^*_{NFA}(p, a) = \delta_{NFA}(p, a)$ であることから δ^*_{NFA} と δ_{NFA} とを区別する必要がないことに注意しよう。これより、文字列 $x \in \Sigma^*$ の入力に対する状態集合 $\{q_1, \ldots, q_k\} \in 2^Q$ の遷移を定めるために、さらに次のように拡張して状態遷移関数 $\delta_{NFA} : 2^Q \times \Sigma^* \to 2^Q$ を定義する：

$$\delta_{NFA}(\{q_1, \ldots, q_k\}, x) = \delta_{NFA}(q_1, x) \cup \ldots \cup \delta_{NFA}(q_k, x).$$

定義 2.6 のように、Σ 上の NFA の状態遷移関数を $2^Q \times \Sigma^* \to 2^Q$ の関数に拡張したときでも、δ^*_{NFA} と δ_{NFA} と区別する必要がなく、以降では同じ記法 δ_{NFA} だけを使っても混乱は起こらない。ただし、定義 2.10（とその後に注意）にあるように、ε-動作付きの NFA においては、Σ 上の状態遷移関数 $\delta_{\varepsilon NFA}$ を文字列 Σ^* に拡張した $\delta^*_{\varepsilon NFA}$ は一般的に $\delta_{\varepsilon NFA}$ とは異なるので注意が必要である。

定義 2.7 (NFA の受理) NFA $(Q, \Sigma, \delta_{NFA}, q_0, F)$ において、語 $w \in \Sigma^*$ に対し、その到達状態集合のどれかが受理集合 F の要素

$$\delta(q_0, w) \cap F \neq \phi$$

であるとき、NFAM は語 w を受理する。このとき、w は受理状態に達するパス (path) を持つという。パスが定義されていない δ の遷移動作に達したとき、そのパスは廃棄され以降の動作は考えない。

Σ 上の NFA $M = (Q, \Sigma, \delta, q_0, F)$ によって受理される語の全体

$$L(M) = \left\{ w \in \Sigma^* \mid \delta(q_0, x) \cap F \neq \phi \right\}$$

を NFA M が受理する言語 $L(M)$ という。

例 2.3 $\Sigma = \{0, 1\}$ 上の NFA として、$Q = \{q_0, q_1\}$、初期集合 $Q_0 = \{q_0\}$、受理集合 $F = \{\boldsymbol{q}_1\}$ とする NFA $M_{N_1} = (Q, \Sigma, \delta_{N_1}, \{q_0\}, \{q_1\})$ の状態遷移が表 2.5 で与えられている。

	δ_{N_1}	0	1
初期	q_0	$\{q_0, q_1\}$	$\{q_1\}$
受理	q_1	ϕ	$\{q_0\}$

表 2.5　NFA の例 M_{N-1} : $\delta(q_1, 0)$ は定義されていない

表 2.5 の NFA M_{N_1} は図 2.10 のように図示される（定義されていない遷移 $\delta_{N_1}(q_1, 0)$ は慣例として表示しない)。初期状態集合 $Q_0 = \{q_0\}$、受理集合 $F = \{q_1\}$ であって、q_0 から入力 0 によって q_0 および q_1 に遷移することに注意しよう。

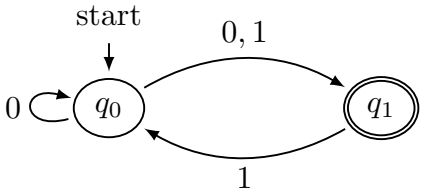


図 2.10　表 2.5 の状態遷移に従う NFA M_{N_1}.

文字列 0011001 の動作を考えよう。状態遷移表から、遷移動作が定義されずに廃棄されるパスを棄却しながら次のように最終的な状態集合 $\{q_0, q_1\}$ に到達する。

$$\begin{aligned}\delta_{N_1}(\{q_0\}, 0011001) &= \delta_{N_1}(\{q_0, q_1\}, 011001)\\ &= \delta_{N_1}(\{q_0, q_1\}, 11001) = \delta_{N_1}(\{q_0, q_1\}, 1001)\\ &= \delta_{N_1}(\{q_0, q_1\}, 001) = \delta_{N_1}(\{q_0, q_1\}, 01)\\ &= \delta_{N_1}(\{q_0, q_1\}, 1) = \{q_0, q_1\}\end{aligned}$$

$\{q_0, q_1\} \cap F \neq \phi$ より、文字列 0011001 を受理することがわかる。

例 **2.4** $\Sigma = \{0, 1\}$ 上の NFA M_{N_2} として、$Q = \{q_0, q_1, q_2\}$、初期集合 $Q_0 = \{q_0\}$、受理集合 $F = \{\boldsymbol{q_2}\}$ とする $M_{N_2} = (Q, \Sigma, \delta_{N_2}, \{q_0\}, \{q_1\})$ の状態遷移が表 2.6 で与えられている。

	δ_{N_2}	0	1
初期	q_0	$\{q_0\}$	$\{q_0, q_1\}$
	q_1	ϕ	$\{q_2\}$
受理	q_2	$\{q_2\}$	ϕ

表 2.6　NFA の例：遷移 $\delta(q_1, 0)$, $\delta(q_2, 1)$ は定義されていない

表 2.6 の NFA M_{N_2} は図 2.11 のように図示される（遷移 $\delta_{N_2}(q_1, 0)$, $\delta_{N_2}(q_2, 1)$ は定義されていない)。q_0 から入力 1 によって q_0 および q_1 に遷移することに注意しよう。

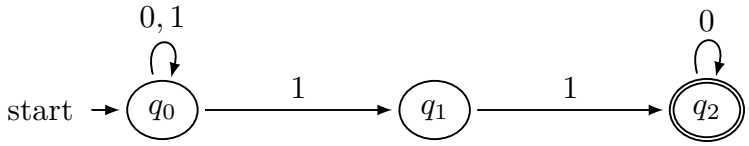


図 2.11　表 2.6 の状態遷移に従う NFA M_{N_2}.

NFA において、初期状態を状態集合 Q_0 と取ることもある。明らかに決定有限オートマトン DFA とは初期状態集合 Q_0 および状態遷移関数 δ に関して

$$\begin{aligned}&|Q_0| = 1\\ &|\delta_{NFA}(q, a)| = 1, \qquad q \in A, a \in \Sigma\end{aligned}$$

であるような非決定有限オートマトン NFA の特別な場合である。

演習 **2.10** 言語受理機械の能力を考えてみよう。入力アルファベット Σ を固定したとき、NFA が受理できる言語集合全体 $\mathrm{NFA}(\Sigma) = \{L(M) \mid M \text{ は } \Sigma \text{ 上の NFA}\}$ を考える。

NFA(Σ) は DFA が受理する正規言語集合全体 DFA(Σ) = $\{L(M) \mid M$ は Σ 上の DFA$\}$ よりも大きいだろうか。

$\mathrm{DFA}(\Sigma) \subsetneq \mathrm{NFA}(\Sigma)$　　$\subsetneq$ は真部分集合

非決定有限オートマトンの言語受理能力と決定有限オートマトンの受理能力については、次の定理 2.2 よりその能力には差がないことがわかる。

定理 2.2 言語 L が非決定有限オートマトン (NFA) $M = (Q, \Sigma, \delta_{NFA}, Q_0, F)$ で受理されるとする。このとき、L を受理する NFA と等価な決定性有限オートマトン (DFA) $M' = (Q, \Sigma, \delta_D, q_0', F')$ の 5 つ組を構成することができる。

証明　部分集合構成法による。初期状態 q_0 から文字列 $w \in \Sigma^*$ によって到達可能な状態集合全体は高々 Q のべき集合 2^Q であることに注意する。M' の状態遷移関数 $\delta_D : Q' \times \Sigma \to Q'$ を次のように構成する。$Q' = 2^Q$、$q_0' = \{q_0\}$, $2^{|Q|}$ 個ある 2^Q の各部分集合を $q_i'(i = 0, \ldots, 2^{|Q|} - 1)$ とし、入力 $a \in \Sigma$ に対する M' の状態遷移を以下のように定める：

$$\delta_D(p', a) = q', \quad p', q' \in 2^Q \qquad \text{iff } \delta_{NFA}(p', a) = q'.$$

また、M' の受理状態集合 F' を

$$F' = \{q' \in Q' \mid q' \cap F \neq \phi\}$$

で定義する。M' では Q の部分集合全体 2^Q の要素（Q の部分集合 $\{q_{i_n}, \ldots, q_{i_k}\}$ を 1 つの状態（$q_i' = [q_{i_n} \ldots q_{i_k}]$ と表すと都合がよい）と見なしているというのが部分集合構成法である。このとき、M' の受理集合 F' は、Q' の元（'集合'）で、M の受理状態 $q_f \in F$ を 1 つ以上含む '集合' の集合とみる。こうして定義した DFA $M' = (Q', \Sigma, \delta_D, q_0', F')$ が受理する Σ 上の文字列集合

$$L(M') = \{w' \in \Sigma^* \mid \delta_D(q_0', w') \in F'\}$$

を考える。構成の仕方から

$$\forall w \in L(M) \Rightarrow w \in L(M') \Leftrightarrow \forall w' \in L(M') \Rightarrow w' \in L(M).$$

したがって、$L(M) = L(M')$. ■

例 2.5 例 2.3 の NFA M_{N_1} と等価な DFA M_{D_1} を構成しよう。

演習 2.11 例 2.4 の NFA M_{N_2} と等価な DFA M_{D_2} を構成しなさい。

2.3 ε-動作付き NFA

2.3.1 ε-動作

いままでの決定有限オートマトン (DFA) または非決定有限オートマトン (NFA) においては、入力がなければ状態遷移はしない、つまり空語 ε が入力されても現在の状態は変わらずに同じ状態に留まる $\delta(p, \varepsilon) = p, \quad p \in Q$ としてきた。

無謀な暴走を許すかのようだが、空文字の入力によって（入力文字無しの）状態遷移を許す機械を考えることができる。空文字入力による状態遷移を **ε-動作**（入力無しの状態遷移動作）という。

図 2.12 は、状態 p が空語 ε によって（何も入力せずとも）q へと遷移する ϵ-動作付きの状態遷移の様子を示している。

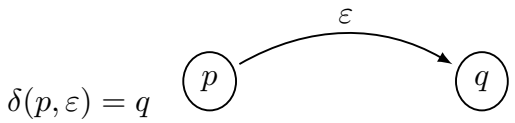


図 2.12　ϵ-動作付きの状態遷移（ε-動作）。状態 p から空入力 ε によって（何も入力せずとも）q へと遷移する様子 $\delta(p, \varepsilon) = q$. この例では一つの状態 q への遷移であるが、状態 p から同時に複数の状態 $q_1, \ldots, q_k$ に遷移してもよい。

定義 2.8 (ε-動作付きの NFA) 各状態 $q \in Q$ から空入力による遷移も許して、入力 $a \in \Sigma \cup \varepsilon$ によって移れる状態集合を定める状態遷移関数 $\delta_{\varepsilon NFA} : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ を持つ非決定有限機械を ε-NFA $M_\varepsilon = (Q, \Sigma_\varepsilon, \delta_{\varepsilon NFA}, q_0, F_\varepsilon)$ と呼ぶ。以下、$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ と記す。

例 2.6 ε-動作する有限オートマトン $M_{\varepsilon 1}$ の例を図 2.13 に示した。状態遷移関数 $\delta_{\varepsilon 1}$ は初期状態 q_0 からは記号 0 の入力によって q_0 自身に遷移する以外に、何も入力せずとも（空文字 ε の ‘入力’ によって）状態 q_1 に ϵ-動作する。また、状態 q_1 から何も入力せずとも（空文字 ε の ‘入力’ によって）状態 q_2 にも ϵ-動作する。

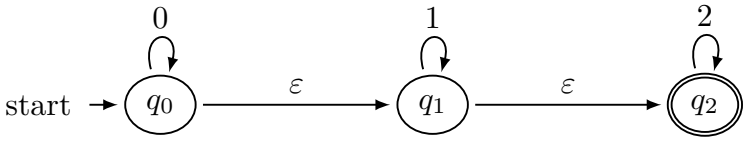


図 2.13　ε-動作のある状態遷移機械 $M_{\varepsilon 1}$

この機械の Σ_ε に関する状態遷移表は、空文字の入力 ε を追加して表 2.7 のように表わされる。

	$\delta_{\varepsilon 1}$	0	1	2	ε
初期	q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
	q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
受理	q_2	ϕ	ϕ	$\{q_2\}$	ϕ

表 2.7　ε-動作する図 2.13 の状態遷移表

演習 2.12 例 2.6 における図 2.13 または表 2.7 で与えられる ε-動作する非決定有限機械が受理する言語（文字列集合）は何かを記述しなさい。

例 2.7 $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $\Sigma = \{1\}$, $F = \{q_5\}$ のとき、次の図 2.14 で示される ε-動作付き NFA $(Q, \Sigma_\varepsilon, \delta_2, q_0, F)$ を考える。初期状態 q_0 にあって入力が 1 であるとき、以下を非決定的に動作する：

- 1 を読んで状態 q_1 に遷移
- 入力を読まずに状態 q_2 にスライドし、入力 1 を読んで状態 q_3 に遷移
- 入力を読まずに状態 q_2 にスライドし、さらに入力を読まずに状態 q_4 にスライドし, 入力 1 を読んで状態 q_5 に遷移
- 状態 q_1, q_3 は入力がなくとも ε-動作する。

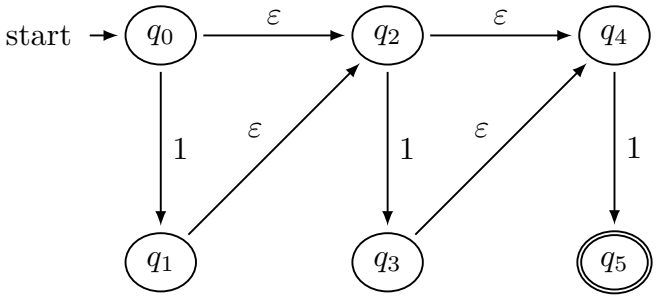


図 2.14　ε-NFA の例 $M_{\varepsilon 2}$.

演習 2.13 例 2.7 の M_{ε_2}(図 2.14) における Σ_ε に関する状態遷移関数 $\delta_{\varepsilon 2}$ の状態遷移表が次の表で与えられていることを確かめなさい

	$\delta_{\varepsilon 1}$	1	ε
初期	q_0	$\{q_1\}$	$\{q_2\}$
	q_1	ϕ	$\{q_2\}$
	q_2	$\{q_3\}$	$\{q_4\}$
	q_3	ϕ	$\{q_4\}$
	q_4	$\{q_5\}$	ϕ
受理	q_5	ϕ	ϕ

演習 2.14 図 2.14 の M_{ε_2} は状態 q_0 から入力 1 で遷移する状態集合が $\{1, 11, 111\}$ であることから、言語 $\{1, 11, 111\}$ を受理することを確かめなさい。

2.3.2　ε-閉包

定義 2.9 状態 $p \in Q$ から ε-動作だけで到達可能な状態集合 $\mathcal{E}(p)$ を **ε-閉包** と呼び、次で定義する。

$$\mathcal{E}(p) = \left\{ \text{状態 } p \text{ から } \varepsilon\text{-動作だけで遷移できる状態集合} \right\}.$$

$p \in \mathcal{E}(p)$ に注意する。ε-閉包は部分集合 $Q' \subseteq Q$ 上に拡張でき、部分集合 $Q' = \{q_1, \ldots, q_k\}$ の ε-閉包を

$$\mathcal{E}\big(\{q_1, \ldots, q_k\}\big) = \mathcal{E}(q_1) \cup \ldots \mathcal{E}(q_k)$$

と定める。

例 **2.8** 例 2.6 の図 2.13 において

$$\mathcal{E}(q_0) = \{q_0, q_1, q_2\},\ \mathcal{E}(q_1) = \{q_1, q_2\},\ \mathcal{E}(q_2) = \{q_2\}.$$

例 **2.9** 例 2.7 の図 2.14 において

$$\mathcal{E}(q_0) = \{q_0, q_2, q_4\}.$$

2.3.3 ε-NFA の状態遷移関数の拡張

節 2.3.1 で、ε-動作だけで到達可能な状態集合として ε-閉包を定義した。ε-NFA の動作として、状態 p から入力 $w \in \Sigma^*$ の道に沿って遷移できるすべての状態集合を $\delta^*_{\varepsilon NFA}(q, w)$ と表せるように、状態遷移関数 $\delta_{\varepsilon NFA}$ を $Q \times \Sigma^*$ から 2^Q への関数 $\delta^*_{\varepsilon NFA}$ として拡張する。

定義 2.10 (ε-NFA の状態遷移関数の拡張) ε-NFA $M_\varepsilon = (Q, \Sigma_\varepsilon, \delta_{\varepsilon NFA}, q_0, F_\varepsilon)$ において、状態遷移関数の拡張 $\delta^*_{\varepsilon NFA} : Q \times \Sigma^* \to 2^Q$ を次のように定義する：
0) 任意の状態 $p \in Q$ に対して

$$\delta^*_{\varepsilon NFA}(p, \varepsilon) = \mathcal{E}(p)$$

1) 入力記号列 $w \in \Sigma^*$ と記号 $a \in \Sigma$ に対して、

$$\delta^*_{\varepsilon NFA}(p, w) = \{q_1, \ldots, q_m\}, \quad q_k \in Q, \quad , \delta_{\varepsilon NFA}(q_i, a) = Q_i \subseteq Q$$

であるとき、

$$\delta^*_{\varepsilon NFA}(p, wa) = \mathcal{E}\Big(\delta_{\varepsilon NFA}(\delta^*_{\varepsilon NFA}(p, w), a)\Big) = \mathcal{E}\Big(\bigcup_{q_i \in \delta^*_{\varepsilon NFA}(p,w)} \delta_{\varepsilon NFA}(q_i, a)\Big)$$
$$= \mathcal{E}\Big(Q_1 \cup \ldots \cup Q_m\Big).$$

さらに、ε-NFA の状態遷移関数 $\delta_{\varepsilon NFA}$ とその拡張 $\delta^*_{\varepsilon NFA}$ による遷移を状態空間 2^Q から 2^Q へ一般化して次のように定める。

$$\delta_{\varepsilon NFA}(\{p_1, \ldots, p_k\}, a) = \delta_{\varepsilon NFA}(p_1, a) \cup \ldots \cup \delta_{\varepsilon NFA}(p_k, a),$$
$$\delta^*_{\varepsilon NFA}(\{p_1, \ldots, p_k\}, w) = \delta^*_{\varepsilon NFA}(p_1, w) \cup \ldots \cup \delta^*_{\varepsilon NFA}(p_k, w).$$

特に、状態 $p \in Q$ から 1 文字入力 $a \in \Sigma$ に対する $\delta_{\varepsilon NFA}$ と $\delta^*_{\varepsilon NFA}$ の挙動は次のようになる。

$$\delta^*_{\varepsilon NFA}(p,a) = \delta^*_{\varepsilon NFA}(p,\varepsilon a) = \mathcal{E}(\delta_{\varepsilon NFA}(\delta^*_{\varepsilon NFA}(p,\varepsilon),a) = \mathcal{E}(\delta_{\varepsilon NFA}(\mathcal{E}(p),a))$$
$$\neq \delta_{\varepsilon NFA}(p,a).$$

すなわち、$\delta^*_{\varepsilon NFA}(p,a)$ は q から $a \in \Sigma$ をラベルに持つ道 (途中に ε をラベルに持つ辺を含んでもよい) を通って到達できる状態の集合で、$\delta_{\varepsilon NFA}(p,a)$ は p からラベル a の辺で直接到達できる状態集合である。

このように、ε-動作のない NFA（定義 2.6）の状態遷移関数 δ_{NFA} の拡大 δ^*_{NFA} とは異なり、文字列には任意の位置に空文字 ε を挿入することができるため、ε-動作付き NFA では状態遷移関数 $\delta_{\varepsilon NFA}$ とその拡大 $\delta^*_{\varepsilon NFA}$ とは区別して考えねばならない（[10, p53、図 2.38] にわかり易い図が掲載されている)。

例 **2.10** 例 2.6 の図 2.13 において

$$\begin{aligned}
\delta^*_1(q_0,\varepsilon) &= \mathcal{E}(q_0) = \{q_0,q_1,q_2\},\\
\delta^*_1(q_0,0) &= \mathcal{E}(\delta_1(\delta^*_1(q_0,\varepsilon),0))\\
&= \mathcal{E}(\delta_1(\{q_0,q_1,q_2\},0))\\
&= \mathcal{E}(\delta_1(q_0,0)\cup\delta_1(q_1,0)\cup\delta_1(q_2,0))\\
&= \mathcal{E}(\{q_0\}\cup\phi\cup\phi)\\
&= \mathcal{E}(\{q_0\}) = \{q_0,q_1,q_2\}.
\end{aligned}$$

$$\begin{aligned}
&\delta^*_1(q_0,1) = \{q_1,q_2\}, \quad \delta^*_1(q_0,2) = \{q_2\},\\
&\delta^*_1(q_1,0) = \phi, \quad \delta^*_1(q_1,1) = \{q_1,q_2\}, \quad \delta^*_1(q_1,2) = \{q_2\},\\
&\delta^*_1(q_2,0) = \phi, \quad \delta^*_1(q_2,1) = \phi, \quad \delta^*_1(q_2,2) = \{q_2\},\\
\delta^*_1(q_0,12) &= \mathcal{E}(\delta_1(\delta^*_1(q_0,1),2))\\
&= \mathcal{E}(\delta_1(\{q_1,q_2\},2))\\
&= \mathcal{E}(\{q_2\}) = \{q_2\}.
\end{aligned}$$

演習 **2.15** 例 2.7 の図 2.14 与えられる ε-NFA において、$\delta^*_2(q_0,\varepsilon), \delta^*_2(q_0,1), \delta^*_2(q_1,\varepsilon), \delta^*_2(q_1,1), \delta^*_2(q_2,\varepsilon), \delta^*_2(q_2,1), \delta^*_2(q_3,\varepsilon), \delta^*_2(q_3,1), \delta^*_2(q_4,\varepsilon), \delta^*_2(q_4,1), \delta^*_2(q_5,\varepsilon), \delta^*_2(q_5,1)$ および、$\delta^*_2(q_0,11), \delta^*_2(q_2,11), \delta^*_2(q_4,11), \delta^*_2(q_0,1$ などを求めなさい。

定義 **2.11** (ε**-動作付き NFA の受理**) ε-動作付き NFA M_ε が文字列 $w \in \Sigma^*_\varepsilon$ を受理するとは、$\delta_{\varepsilon NFA}$ を ε-閉包を使って拡張した状態遷移関数 δ^*_ε について

$$\delta^*_{\varepsilon NFA}(q_0,w) \cap F_\varepsilon \neq \phi$$

のときである。この ε-NFA M_ε が受理する言語 $L(M_\varepsilon)$ を

$$L(M_\varepsilon) = \{w \in \Sigma_\varepsilon^* \,|\, \delta^*_{\varepsilon NFA}(q_0, w) \cap F_\varepsilon \neq \phi\}$$

で定義する。

以下、ε-NFA の Σ_ε 上の文字列上の状態遷移関数 $\delta_{\varepsilon NFA}$ の Σ_ε 上の動作は上記の定義 2.10 のように考える。このとき、誤解のない範囲で $\delta_{\varepsilon NFA}$ と $\delta^*_{\varepsilon NFA}$ と区別することなく、同一の表記 $\delta_{\varepsilon NFA}$ を使うことがある。

演習 2.16 言語受理機械の能力を考えてみよう。入力アルファベット Σ を固定したとき、ε-NFA が受理できる言語集合全体 ε-NFA$(\Sigma) = \{L(M_\varepsilon) \,|\, M_\varepsilon$ は Σ_ε 上の ε-NFA$\}$ を考える。ε-NFA(Σ) は NFA が受理する正規言語集合の全体 NFA$(\Sigma) = \{L(M) \,|\, M$ は Σ 上の NFA$\}$ よりも大きいのだろうか。

NFA$(\Sigma) \subsetneq \varepsilon$-NFA$(\Sigma)$　　$\subsetneq$ は真部分集合.

2.3.4 ε-動作を除去した等価な NFA

ε-NFA から ε-動作を除去して、同じ言語を受理する NFA を構成することができる。次の定理 2.3 から、ε-動作付き非決定有限オートマトンの言語受理能力は非決定有限オートマトンの受理能力と同等であることがわかる。

定理 2.3 言語 L が ε-NFA $M_\varepsilon = (Q, \Sigma_\varepsilon, \delta_{\varepsilon NFA}, q_0, F_\varepsilon)$ で受理されるとする。このとき、L を受理する M_ε と等価な ε 動作を持たない NFA $M' = (Q, \Sigma, \delta'_{NFA}, q_0, F'_{NFA})$ を構成することができる。

証明　M_ε の状態遷移関数 $\delta_{\varepsilon NFA} : Q \times \Sigma_\varepsilon \to 2^Q$ を定義 2.10 で Σ_ε^* 上に拡張した $\delta^*_{\varepsilon NFA} : Q \times \Sigma_\varepsilon^* \to 2^Q$ を使って、目的の NFA M' の $\delta'_{NFA} : Q \times \Sigma \to 2^Q$ を次のように定める。

$$\begin{aligned}\delta'_{NFA}(p, a) &= \delta^*_{\varepsilon NFA}(p, a), \quad p \in Q, a \in \Sigma \\ &= \mathcal{E}\Big(\bigcup_{q' \in \mathcal{E}(p)} \delta_{\varepsilon NFA}(q', a)\Big).\end{aligned}$$

また、M' の受理状態 F'_{NFA} を次にように定める。

$$F'_{NFA} = \begin{cases} F \cup \{q_0\} & \text{iff } \mathrm{cl}(q_0) \cap F_\varepsilon \neq \phi \\ F & \text{iff } \mathrm{cl}(q_0) \cap F_\varepsilon = \phi. \end{cases}$$

こうして定義した NFA $M' = (Q, \Sigma, \delta'_{NFA}, q_0, F'_{NFA})$ が受理する Σ 上の文字列集合

$$L(M') = \{w \in \Sigma^* \mid \delta'_{NFA}(q_0, w) \cap F_{NFA} \neq \phi\}$$

を考える。構成の仕方から

$$\forall w \in L(M_\varepsilon) \Rightarrow w \in L(M'_{NFA}) \Leftrightarrow \forall w' \in L(M'_{NFA}) \Rightarrow w' \in L(M_\varepsilon).$$

したがって、$L(M_\varepsilon) = L(M'_{NFA})$.

■

例 2.11 例 2.6 で取り上げた ε-NFA $M_{\varepsilon 1}$ を、定理 2.3 に従って構成した、ε-動作を除去した等価な非決定有限オートマトン $M_{NFA1} = (Q = \{q_0, q_1, q_2\}, \Sigma = \{0, 1, 2\}, \delta'_{NFA1}, q_0, F'_1\})$ を図 2.15 およびその状態遷移表を表 2.8 に示した。

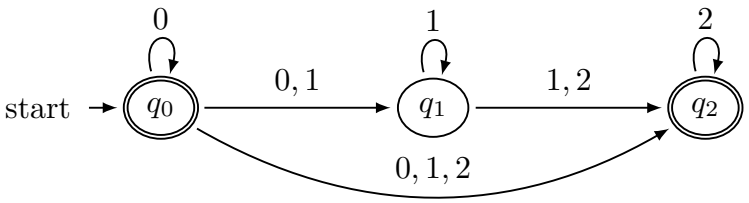


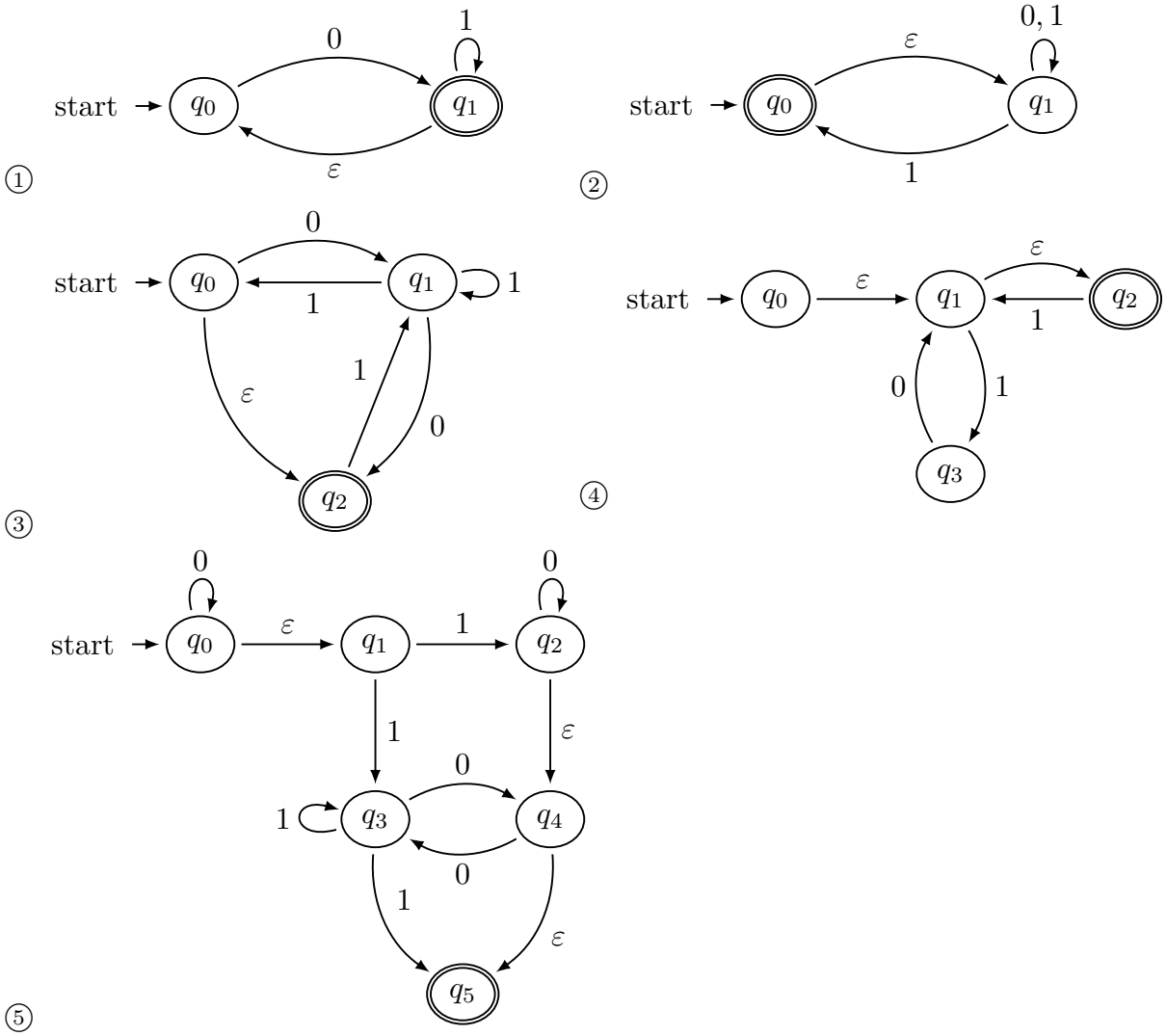
図 2.15　例 2.6（17 ページ）の $M_{\varepsilon 1}$ と等価な ε-動作を除去した NFA M_{NFA1}.

	δ'_{NFA1}	0	1	2
初期/受理	q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
	q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
受理	q_2	ϕ	ϕ	$\{q_2\}$

表 2.8　例 2.6 から ε-動作を除去した NFA の状態遷移表

演習 2.17 例 2.7 で取り上げた ε-NFA $M_{\varepsilon 2}$ について、ε-動作を除去して等価な非決定有限オートマトン $M_{NFA2} = (Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}, \Sigma = \{1\}, \delta'_{NFA2}, q_0, F'_2)$ を図示し、その状態遷移表を与えなさい。

演習 2.18 次の①から⑤の ε-NFA から ε-動作を除去して等価な NFA を構成しなさい [15, p.111, 例題 4.40]。



2.4 出力のある順序機械

有限機械の状態と入力との組によって次の状態と出力が決まる機械を順序機械 (sequential machine) という。順序機械は入力記号列から出力記号列への変換器 (transducer) と考えることができ、以下で説明するようにミーリー機械とムーア機械の 2 種類がある。この 2 つは互いに他方の機械に変換できるという意味で同等であることがわかる。

2.4.1 Mealy 機械

定義 2.12 (Mealy 機械) 有限状態集合 Q、入力アルファベット Σ、状態遷移関数 $\delta : Q \times \Sigma \to Q$、出力関数 $\lambda : Q \times \Sigma \to \Delta$ および初期状態 $q_0 \in Q$ について、入力列 $\boldsymbol{x} \in \Sigma^*$

で到達する状態が $\delta(q_0, \boldsymbol{x}) = p$ であるとき入力列 $\boldsymbol{x}a$ の出力が

$$\mathcal{O}_\lambda(\boldsymbol{x}a) = \lambda(\delta(q_0, \boldsymbol{x}), a)$$

で定まる順序機械を Mealy 機械 $M = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$ と称する。

状態遷移関数 δ を $Q \times \Sigma^*$ 上の関数に拡張したのと同様に、出力関数 λ を $\lambda : Q \times \Sigma^* \to \Delta^*$ に次のように再帰的に拡張しておく。

$$\begin{aligned} &\lambda(q, \varepsilon) = \varepsilon \qquad q \in Q \\ &\lambda(q, ax) = \lambda(q, a)\lambda(\delta(q, a), x) \qquad q \in Q, a \in \Sigma, x \in \Sigma^*. \end{aligned}$$

$a \in Q, \mathrm{B} \in \Delta$ に対して状態遷移 $\delta(p, a) = q$ と出力が $\lambda(p, a) = \mathrm{B}$ によって一意に定まるように、状態 p とそれへの入力 a および出力 B 付き状態遷移 q との関係を図 2.16 にように描く。

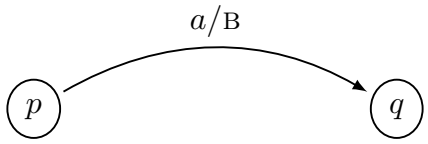


図 2.16　Mealy 機械。状態 p から入力 $a \in \Sigma$ に応じて q へと遷移する際に出力 $\mathrm{B} \in \Delta$ を生じる

例 **2.12** 次の状態遷移表 2.9 は、状態 $Q = \{p, q\}$、入力 $\Sigma = \{0, 1\}$、出力 $\Delta = \{0, 1\}$ を持つ Mealy 機械 M_λ を定義する。M_λ は図 2.17 にように描くことができる。

状態	遷移先		出力	
	入力		入力	
	0	1	0	1
p	p	q	0	0
q	p	q	0	1

表 2.9　Mealy 機械 M_λ の状態遷移表

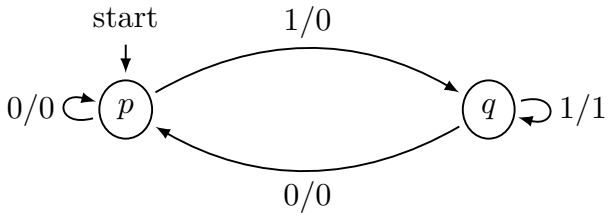


図 2.17　Mealy 機械 M_λ.

この Mealy 機械 M_λ に入力記号列 0101110... を入力したときの状態遷移とその出力は次のようになる。

入力		0		1		0		1		1		1		0		...
状態	p		p		q		p		q		q		q		p	...
出力		0		0		0		0		1		1		0		...

2.4.2　ムーア機械

定義 2.13 (Moore 機械) 有限状態集合 Q、入力アルファベット Σ、状態遷移関数 $\delta : Q \times \Sigma \to Q$、状態出力関数 $\mu : Q \to \Delta$ とする。このとき、初期状態 $q_0 \in Q$ について、入力記号列 $\boldsymbol{x} \in \Sigma^*$ の出力 $\mathcal{O}_\tau(x)$ が再帰的に

$$\mathcal{O}_\tau(\varepsilon) = \tau(\delta(q_0, \varepsilon) = \tau(q_0)$$
$$\mathcal{O}_\tau(x) = \tau(\delta(q_0, x))$$

で与えられる順序機械を Moore 機械 $M = (Q, \Sigma, \delta, \Delta, \tau, q_0)$ と称する。

$a \in Q$, B,C $\in \Delta$ として、状態出力 $\tau(p) =$ B, $\tau(q) =$ C のある状態遷移 $\delta(p, a) = q$ が一意に定まる出力付き状態 p と q との関係を図 2.18 にように描く。

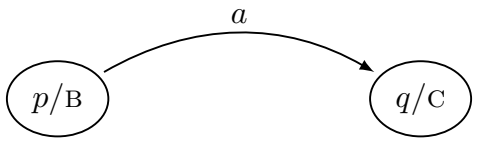


図 2.18　Moore 機械。出力 B を持つ状態 p が入力 $a \in \Sigma$ によって出力 C を持つ q へ遷移する（$p, q \in Q$, B, C $\in \Delta$。

例 **2.13** 次の状態遷移表 2.10 は、状態 $Q = \{r, s, t\}$、入力 $\Sigma = \{0, 1\}$、出力 $\Delta = \{0, 1\}$ を持つ Moore 機械 M_μ を定義する。M_μ は図 2.19 にように描くことができる。

状態	遷移先		出力
	入力		
	0	1	
p	p	q	0
q	p	q	1

表 2.10　Moore 機械 M_μ の状態遷移表

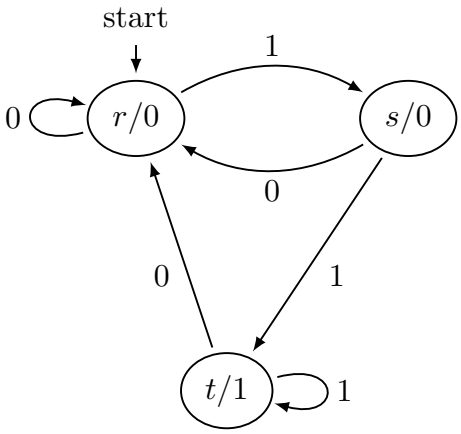


図 2.19　Moore 機械 M_μ.

この Moore 機械 M_μ に入力記号列 0101110... を入力したときの状態遷移とその出力は次のようになる。

入力	ε	0	1	0	1	1	1	0	...
状態	r	r	s	r	s	t	t	r	...
出力	0	0	0	0	0	1	1	0	...

2.4.3　順序機械としての有限オートマトン

出力が 0 と 1 のような 2 値 $\Delta = \{0, 1\}$ だけを持ち、ある入力記号列を入力し終わった段階で 1 または 0 を出力する順序機械を考える。この順序機械は、入力記号列を読み込ん

で 1 が出力されたら入力を受理 (accept) し、0 が出力されたら入力列を受理しないと判定する認識機械と見なすことができる。

実際、状態出力関数 $\tau(q) = \{0, 1\}(q \in Q)$ を持つ Moore 機械 $M = (Q, \Sigma, \delta, \{0, 1\}, \tau, q_0)$ を考える。出力 1 を持つ状態を受理状態、出力 0 を持つ状態を非受理状態としたとき、受理状態の集合 $F \subseteq Q$ を

$$F = \{p \in Q \mid \tau(p) = 1\}$$

で定める。$x \in \Sigma^*$ を入力列として M に与え、初期状態 q_0 から開始して文字列を読み込みながら最後に達した状態で 1 または 0 の出力に応じて入力 x を受理するか否かを定める認識する Moore 機械が有限オートマトン $(Q, \Sigma, \delta, q_0, F)$ に他ならない。

2.4.4 Mealy 機械と Moore 機械の同等性

例に挙げた Mealy 機械 M_λ と Moore 機械 $M_m u$ の入力入力記号列 0101110... に対する状態と出力を並べてみると、それらの出力は初期状態 q_0 に関する出力 (0) を除いてずらすと同じであることがわかる。

入力	ε	0	1	0	1	1	1	0	...
M_λ の状態	p	p	q	p	q	q	q	p	...
M_λ の出力		0	0	0	0	1	1	0	...
M_μ の状態	r	r	s	r	s	t	t	r	...
M_μ の出力	(0)	0	0	0	0	1	1	0	...

このことは一般的に成立し、初期状態に対する空文字 ε 入力に対する出力を除いて、任意の入力文字列に対して Mealy 機械と Moore 機械とが同じ出力文字列を生成するように相互変換できることが次の 2 つの定理からわかる [10, p.19]。この意味において Mealy 機械と Moore 機械は同等である。

定理 2.4 与えられた Mealy 機械から、初期状態 q_0 に対する空文字 ε 入力に対する出力を除いて、同じ出力列を生成する Moore 機械を構成することができる。

証明　まず、例 2.12 の Mealy 機械のように遷移先途中で出力される出力が各遷移先ごとに同じになっている場合、つまり図 2.20 のような “各状態 q に対してそこへの遷移途中の出力がすべて同一の B” である場合に、状態だけから定まる出力関数 $\mu : Q \to \Delta$ を $\mu(q) = \text{B}$ で定義して目的の Moore 機械 M_μ が構成できることを示す。

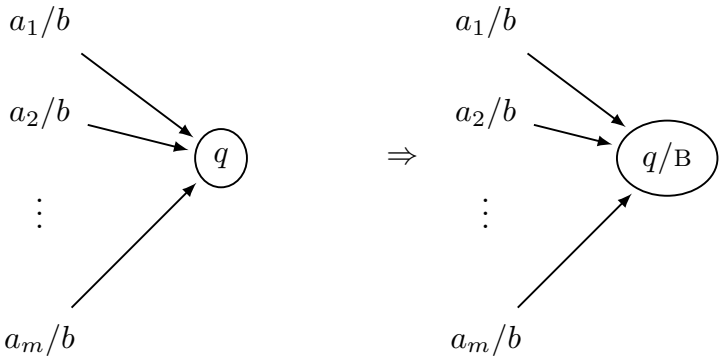


図 2.20　Mealy 機械 M_λ の各状態 q に対して、そこへの遷移途中の出力がすべて同じ B であるときの Moore 機械 M_μ への変換

図 2.21 の Mealy 機械 $M_{\lambda'}$ は、以上の手順によって例 2.13 の Moore 機械（図 2.19）に変換することができ、初期状態 q_0 に対する出力を除いて全ての入力列に関して同じ出力列を発生する。

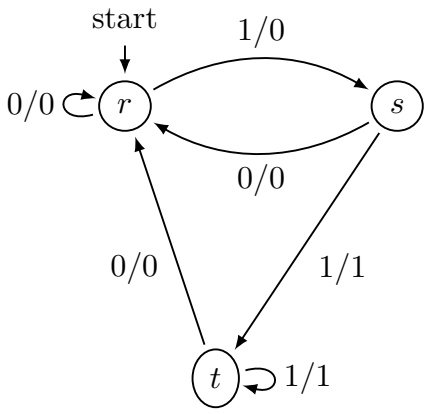


図 2.21　Mealy 機械 $M_{\lambda'}$ は例 2.13 の Moore 機械（図 2.19）と初期状態からを除いて同じ出力を持つ

次に、一般の Mealy 機械で遷移途中の出力は遷移先が同じでも、図 2.22 の左側のように、遷移状態事に同じ出力とはならない場合でも、初期条件についての出力を除いて等価な Moore 機械が構成できることを示す。任意の Mealy 機械 M_λ に対して、以下の手続きに従って状態を増やすことによって、M_λ の同じ出力列を発生し、各遷移先状態事に同じ出力となるような Mealy 機械 $M_{\lambda'}$ を作ることが可能である。

図 2.22 の左の Mealy 機械 M_λ は、ある状態 q に遷移する途中で出力される記号が

$\mathrm{B}_1, \mathrm{B}_2, \ldots, \mathrm{B}_n$ の n 種類ある様子を示している。このとき、遷移先状態 q をその途中の出力記号に応じて n 個の状態 $q_1, q_2, \ldots, q_n$ に置き換えて、遷移先状態 q_i への途中出力がみな同じ B_i になるようにする。ただし、各 q_i から入力に応じて遷移する先は、図 2.22 の右にように、元の q の遷移先と同じ状態とする。

したがって、一般の Mealy 機械 M_λ に対して、各 q_i からの各遷移先途中についても図 2.22 の変換と同様な手順を繰り返すことによって、各状態遷移途中の出力記号がみ Mealy 機械 $M_{\lambda'}$ を構成し、次いで図 2.20 の変換によって Mealy 機械 $M_{\lambda'}$ を目的の Moore 機械に変換することができる。

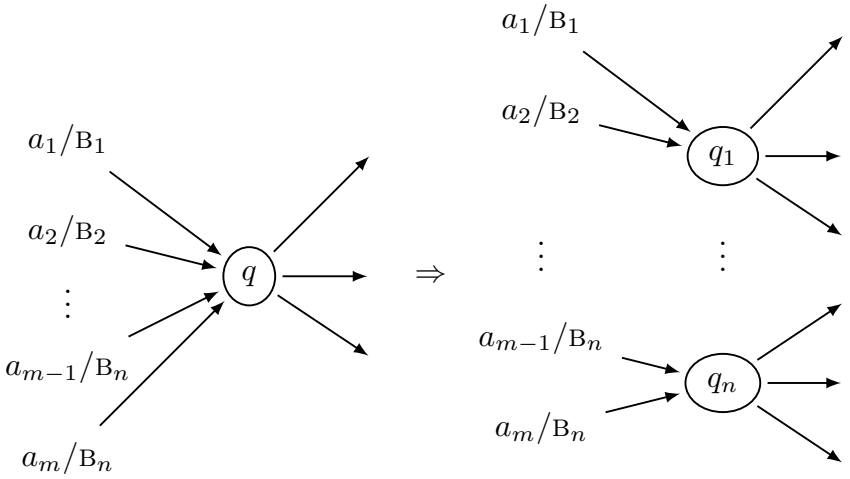


図 2.22　一般の Mealy 機械 M_λ と同値な Mealy 機械 $M_{\lambda'}$ への変換

例 2.12 の Mealy 機械 M_λ（図 2.17）では、状態 q に遷移する途中の出力は 0 と 1 の 2 種類ある。上の手順にしたがって、状態 q を途中出力 0 を持つ状態を $q_{(0)}$ に、途中出力 1 を持つ状態を $q_{(1)}$ として 2 状態に置き換えて、図 2.23 の Mealy 機械 $M_{\lambda''}$ が得られる。集合 $Q = \{p, q_{(0)}, q_{(1)}\}$ から $Q'' = \{r, s, t\}$ への同型対応 $p \mapsto r, q_{(0)} \mapsto s, q_{(1)} \mapsto t$ によって、$M_{\lambda''}$ は図 2.21 の Mealy 機械 $M_{\lambda'}$ と同型になり、結果として例 2.13 の Moore 機械 M_μ が得られる。

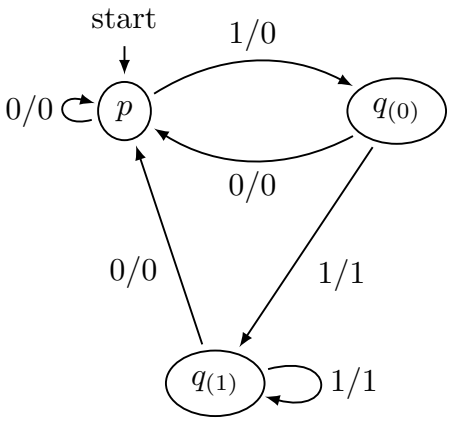


図 2.23　例 2.12 の Mealy 機械 M_λ（図 2.17）と等価な Mealy 機械 $M_{\lambda''}$。$M_{\lambda''}$ は例 2.13 の Moore 機械（図 2.19）と初期状態からを除いて同じ出力を持つ。

■

定理 2.5 与えられた Moore 機械から、初期状態 q_0 に対する空文字 ε 入力に対する出力を除いて、同じ出力列を生成する Mealy 機械を構成することができる。

証明　Moore 機械 $M = (Q, \Sigma, \delta, \Delta, \mu, q_0)$ の各状態 $q \in Q$ に関する出力 $\mathrm{B} = \mu(q)$ に対応して、入力 a よって q に遷移する際の出力関数を $\lambda(q, a) = \mathrm{B}$ で定義した Mealy 機械 $M' = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$ を構成すればよい。 ■

順序機械 M の 2 つの状態 s と t に対して、どんな入力にたいしても同じ出力が得られるとき、この 2 状態は**等価** (equivalent) という。つまり、等価な 2 状態は区別する必要がなく 1 つに結合でき、状態 t をそれと等価な状態 s に付け替え、状態 t を省略して得られる（状態数が 1 つ減った）機械 M' は元の機械と同じ入出力特性をもつ。与えられた機械の状態に等価な状態があるとき、総合してより少ない状態数を持つ機械に変換する操作を**簡単化** (simplification) という。

互いに同型な Mealy 機械 $M_{\lambda'}$（図 2.21）の状態 s と t および $M_{\lambda''}$（図 2.23）の状態 $q_{(0)}$ と $q_{(1)}$ とは、入力 0 または 1 について同じ出力と同じ状態遷移するために等価であり、図 2.17 の Mealy 機械 M_λ に簡単化することができることに注意しよう（Myhill-Nerode の定理 4.6（65 ページ）参照）。

演習 2.19 入力 $\Sigma = \{a, b\}$ と出力 $\Delta = \{x, y, z\}$ を持つ図 2.24 および図 2.25 の Mealy

機械から、初期状態に対する空文字 ε 入力に対する出力を除いて、同じ出力列を生成する Moore 機械をそれぞれ構成しなさい [15, p.87]。

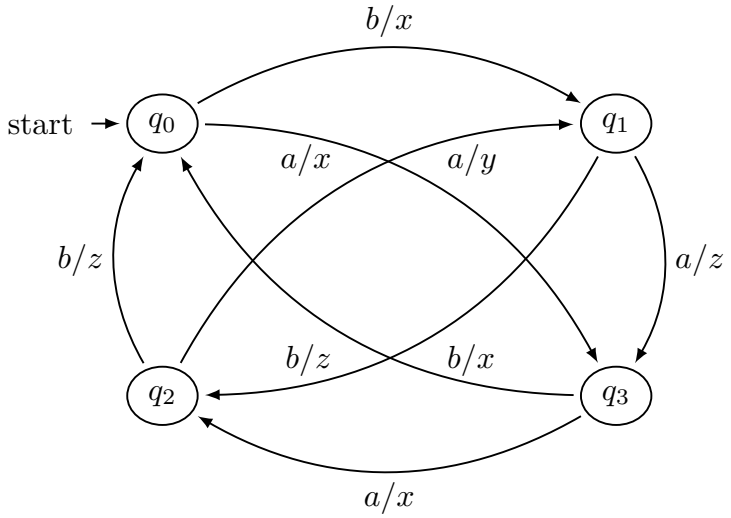


図 2.24　Mealy 機械. $\delta(q_0, a) = q_3, \delta(q_2, a) = q_1, \delta(q_3, b) = q_0, \delta(q_1, b) = q_2$.

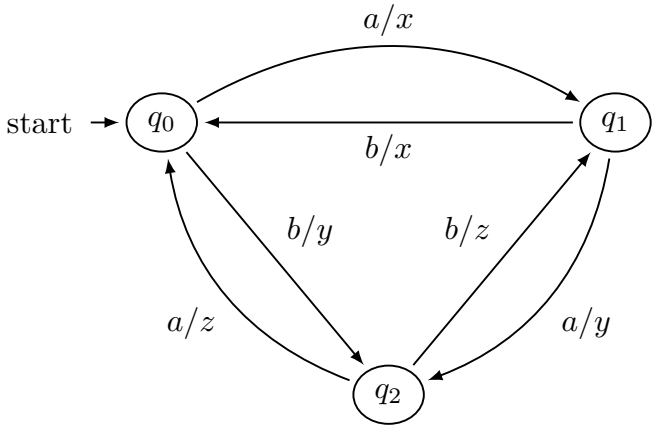


図 2.25　Mealy 機械

第 3 章

正規表現

3.1　正規集合の演算

定義 3.1 アルファベット Σ 上の有限オートマトン $M = (\Sigma, Q, \delta, q_0, F)$ が受理する語の集合を**正規集合**（regular set）と称する。

正規集合の族

$$\mathcal{R}(\Sigma) = \{L(M) \mid M = (\Sigma, Q, \delta, q_0, F) \text{ は有限オートマトン }\}$$

は集合演算 $\cap, \cup$ および補集合 ^{c} のもとで Bool 代数をなすことがわかる。このためには次の 2 つの補題が成り立つことを見ればよい。

補題 3.1 正規集合 $L \subseteq \Sigma^*$ の補集合 $L^c = \Sigma^* - L$ は正規集合である。

証明　L を受理する DFA を $M = (\Sigma, Q, \delta, q_0, F)$ とする。受理状態集合 F の補集合を改めて受理状態とする DFA $M^c = (\Sigma, Q, \delta, q_0, Q - F)$ を考えると

$$L(M^c) = \Sigma - L(M)$$

となる。 ■

補題 3.2 $L_1, L_2 \subseteq \Sigma^*$ を正規集合とする。このとき

(1) $L_1 \cup L_2$ は正規集合である。
(2) $L_1 \cap L_2$ は正規集合である。

証明

(1) $M_i = (\Sigma, Q_i, \delta_i, q_0^i, F_i)$ を言語 L_i を受理する DFA とする（$i = 1, 2$）。このとき次で定義される DFA $M_\cup = (\Sigma, Q_1 \times Q_2, \delta, (q_0^1, q_0^2), F_\cup)$ を考える。

$$\delta((q^1, q^2), a) = (\delta_1(q^1, a), \delta_2(q^2, a)), \qquad q^1 \in Q_1, q^2 \in Q_2, a \in \Sigma$$
$$F_\cup = F_1 \times Q_2 \cup Q_1 \times F_2.$$

このとき、

$$L(M_\cup) = L(M_1) \cup L(M_2) = L_1 \cup L_2$$

であることが確かめられる。実際、入力文字列の長さに関する帰納法によって、

$$\delta((q^1, q^2), \varepsilon) = (q^1, q^2) = (\delta_1(q^1, \varepsilon), \delta_2(q^2, \varepsilon))$$

および長さ n 以下の語 x に対して

$$\delta((q^1, q^2), x) = (\delta_1(q^1, x), \delta_2(q^2, x))$$

より、

$$\begin{aligned}
\delta((q^1, q^2), ax) &= \delta(\delta(q^1, q^2), a), x)) \\
&= \delta((\delta_1(q^1, a), \delta_2(q^2, a)), x) \\
&= \big(\delta_1(\delta_1(q^1, a), x), \delta_2(\delta_2(q^2, a)), x\big) \\
&= \big(\delta_1(q^1, ax), \delta_2(q^2, ax)\big)
\end{aligned}$$

これより、

$$\begin{aligned}
x \in L(M_\cup) &\rightleftharpoons \delta((q_0^1, q_0^2), x) \in F_\cup \\
&\rightleftharpoons \big(\delta_1(q_0^1, x), \delta_2(q_0^2, x)\big) \in F_1 \times Q_2 \cup Q_1 \times F_2 \\
&\rightleftharpoons \delta_1(q_0^1, x) \in F_1 \text{ or } \delta_2(q_0^2, x) \in F_2 \\
&\rightleftharpoons x \in L(M_1) \cup L(M_2)
\end{aligned}$$

これより (1) が示された。

(2) 次に DFA $M_\cap = (\Sigma, Q_1 \times Q_2, \delta, (q_0^1, q_0^2), F_\cap)$ を

$$F_\cap = F_1 \times F_2$$

として同様の議論をして (2) が示される。

■

これらの補題から、次の定理が得られる。

定理 3.1 アルファベット Σ 上の正規集合の族 $\mathcal{R}(\Sigma)$ は集合演算 $\cup, \cap$ および ^{c} に関してブール代数をなす。

3.2 正規集合の同値類

Σ 上の正規集合は同等な命題によって特徴付けることができる。Σ^* 上の関係 R が

$x, y \in \Sigma$ に対して $xRy \Rightarrow$ 任意の $z \in \Sigma^*$ に対して $xzRyz$

を満たすとき、関係 R は**右不変**（right invariant）という。Σ^* 上の同値関係 R の同値類の数が有限であるとき、R は**有限指数**（finite index）を持つという。このとき、正規集合は次の Myhill-Nerode の定理（節 4.6）が成立する。この意味とその応用については、節 4.1 の正規集合の等価性の問題で詳しく取り上げる。

定理 3.2 (Myhill-Nerode) Σ 上の言語 L に対して定義される右不変な同値関係を R_L とするとき、以下の命題は互いに同等である。ここで、同値関係の指数とはその同値類の濃度 (集合の要素数）である。

(1) L は正規言語（有限オートマトンの受理集合）である。
(2) L は、有限指数を持つ右不変な同値関係 R のその幾つかの同値類の直和として表される。
(3) 同値関係 R_L の指数は有限である。

3.3 正規表現

FA が受理する正規言語はどのように表現されるだろか。一般に機械 M が与えられたとき、M によって受理される言語は無限個の語からなる。以下で見るように、もし M が有限オートマトンであるとき、M によって受理される正規言語（一般に無限個の語からなる）は有限的な表現によって表されるという著しい特徴がある。この表記法が正規表現である。

注意 3.1 テキスト処理においてしばしば利用される正規表現とは、指定する文字列パターンを正規表現で表して、そのパターンがテキストの部分文字列にマッチする（パターンを factor として持つ）かを判定し、目的の処理を行うための作業を意味する用語である。文字列照合については、第 5 章で考えた。

もしテキスト全体が正規表現されているならば、テキストはある有限オートマトンで受理されることになるが、プログラミングのソースコードや自然言語は有限オートマトンでは受理できない。

定義 3.2 (正規表現) アルファベット Σ 上の正規表現 (regular expression) を次のように演算 $+$ と連接および Kleene 閉包 $*$ を使って再帰的に定義する。

(1) $\boldsymbol{\phi}$ は正規表現で、それが表す集合は空集合である。
(2) $\boldsymbol{\varepsilon}$ は正規表現で。それが表す集合は $\{\varepsilon\}$ である。
(3) Σ の各要素 a に対して $\boldsymbol{a}$ は正規表現で、それが表す集合は $\{a\}$ である。
(4) $\boldsymbol{\alpha}$ と $\boldsymbol{\beta}$ が言語 A と B を表す正規表現のとき、$(\boldsymbol{\alpha}+\boldsymbol{\beta})$ は正規表現で、それが表す集合は $A \cup B$. $(\boldsymbol{\alpha\beta})$ は正規表現で、それが表す集合は AB。$(\boldsymbol{\alpha}^*)$ は正規表現で、それが表す集合は A^*.

2 つの正規表現 $\boldsymbol{R}, \boldsymbol{S}$ が $\boldsymbol{R} = \boldsymbol{S}$ であるとは、それらの表す集合が一致することである。

例 3.1 正規表現 $\mathbf{00}$ は集合 $\{00\}$ を表し、正規表現 $(\mathbf{0}+\mathbf{1})^*$ は、0 と 1 からなる任意の文字列集合を表す。

$(\mathbf{0}+\mathbf{1})^*\mathbf{00}(\mathbf{0}+\mathbf{1})^*$ は 00 を含む 0 と 1 からなる文字列全体、$(\mathbf{1}+\mathbf{10})^*$ は、空列および 1 で始まり 2 つ連続した 0 を含まない 0 と 1 からなる文字列全体を表す。

また、$(\mathbf{0}+\mathbf{1}^*)^* = (\mathbf{0}+\mathbf{1})^*$ であることは Kleene 閉包の定義からわかる。

命題 3.1 (正規表現の性質) 正規表現 $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ を正規表現とするとき、以下が成立する。

(1) 交換律 $\boldsymbol{\alpha}+\boldsymbol{\beta} = \boldsymbol{\beta}+\boldsymbol{\alpha}$
(2) 結合律 $\boldsymbol{\alpha}+(\boldsymbol{\beta}+\boldsymbol{\gamma}) = (\boldsymbol{\alpha}+\boldsymbol{\beta})+\boldsymbol{\gamma}, \quad \boldsymbol{\alpha}(\boldsymbol{\beta\gamma}) = (\boldsymbol{\alpha\beta})\boldsymbol{\gamma}$
(3) 分配律 $\boldsymbol{\alpha}(\boldsymbol{\beta}+\boldsymbol{\gamma}) = \boldsymbol{\alpha\beta}+\boldsymbol{\alpha\gamma}, \quad (\boldsymbol{\alpha}+\boldsymbol{\beta})\boldsymbol{\gamma} = \boldsymbol{\alpha\gamma}+\boldsymbol{\beta\gamma}$
(4) ベキ等律 $\boldsymbol{\alpha}+\boldsymbol{\alpha} = \boldsymbol{\alpha}$
(5) 単位元 $\boldsymbol{\alpha}+\boldsymbol{\phi} = \boldsymbol{\alpha}, \quad \boldsymbol{\alpha\phi} = \boldsymbol{\phi\alpha} = \boldsymbol{\alpha}, \quad \boldsymbol{\alpha\phi}^* = \boldsymbol{\phi}^*\boldsymbol{\alpha} = \boldsymbol{\alpha}$
(6) $\boldsymbol{\alpha}^* = \boldsymbol{\phi}^* + \boldsymbol{\alpha}^*\boldsymbol{\alpha}, \quad \boldsymbol{\alpha}^* = (\boldsymbol{\phi}^* + \boldsymbol{\alpha})^*$

証明　Salomaa[17] では公理系として挙げている。正規表現と語の連接の定義から示

すことができる。

(5) の $\boldsymbol{\alpha\phi} = \boldsymbol{\phi}$ を示そう。言語 A, B, C に関して言語の連接が分配的であることから $A(B \cap C) = (AB) \cap (AC)$. B と C に共通の語がなければ AB と AC にも共通の語はなく、$B \cap C = \phi$ は $(AB) \cap (AC) = \phi$ となる。よって、$A\phi = \phi$.

$\boldsymbol{\alpha\phi}^* = \boldsymbol{\alpha}$ については、Kleene の $*$ 演算の定義から $\boldsymbol{\phi} = \{\varepsilon\}$ であることに注意する。よって、$\boldsymbol{\alpha\phi}^* = \boldsymbol{\alpha}\varepsilon = \boldsymbol{\alpha}$ である。 ■

演習 **3.1** $\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}$ を正規表現とする。Kleene の $*$ 演算について次の等式を示しなさい。

(1) $\boldsymbol{R}^* = (\boldsymbol{R}^*)^* = \boldsymbol{R}^*\boldsymbol{R}^* = (\varepsilon + \boldsymbol{R})^* = \varepsilon + \boldsymbol{R}\boldsymbol{R}^* = \varepsilon + \boldsymbol{R}^*\boldsymbol{R}$

(2) $(\boldsymbol{R} + \boldsymbol{S})^* = (\boldsymbol{R}^*\boldsymbol{S}^*)^* = (\boldsymbol{R}^*\boldsymbol{S})^*\boldsymbol{R}^* = \boldsymbol{R}^*(\boldsymbol{S}\boldsymbol{R}^*)^* = \boldsymbol{R}^* + \boldsymbol{R}^*\boldsymbol{S}(\boldsymbol{R} + \boldsymbol{S})^* = (\boldsymbol{R}^*\boldsymbol{S})^* + (\boldsymbol{S}^*\boldsymbol{R})^*$

(3) $(\boldsymbol{R}\boldsymbol{S})^*\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{S}\boldsymbol{R})^*$

(4) $(\boldsymbol{R}\boldsymbol{S} + \boldsymbol{R})^*\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{S}\boldsymbol{R} + \boldsymbol{R})^*$

(5) $(\boldsymbol{R}^*\boldsymbol{S})^* = \varepsilon + (\boldsymbol{R} + \boldsymbol{S})^*\boldsymbol{S}$

(6) $(\boldsymbol{R}\boldsymbol{S}^*\boldsymbol{T})^* = \varepsilon + \boldsymbol{R}(\boldsymbol{S} + \boldsymbol{T}\boldsymbol{R})^*\boldsymbol{T}$

3.3.1 正規表現の閉包性

命題 **3.2** アルファベット Σ 上の正規表現の集合 $\mathcal{R}$ は代数系である。すなわち、正規表現 $\boldsymbol{R}_1, \boldsymbol{R}_2, \boldsymbol{R}$ に対して、和 $\boldsymbol{R}_1 + \boldsymbol{R}_2$、連接 $\boldsymbol{R}_1\boldsymbol{R}_2$、Kleene 閉包 $\boldsymbol{R}^*$ の各演算も正規表現となって閉じている。また、補集合 $\boldsymbol{R}^c = \Sigma^* - \boldsymbol{R}$ も正規表現である。

証明　正規表現の再帰的定義 3.2 から、正規表現の和、連接、閉包演算を行っても正規表現になっている。

$\boldsymbol{R}$ を受理する DFA $M = (\Sigma, Q, \delta, q_0, F)$ を考えると、$F^c = Q - F$ とする DFA $(\Sigma, Q, \delta, q_0, F^c)$ は $\boldsymbol{R}^c$ を受理し、$\boldsymbol{R}^c$ は正規言語であることがわかる。 ■

命題 **3.3** アルファベット Σ 上の正規表現の集合 $\mathcal{R}$ は半環をなす。すなわち、$\mathcal{R}$ は

(1) 加法について可換モノイドである。

(2) 連接を乗法として非可換モノイドである。

(3) 乗法の加法に関する分配律が成立する。

命題 **3.4** Σ 上の正規言語 $\boldsymbol{R}_1, \boldsymbol{R}_2$ について、$\boldsymbol{R}_1 \cap \boldsymbol{R}_2$ も正規言語である。

証明　集合演算に関する de Morgan の法則より、$\boldsymbol{R}_1 \cap \boldsymbol{R}_2 = (\boldsymbol{R}_1 \cup \boldsymbol{R}_2)^c$. 命題 3.2 より正規言語は集合の和、補集合に関して閉じており、よって $\boldsymbol{R}_1 \cap \boldsymbol{R}_2$ も正規表現である。
■

正規言語の閉包性と対照的に文脈自由言語においては、和集合（$\cup$）および Kkeen 演算（*）については閉じているが、交わり（$\cap$）および補集合（^{c}）演算については閉じていない（第**??**節参照）。

3.4　線形再帰方程式と正規表現

補題 **3.3 (Arden の補題 [16][17])** $\boldsymbol{X}, \boldsymbol{S}, \boldsymbol{T}$ が正規表現であり、かつ $\varepsilon \notin \boldsymbol{T}$ のとき、$\boldsymbol{X}$ に関する線形再帰方程式は次の一意的解を持つ。

$$\boldsymbol{X} = \boldsymbol{S} + \boldsymbol{X}\boldsymbol{T} \quad \text{に対して} \quad \boldsymbol{X} = \boldsymbol{S}\boldsymbol{T}^*$$
$$\boldsymbol{X} = \boldsymbol{S} + \boldsymbol{T}\boldsymbol{X} \quad \text{に対して} \quad \boldsymbol{X} = \boldsymbol{T}^*\boldsymbol{S}$$

証明　言語集合に関する命題 $P = \{\boldsymbol{X} = \boldsymbol{S} + \boldsymbol{X}\boldsymbol{T}\}$ と $Q = \{\boldsymbol{X} = \boldsymbol{S}\boldsymbol{T}^*\}$ とが同値であることを $\boldsymbol{T} = \boldsymbol{\phi}$ および $\boldsymbol{T} \neq \boldsymbol{\phi}$ の場合に分けて示す [15]。

(1) $\boldsymbol{T} = \boldsymbol{\phi}$ のとき。任意の言語 L と空言語 ϕ との連接が $L\phi = \phi L = \phi$ および $\boldsymbol{\phi}^* = \{\varepsilon\}$ であることから、$\boldsymbol{X}\boldsymbol{T} = \boldsymbol{\phi}$ より $P = \{\boldsymbol{X} = \boldsymbol{S}\}$. 一方、$\boldsymbol{S}\boldsymbol{T}^* = \boldsymbol{S}\varepsilon = \boldsymbol{S}$ より $Q = \{\boldsymbol{X} = \boldsymbol{S}\}$. よって、$P = Q$.

(2) $\boldsymbol{T} \neq \boldsymbol{\phi}$ のとき。
まず、$Q \to P$ を示す。Q であるとき、P の $\boldsymbol{X}$ に代入すると、$\boldsymbol{X} = \boldsymbol{S} + \boldsymbol{S}\boldsymbol{T}^*\boldsymbol{T} = \boldsymbol{S}(\varepsilon + \boldsymbol{T}^*\boldsymbol{T}) = \boldsymbol{S}\boldsymbol{T}^*$ となり、P は Q に一致。
次に $P \to Q$ を示す。つまり、P であるとき $\boldsymbol{S}\boldsymbol{T}^* \subset \boldsymbol{X}$ かつ $\boldsymbol{X} \subset \boldsymbol{S}\boldsymbol{T}^*$ を示すことによって $\boldsymbol{X} = \boldsymbol{S}\boldsymbol{T}^*$ であることを証明し、そのことから Q が成立することを示す。

(a) P であることより、$\boldsymbol{S} \subset \boldsymbol{X}$, $\boldsymbol{X}\boldsymbol{T} \subset \boldsymbol{X}$ である。これより、$\boldsymbol{S}\boldsymbol{T} \subset \boldsymbol{X}\boldsymbol{T} \subset \boldsymbol{X}$. したがって $\boldsymbol{S}\boldsymbol{T}^2 \subset \boldsymbol{X}\boldsymbol{T}$ となって、$\boldsymbol{S}\boldsymbol{T}^2 \subset \boldsymbol{X}\boldsymbol{T} \subset \boldsymbol{X}$. これを繰り返すと、$\boldsymbol{S}\boldsymbol{T}^* \subset \boldsymbol{X}$ が得られる。

(b) $\boldsymbol{X}$ に含まれ $\boldsymbol{ST}^*$ に含まれない集合 $\boldsymbol{K} \equiv \boldsymbol{X} - \boldsymbol{ST}^*$ が $\boldsymbol{K} \neq \boldsymbol{\phi}$ と仮定する。$\boldsymbol{K}$ に属する最短の語を w とする（当然、$w \in \boldsymbol{X}$ である）。$w \notin \boldsymbol{ST}^*$ であることから $w \notin \boldsymbol{S}$、よって P であることから $w \in \boldsymbol{XT}$.
$\boldsymbol{T} \neq \{\varepsilon\}$ より、w はある $u \in \boldsymbol{X}$ および $v \in \boldsymbol{T}$ を使って

$$w = uv \qquad \text{ただし、} |u| < |w|$$

と書ける。しかし一方、w は $\boldsymbol{K} \subset \boldsymbol{X}$ の最短の語であるので、$|u| < |w|$ である u は $u \notin \boldsymbol{K}$ であるので、$u \in \boldsymbol{ST}^*$ でなければならない。これより、$w = uv \in (\boldsymbol{ST}^*)(\boldsymbol{T}) = \boldsymbol{ST}^*\boldsymbol{T} \subset \boldsymbol{ST}^*$. したがって、$w \in \boldsymbol{ST}^*$ であることになって、仮定 $w \notin \boldsymbol{ST}^*$ に矛盾する。すなわち、$\boldsymbol{K} = \boldsymbol{X} = \boldsymbol{ST}^* = \phi$、$\boldsymbol{X} \subset \boldsymbol{ST}^*$ がわかった。

(a), (b) より $\boldsymbol{X} = \boldsymbol{ST}^*$ が示され、$P \to Q$ が証明された。

■

例 **3.2** アルファベット $\Sigma = \{0, 1\}$ 上の正規表現に関して、次の $\boldsymbol{X}, \boldsymbol{Y}$ に関する線形再帰方程の解を求めてみる [15, p.100]。

(1) $\boldsymbol{X} = \boldsymbol{00} + \boldsymbol{11} + \boldsymbol{X0} + \boldsymbol{X1}$
(2) $\boldsymbol{X} = \boldsymbol{01} + \boldsymbol{X0}^*\boldsymbol{1}$
(3) $\boldsymbol{X} = \boldsymbol{0} + \boldsymbol{X1}^*\boldsymbol{01} + \boldsymbol{Y1}(\boldsymbol{01})^*$, $\boldsymbol{Y} = \boldsymbol{X1}$
(4) $\boldsymbol{X} = \boldsymbol{11}^*\boldsymbol{0} + \boldsymbol{X01} + \boldsymbol{Y1}$, $\boldsymbol{Y} = \boldsymbol{11}^* + \boldsymbol{X0} + \boldsymbol{Y1}$

補題 3.3 と Kleene 演算に関する演習 3.1 を使う。正規表現の表式は語の集合であることに注意する。

(1) $(\boldsymbol{00} + \boldsymbol{11})(\boldsymbol{0} + \boldsymbol{1})^*$
(2) $\boldsymbol{01}(\boldsymbol{0}^*\boldsymbol{1})^* = \boldsymbol{01}(\varepsilon + \boldsymbol{0}^*\boldsymbol{1}(\boldsymbol{0}^*\boldsymbol{1})^*) = \boldsymbol{01} + \boldsymbol{010}^*\boldsymbol{1}(\boldsymbol{0}^*\boldsymbol{1})^* = \boldsymbol{01} + \boldsymbol{01}(\boldsymbol{0}^*\boldsymbol{1})^* = \boldsymbol{01} + (\boldsymbol{0}^*\boldsymbol{1})^*$
(3) $\boldsymbol{Y} = \boldsymbol{X1}$ を代入すると、解くべき方程式 $\boldsymbol{X} = \boldsymbol{0} + \boldsymbol{X}(\boldsymbol{1}^*\boldsymbol{01} + \boldsymbol{11}(\boldsymbol{01})^*)$ を得る。これより、$\boldsymbol{X} = \boldsymbol{0}(\boldsymbol{1}^*\boldsymbol{01} + \boldsymbol{11}(\boldsymbol{01})^*)^*$, $\boldsymbol{Y} = \boldsymbol{0}(\boldsymbol{1}^*\boldsymbol{01} + \boldsymbol{11}(\boldsymbol{01})^*)^*\boldsymbol{1}$
(4) $\boldsymbol{Y} = (\boldsymbol{11}^* + \boldsymbol{X0}) + \boldsymbol{Y1}$ として $\boldsymbol{Y}$ について解くと、$\boldsymbol{Y} = (\boldsymbol{11}^* + \boldsymbol{X0})\boldsymbol{1}^* = \boldsymbol{11}^* + \boldsymbol{X01}^*$ を得る. これを $\boldsymbol{X}$ の方程式に代入して、$\boldsymbol{X} = \boldsymbol{11}^*\boldsymbol{0} + \boldsymbol{11}^*\boldsymbol{1} + \boldsymbol{X}(\boldsymbol{01} + \boldsymbol{01}^*\boldsymbol{1})$. これを解いて、$\boldsymbol{X} = \boldsymbol{11}^*(\boldsymbol{0} + \boldsymbol{1})(\boldsymbol{01} + \boldsymbol{01}^*\boldsymbol{1})^* = \boldsymbol{11}^*(\boldsymbol{0} + \boldsymbol{1})(\boldsymbol{01}^*\boldsymbol{1})^*$. これを $\boldsymbol{Y}$ の式に代入して、$\boldsymbol{Y} = \boldsymbol{11}^*(\varepsilon + \boldsymbol{11}^*(\boldsymbol{0} + \boldsymbol{1})(\boldsymbol{01}^*\boldsymbol{1})^*\boldsymbol{01}^*)$.

例 **3.3** 図 3.1 の有限オートマトン M_a, M_b, M_c, M_d が定める正規表現を求めてみる [15, pp.101–102]。

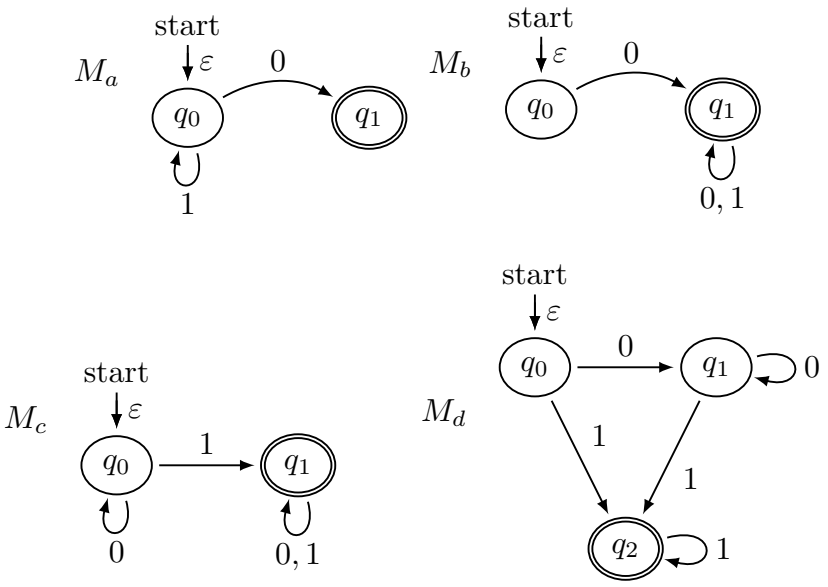


図 3.1　オートマトンから正規表現を求める

有限オートマトン $M = (\Sigma, Q, \delta, q_0, F)$ において、初期状態 q_0 から状態 $q_k \in Q$ に遷移する入力 $x \in \Sigma^*$ 語の集合 $\boldsymbol{R}_k$ がすべて正規表現である。正規言語 $\{\boldsymbol{R}_k\}$ 同士に関する再帰方程式を補題 3.3 を使って解いて求める正規表現を得る。

a) ここでは簡単に、受理状態 q_1 に遷移するためには 0 回以上の 1 が続いた後に 0 である記号列でなければならないと考えて、直ちに $\mathbf{1}^*\mathbf{0}$ を得る。形式的には、$\boldsymbol{R}_0 = \boldsymbol{\varepsilon} + \boldsymbol{R}_0\mathbf{1}$, $\boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{0}$ を連立させて解く（$\boldsymbol{R}_0 = \boldsymbol{\varepsilon} + \mathbf{1}\boldsymbol{R}_0$ ではないことに注意する）。補題 3.3 より $\boldsymbol{R}_0 = \boldsymbol{\varepsilon}\mathbf{1}^* = \mathbf{1}^*$、よって $\boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{0} = \mathbf{1}^*\mathbf{0}$.

b) ここでは簡単に、受理状態 q_1 に遷移するため 1 回の 0 の後は 0 または 1 が 0 回以上現れる記号列でなければならない考えて、直ちに $\mathbf{0}(\mathbf{0}+\mathbf{1})^*$ を得る。$\boldsymbol{R}_0 = \boldsymbol{\varepsilon}$, $\boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{0} + \boldsymbol{R}_1(\mathbf{0}+\mathbf{1})$ を連立させる。$\boldsymbol{R}_1 = \mathbf{0}\boldsymbol{R}_0 + \boldsymbol{R}_1(\mathbf{0}+\mathbf{1})$ や $\boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{0} + (\mathbf{0}+\mathbf{1})\boldsymbol{R}_1$ などではないことに注意する。$\boldsymbol{R}_1 = \mathbf{0} + \boldsymbol{R}_1(\mathbf{0}+\mathbf{1})$ より、$\boldsymbol{R}_1 = \mathbf{0}(\mathbf{0}+\mathbf{1})^*$.

c) 状態 q_0 と q_1 に遷移するための正規表現 $\boldsymbol{R}_0$ および $\boldsymbol{R}_1$ は次の関係にある。

$$\boldsymbol{R}_0 = \boldsymbol{\varepsilon} + \boldsymbol{R}_0\mathbf{0}, \quad \boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{1} + \boldsymbol{R}_1(\mathbf{0}+\mathbf{1})$$

補題 3.3 より、$\boldsymbol{R}_0 = \boldsymbol{\varepsilon}\mathbf{0}^* = \mathbf{0}^*$. これを $\boldsymbol{R}_1$ の式に代入して、$\boldsymbol{R}_1 = \mathbf{0}^*\mathbf{1} + \boldsymbol{R}_1(\mathbf{0}+\mathbf{1})$.

よって、$\boldsymbol{R}_1 = \mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^*$ を得る。

d) 状態 q_0, q_1, q_2 に遷移するための正規表現 $\boldsymbol{R}_0, \boldsymbol{R}_1$ および $\boldsymbol{R}_2$ は次の関係にある。

$$\boldsymbol{R}_0 = \boldsymbol{\varepsilon}, \quad \boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{0} + \boldsymbol{R}_1\mathbf{0}, \quad \boldsymbol{R}_2 = \boldsymbol{R}_0\mathbf{1} + \boldsymbol{R}_1\mathbf{1} + \boldsymbol{R}_2\mathbf{1}$$

$\boldsymbol{R}_0$ を $\boldsymbol{R}_1$ の式に代入して、$\boldsymbol{R}_1 = \boldsymbol{\varepsilon}\mathbf{0} + \boldsymbol{R}_1\mathbf{0} = \mathbf{0} + \boldsymbol{R}_1\mathbf{0}$. 補題 3.3 より、$\boldsymbol{R}_1 = \mathbf{00}^*$. これらを $\boldsymbol{R}_2$ の式に代入して、$\boldsymbol{R}_2 = (\mathbf{1} + \mathbf{00}^*\mathbf{1}) + \boldsymbol{R}_2\mathbf{1}$. したがって、$\boldsymbol{R}_2 = (\mathbf{1} + \mathbf{00}^*\mathbf{1})\mathbf{1}^*$.

例 3.4 有限オートマトンが受理する言語集合としての正規表現は一意には定まらない。図 3.2 の有限オートマトン M_e が受理する言語の正規表現を考えてみよう。

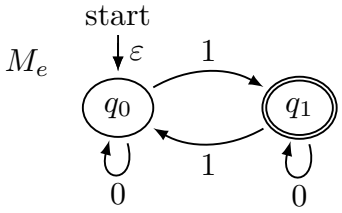


図 3.2　正規表現の形は一意的に定まらない

状態 q_0 と q_1 に遷移するための正規表現 $\boldsymbol{R}_0$ および $\boldsymbol{R}_1$ は次の関係にある。

$$\boldsymbol{R}_0 = \boldsymbol{\varepsilon} + \boldsymbol{R}_0\mathbf{0} + \boldsymbol{R}_1\mathbf{1} \tag{1}$$
$$\boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{1} + \boldsymbol{R}_1\mathbf{0} \tag{2}$$

式 (1) を $\boldsymbol{R}_0 = (\boldsymbol{\varepsilon} + \boldsymbol{R}_1\mathbf{1}) + \boldsymbol{R}_0\mathbf{0}$ として、補題 3.3 より

$$\boldsymbol{R}_0 = (\boldsymbol{\varepsilon} + \boldsymbol{R}_1\mathbf{1})\mathbf{0}^*$$

を得て、これを (2) に代入して次を得る。

$$\boldsymbol{R}_1 = (\boldsymbol{\varepsilon} + \boldsymbol{R}_1\mathbf{1})\mathbf{0}^*\mathbf{1} + \boldsymbol{R}_1\mathbf{0} = \mathbf{0}^*\mathbf{1} + \boldsymbol{R}_1(\mathbf{10}^*\mathbf{1} + \mathbf{0})$$

したがって、

$$\boldsymbol{R}_1 = \mathbf{0}^*\mathbf{1}(\mathbf{10}^*\mathbf{1} + \mathbf{0})^*. \tag{3}$$

一方、式 (2) から $\boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{10}^*$ を得て、これを (1) に代入して

$$\boldsymbol{R}_0 = \boldsymbol{\varepsilon} + \boldsymbol{R}_0\mathbf{0} + \boldsymbol{R}_0\mathbf{10}^*\mathbf{1} = \boldsymbol{\varepsilon} + \boldsymbol{R}_0(\mathbf{0} + \mathbf{10}^*\mathbf{1})$$

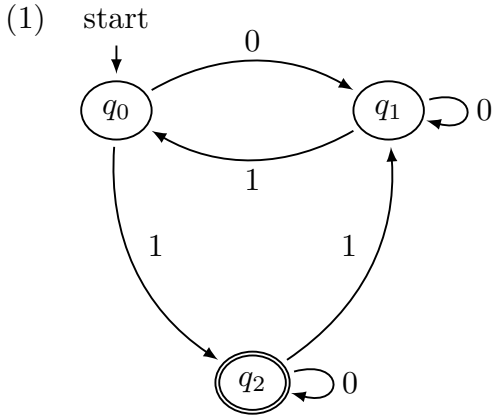
これより、$\boldsymbol{R}_0 = \varepsilon(\mathbf{0} + \mathbf{10}^*\mathbf{1})^* = (\mathbf{0} + \mathbf{10}^*\mathbf{1})^*$ となる。(2) に代入して、$\boldsymbol{R}_1 = (\mathbf{0} + \mathbf{10}^*\mathbf{1})^*\mathbf{1} + \boldsymbol{R}_1\mathbf{0}$. よって、

$$\boldsymbol{R}_1 = (\mathbf{0} + \mathbf{10}^*\mathbf{1})^*\mathbf{10}^* \tag{4}$$

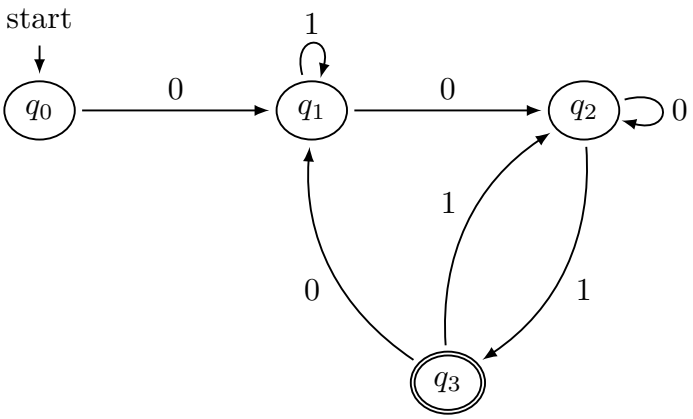
以上から、M_e が受理する言語の正規表現として、式 (3) および (4) の異なる表現が得られた。このように、一般に代入に仕方には任意性があるために、得られる正規表現は様々になり、さらに、演習 3.1 のように Kleene 閉包の表し方もいろいろあるために一意的とはならない。

演習 3.2 次の (1) から (6) の DFA が受理する言語の正規表現を求めなさい。

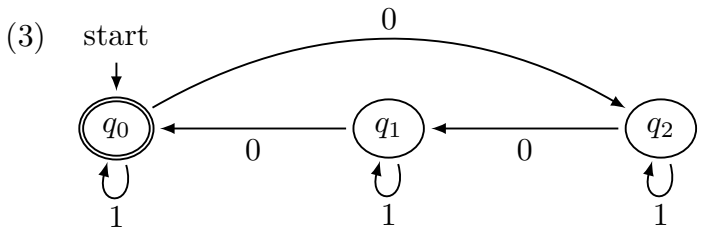
(1)



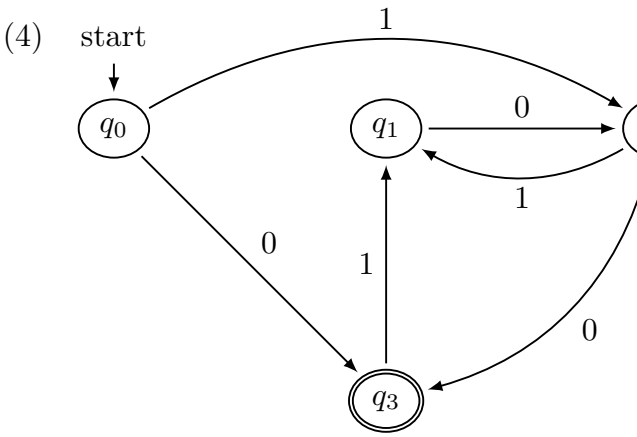
(2)

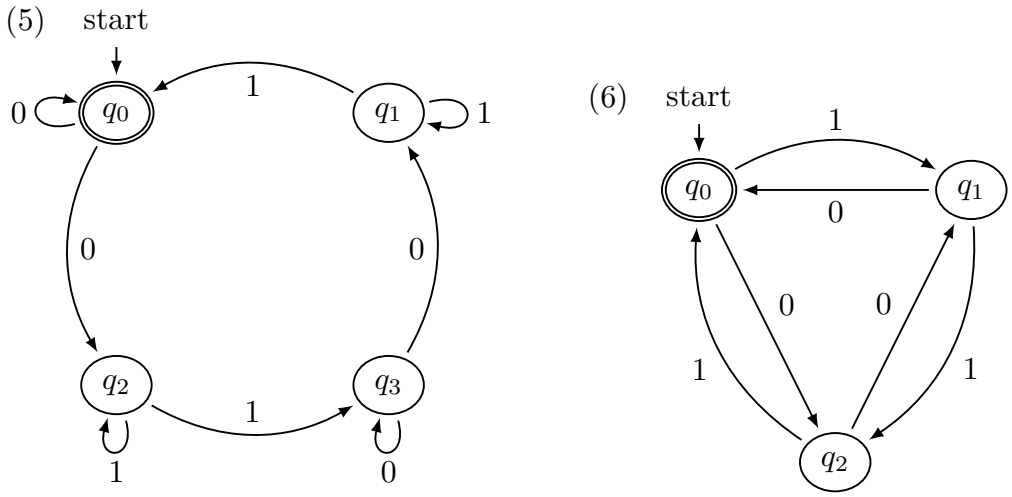


(3)



(4)





3.5　DFA と正規表現の同等性

正規表現はそれによって定義される語の集合すなわち言語を表している。言語はそれを受理する機械によって定義されるという観点に沿って言えば、正規表現で表される言語を受理する機械が存在する（この機械は正規機械というべきものである）。

各正規言語で表される語の集合にはこれを受理する有限オートマトンが存在し、正規表現で表される言語クラスは有限オートマトンで受理される集合のクラスと完全に一致する。図 3.3（3 ページの図 2.1 と同じもの）はこの論理構成を示している。

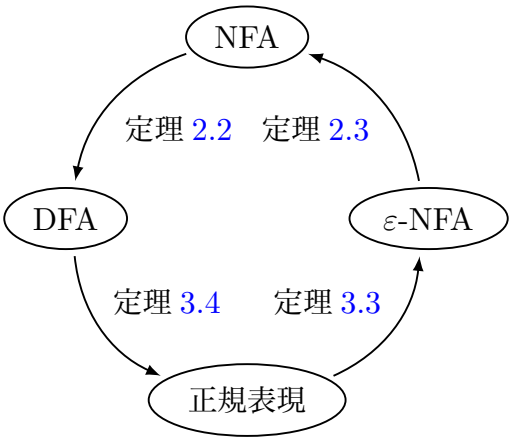


図 3.3　正規表現で表される言語を受理する DFA が存在する

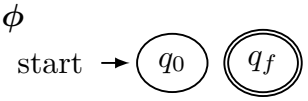
定理 3.3 $\boldsymbol{R}$ を正規表現とするとき、言語 $\boldsymbol{R}$ を受理する ε-NFA が存在する。

証明　与えられた正規表現 $\boldsymbol{R}$ から、言語 $\boldsymbol{R}$ を受理するただ 1 つの最終状態 q_f を有する ε-NFA が構成できることを、正規表現内の演算回数に関する帰納法によって証明する。

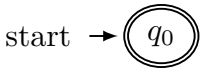
[演算回数が 0 のとき]

正規表現の定義 3.2 から、$\boldsymbol{R}$ は $\boldsymbol{\phi}, \boldsymbol{\varepsilon}$ または $\boldsymbol{a}$（$a \in \Sigma$）である。これらに対応する ε-NFA（DFA）は次の図 (i), (ii), (iii) である。

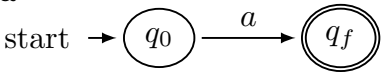
(i) $\boldsymbol{\phi}$



(ii) $\boldsymbol{\varepsilon}$



(iii) $\boldsymbol{a}$



[演算回数が $i \geqq 1$ のとき]

i より少ない演算回数を持つ正規表現 $\boldsymbol{R}_1, \boldsymbol{R}_2$ について、定理が成立すると仮定する。このとき、$\boldsymbol{R}_1, \boldsymbol{R}_2$ を受理するただ 1 つの受理状態を持つ ε-NFA $M_1 = (\Sigma_1, Q_1, \delta_1, q_1, F_1 = \{f_1\})$ および $M_2 = (\Sigma_2, Q_2, \delta_2, q_2, F_2 = \{f_2\})$ が存在する。ただし、M_1, M_2 の状態空間は異なる名前を持つ $Q_1 \cap Q_2 = \phi$ とする。

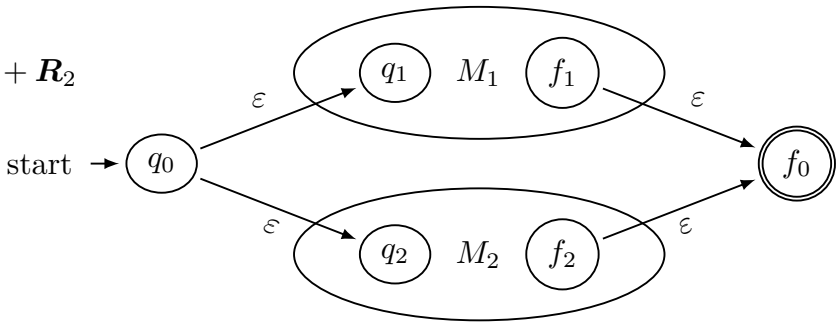
正規表現 $\boldsymbol{R}$ の形 $\boldsymbol{R} = \boldsymbol{R}_1 + \boldsymbol{R}_2$, $\boldsymbol{R} = \boldsymbol{R}_1\boldsymbol{R}_2$, $\boldsymbol{R} = \boldsymbol{R}_1^*$ の場合にのそれぞれについて証明する。

iv) $\boldsymbol{R} = \boldsymbol{R}_1 + \boldsymbol{R}_2$ の場合。Q_1, Q_2 に属さない新たな初期状態 q_0 と受理状態 f_0 を加えて、図のように M_1 と M_2 を並列につないで ε-NFA

$$M = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2 \cup \{q_0, f_0\}, \delta, q_0, \{f_0\})$$

を構成する。M は目的の $\boldsymbol{R} = \boldsymbol{R}_1 + \boldsymbol{R}_2$ を受理する。

$\boldsymbol{R} = \boldsymbol{R}_1 + \boldsymbol{R}_2$

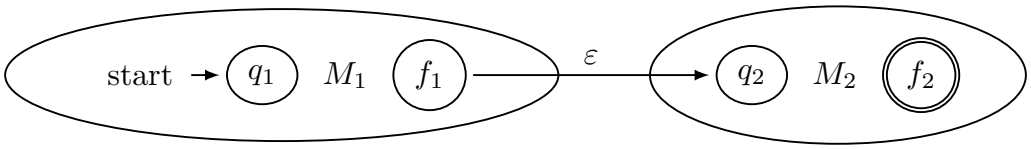


v) $\boldsymbol{R} = \boldsymbol{R}_1\boldsymbol{R}_2$ の場合。M_1 と M_2 を図のように直接につないで ε-NFA

$$M = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2, \delta, q_1, \{f_2\})$$

を構成する。M は目的の $\boldsymbol{R} = \boldsymbol{R}_1\boldsymbol{R}_2$ を受理する。

$\boldsymbol{R} = \boldsymbol{R}_1\boldsymbol{R}_2$

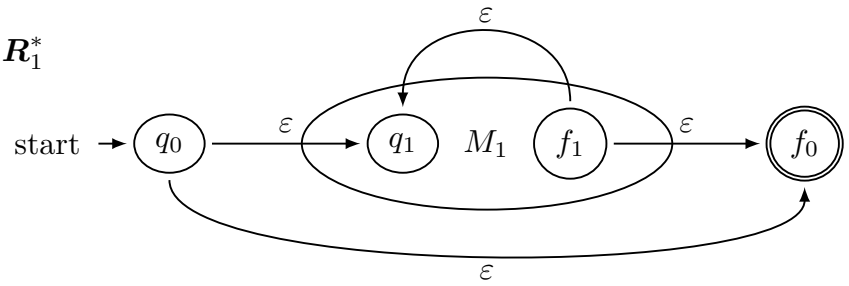


vi) $\boldsymbol{R} = \boldsymbol{R}_1^*$ の場合。Q_1, Q_2 に属さない新たな初期状態 q_0 と受理状態 f_0 を加えて、M_1 を使って図のように ε-NFA

$$M = (\Sigma_1 Q_1 \cup \{q_0, f_0\}, \delta, q_0, \{f_0\})$$

を構成する。M は目的の $\boldsymbol{R} = \boldsymbol{R}_1^*$ を受理する。

$\boldsymbol{R} = \boldsymbol{R}_1^*$



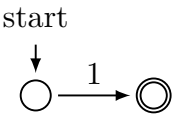
したがって、正規表現 $\boldsymbol{R}$ が与えられたとき、その正規表現を和と連接および Kleene 演算に分解して以上のアルゴリズムに従って ε-NFA が構成できることが証明できた。 ■

例 **3.5** 正規表現 $\mathbf{1}^*(\mathbf{0}^*\mathbf{1})^*$ を受理する ε-NFA を構成する。

正規表現 $\mathbf{1}$ と $\mathbf{0}$ を受理する機械から、$\mathbf{1}^*$ と $\mathbf{0}^*\mathbf{1}$ を受理する機械、次いで $(\mathbf{0}^*\mathbf{1})^*$ を受理する機械を構成し、$\mathbf{1}^*$ と $(\mathbf{0}^*\mathbf{1})^*$ との連接を受理する ε-NFA を構成すればよい。各段階で ε-NFA は自動的に生成することができるが、ε 動作する状態数が増加してしまう。ε 動作する状態を刈り取ることも考えてみよう。

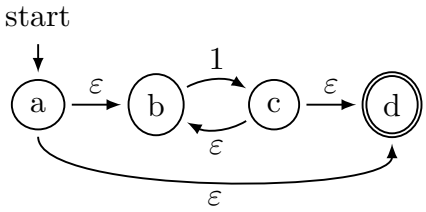
正規表現 $\mathbf{1}$ を受理する機械は次のようになる。

$\mathbf{1}$



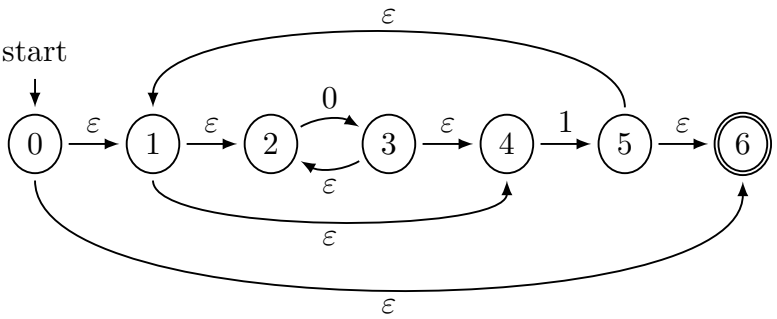
これより、定理 3.3 の証明から $\mathbf{1}^*$ を受理する機械が次のように構成できる。

$\mathbf{1}^*$



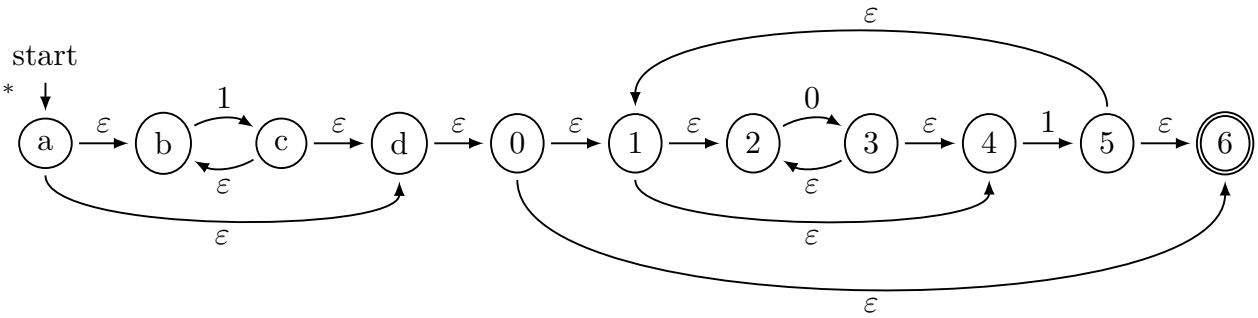
同様に考えて、正規表現 $(\mathbf{0}^*\mathbf{1})^*$ を受理する機械は次のようになる。

$\mathbf{0}^*\mathbf{1}$



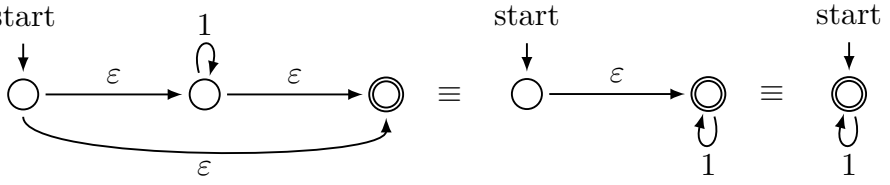
したがって、$\mathbf{1}^*$ と $(\mathbf{0}^*\mathbf{1})^*$ を受理する機械を定理 3.3 の証明のように直列につないで、次の $\mathbf{1}^*(\mathbf{0}^*\mathbf{1})^*$ を受理する ε-NFA が構成できる。

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



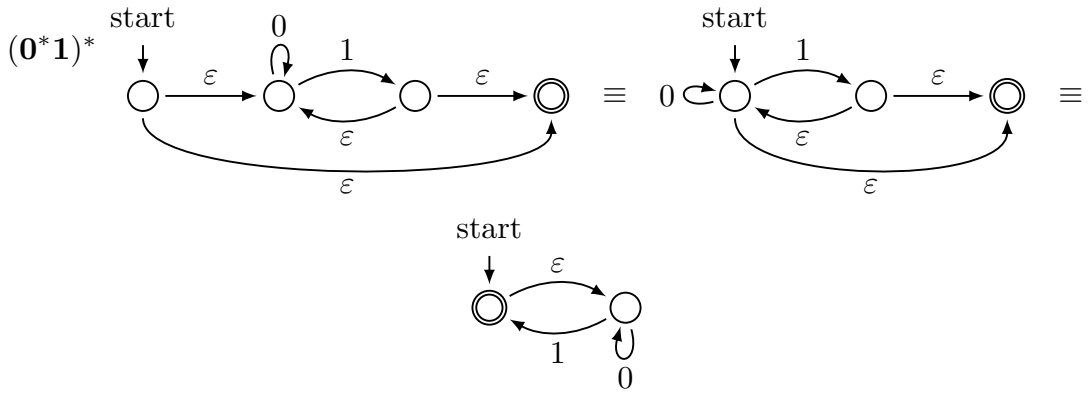
さて、$\mathbf{1}^*$ を受理する機械として、ε-動作で遷移する状態を減らして次のような機械は同等である。

$\mathbf{1}^*$

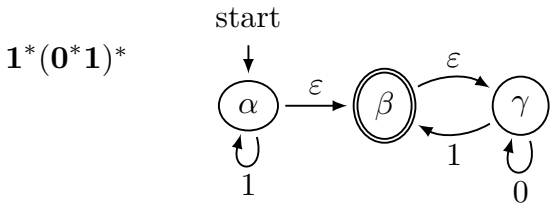


同様に $(\mathbf{0}^*\mathbf{1})^*$ を受理する機械として、て次のように ε-動作で遷移する状態を減らした

同等な機械を作ることができる。



したがって、$\mathbf{1}^*(\mathbf{0}^*\mathbf{1})^*$ を受理する次のような ε-NFA を得ることができる。



この例からわかるようにある正規表現を受理する ε-NFA が多数存在する。その中で状態数を減らして機械を簡素化するために、ここでは個別に工夫した。

節 4.6.1 では、ある言語を受理する可能な限り状態数を減らした最小の機械を考える。

定理 3.4 言語 L が DFA で受理されるとき、L は正規表現で表される。

3.6 正規言語の反復補題

補題 3.4 (正規言語の反復補題) 正規言語 L に対して次の条件を満たす定数 n_L が存在する。長さ n_L 以上の L 内の語 z は次の条件を満たすように $z = uvx$ と分解できる。

(1) $|uv| \leqq n_L$,
(2) $|v| \geqq 1$,
(3) 任意の $i \geqq 0$ について、$uv^i x \in L$.

この定数 n_L は、L を受理する最小（状態数が最も少ない）の有限オートマトンの状態数を越えない。

証明　言語 L が決定性有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ により受理されるとする。ただし、M を L を受理する最小状態を有するように選んでおく。M の状態数を $n_L = |Q|$ とする。長さ n_L 以上の語 $z = a_1 a_2 \ldots a_m (m \geqq n_L)$ が M で受理されるとしたとき、その状態遷移は

$$\delta(a_i, p_{i-1}) = p_i \quad 1 \leqq i \leqq m, a_i \in \Sigma$$

となる。ただし、$p_0 = q_0, p_m \in F$ である。この状態遷移列 $p_0, p_1, \ldots, p_m$ のはじめの $n_L + 1$ の状態 $p_0, p_1, \ldots, p_{n_L}$ を考えると、これに登場する状態数は高々 n_L 個しかなく鳩ノ巣原理より、すくなくともある状態 q_L が次のように 2 回は現れる：

$$p_{n_1} = p_{n_2} = q_L, \quad 0 \leqq n_1 < n_2 \leqq n_L$$

このとき、$z = uvx$ と分解するように $u = a_1 \ldots a_{n_1}, v = a_{n_1+1} \ldots a_{n_2}, x = a_{n_2+1} \ldots a_m$ とおけば

$$p_0 \stackrel{u}{\to} p_{n_1} \stackrel{v}{\to} p_{n_2} \stackrel{x}{\to} p_m$$

のように状態遷移する。このとき、M では同じ状態間を遷移させる語 v を任意回数 $i \geqq 0$ 繰り返す状態遷移経路が可能で、これらはすべて L の属する語となる。 ■

演習 3.3 正規言語に関する反復補題をつかって、記号 a, b からなる言語

$$L_1 = \{a^k b^k \mid k \geqq 0\}$$

は正規言語でないことを示しなさい。

演習 3.4 正規言語に関する反復補題をつかって、a の列で長さが完全平方である語全体からなる言語

$$L_2 = \{a^{k^2} \mid k \geqq 1\}$$

は正規でないことを示しなさい。

第 4 章

オートマトンの等価性

4.1 等価性の問題

DFA $M = (Q, \Sigma, \delta, q_0, F)$ が受理する言語 $L(M)$ は

$$L(M) = L(q_0) = \{x \in \Sigma^* \mid q_0 \underset{M}{\overset{x}{\Rightarrow}}{}^* r, r \in F\}$$

と書いて初期状態 q_0 から受理状態に遷移する入力記号列 x 全体と見なすのであった。任意の状態 $p \in Q$ に対しても、p から受理状態に遷移する入力記号列の集合 $L(p)$

$$L(p) = \{y \in \Sigma^* \mid p \underset{M}{\overset{y}{\Rightarrow}}{}^* r, r \in F\}$$

を定めることができる。

ところで、機械 M が与えられるとこれが受理する言語 $L(M)$ が定まるのであるが、異なる機械 M_1 と M_2 が与えられたとき、それらが受理する言語 $L(M_1)$ および $L(M_2)$ が同じであるのかどうかは直ちに明かではない。また、与えられたオートマトンであっても、状態の数を減らす簡略化または最小化によって、元のオートマトンと同じ言語を受理する機械を構成できるかどうかも興味深い。

こうして、異なる機械であっても同じ言語を受理するかどうか、機械を簡略化して同じ言語を受理するようにするなど、機械の等価性が問題になる。

定義 4.1 (DFA の等価性) 2 つの DFA $M_1 = (Q_1, \Sigma_1, \delta_1, q_{10}, F_1)$ と $M_2 = (Q_2, \Sigma_2, \delta_2, q_{20}, F_2)$ に対し $L(M_1) = L(M_2)$ であるとき、M_1 と M_2 は等価 (equivalent) といい $M_1 \equiv M_2$ と記す。$L(M_1) \neq L(M_2)$ であるとは、一方に受理されるが他方では受理されない入力文字列が存在するときで、M_1 と M_2 は等価でないといい $M_1 \not\equiv M_2$ と記す。

定義 4.2 (状態の等価性) DFA $M = (Q, \Sigma, \delta, q_0, F)$ において、任意の 2 状態 $p, q \in Q$ に対して、$L(p) = L(q)$ であるとき、2 状態は等価であるといい、$p \equiv q$ と記す。

定義 4.3 DFA $M = (Q, \Sigma, \delta, q_0, F)$ において適当な入力列 $x \in \Sigma^*$ が存在して

$$q_0 \overset{x}{\underset{M}{\Rightarrow}}{}^{*} p$$

であると、p は初期状態 q_0 から**到達可能** (reachable) という。

到達可能状態の検出（節 4.2)、2 つの機械の等価性判定（節 4.3）および状態対の等価性判定（節 4.4）は、有限オートマトンの最小化（節 4.6.1）を具体的に実施するために使われる。

等価性（同値性）は一般に順序機械について定義される。

定義 4.4 入力アルファベット Σ と出力アルファベット Δ を共有する出力関数 λ を持つ 2 つの順序機械 $S_i = (Q_i, \Sigma, \delta_i, \Delta, \lambda_i, q_0^i)$（$i = 1, 2$）とする。

[Mealy 機械の状態等価性]

状態 $p \in Q_1$ と $q \in Q_2$ が等価 $p \equiv q$（equivalent）であるとは、任意の入力語 $x \in \Sigma^*$ に対して、その出力列が等しい

$$\lambda_1(p, x) = \lambda_2(q, x)$$

ときである。

[Moore 機械の状態等価性]

状態 $p \in Q_1$ と $q \in Q_2$ が等価 $p \equiv q$ であるとは、任意の入力語 $x \in \Sigma^*$ に対して、その出力列が等しい

$$\mathcal{O}_{\lambda_1}(p, x) = \mathcal{O}_{\lambda_2}(q, x)$$

ときである。

演習 4.1 順序機械の等価性 $\equiv$ は同値関係であることを示しなさい。

補題 4.1 順序機械 S_i において、状態 $p \in Q_1$ と $q \in Q_2$ が等価 $p \equiv q$ であるとき、任意の入力語 $x \in \Sigma^*$ によって到達する状態もまた等価 $\delta_1(p, x) \equiv \delta_2(q, x)$ である。

証明　(Mealy 機械の場合)　$p \equiv q$ であることから、任意の $x, y \in \Sigma^*$ に対して

$$\lambda_1(p, x) = \lambda_2(q, x)$$
$$\lambda_1(p, xy) = \lambda_2(q, xy)$$

語の長さの帰納法によって次が成立することに注意する。

$$\lambda_1(p, xy) = \lambda_1(p, x)\lambda_1(\delta(p, x), y)$$
$$\lambda_2(q, xy) = \lambda_2(q, x)\lambda_2(\delta(q, x), y).$$

したがって

$$\lambda_1(\delta(p, x), y) = \lambda_2(\delta(q, x), y)$$

となって、状態の等価性の定義より $\delta(p, x) \equiv \delta(q, x)$ を得る。
(Moore 機械の場合)　$p \equiv q$ であることから、任意の $x, y \in \Sigma^*$ に対して

$$\mathcal{O}_{\lambda_1}(p, x) = \mathcal{O}_{\lambda_2}(q, x)$$
$$\mathcal{O}_{\lambda_1}(p, xy) = \mathcal{O}_{\lambda_2}(q, xy)$$

語の長さの帰納法によって次が成立することに注意する。

$$\mathcal{O}_{\lambda_1}(p, xy) = \mathcal{O}_{\lambda_1}(p, x)\mathcal{O}_{\lambda_1}(\delta(p, x), y)$$
$$\mathcal{O}_{\lambda_2}(q, xy) = \mathcal{O}_{\lambda_2}(q, x)\mathcal{O}_{\lambda_2}(\delta(q, x), y).$$

したがって

$$\mathcal{O}_{\lambda_1}(\delta(p, x), y) = \mathcal{O}_{\lambda_2}(\delta(q, x), y)$$

となって、状態の等価性の定義より $\delta(p, x) \equiv \delta(q, x)$ を得る。 ■

等価性は状態同士でなく順序機械同士の等価性に拡大できる。

定義 4.5 2 つの順序機械 S_1, S_2 が等価（同値）であるとは、任意の $p \in Q_1$ に対して $p \equiv q$ である $q \in Q_2$ が存在し、かつ任意の $q \in Q_2$ に対して $q \equiv p$ である $p \in Q_1$ が存在するときである。

演習 4.2 順序機械 S_1, S_2 の等価性は同型性ではないことを説明しなさい。等価であっても、その状態数 $|Q_1|$ と $|Q_2|$ は必ずしも同じでない例を挙げなさい。

定義 4.6 順序機械 $M = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$ が既約（irreducible）であるとは、$p, q \in Q$ に関して $p \equiv q$ のとき $p = q$ であるときである。

与えられた順序機械については次が成立することがわかっている。

定理 4.1 任意の順序機械 M に対し、$M \equiv M'$ であるような既約機械 M' が存在する。

証明　$M=(Q,\Sigma,\delta,\Delta,\lambda,q_0)$ に対して、$[p]$ を $p\in Q$ を代表元とする同値類とする。このとき

$$
\begin{aligned}
&Q'=\{[p]\,|\,p\in Q\},\\
&\delta'([p],a)=[\delta(p,a)],\ quada\in\Sigma,\\
&M\text{ が Mealy 機械のとき}\quad \lambda'([p],a)=\lambda(p,a),\\
&M\text{ が Moore 機械のとき}\quad \mathcal{O}_{\lambda'}([p],a)=\mathcal{O}_{\lambda}(p,a)
\end{aligned}
$$

と定めると、補題 4.1 より $p\equiv q$ のとき $\delta(p.a)=\delta(q,a)$ であることから

$$
\begin{aligned}
&M\text{ が Mealy 機械のとき}\quad \lambda'([p],a)=\lambda'([q],a)\\
&M\text{ が Moore 機械のとき}\quad \mathcal{O}_{\lambda'}([p],a)=\mathcal{O}_{\lambda'}([q],a)
\end{aligned}
$$

となって、δ' や λ' における $[p]$ は代表元 $p\in Q$ の選び方によらずに $M'=(Q',\Sigma,\delta',\Delta,\lambda',q_0')$ は M から一意に決まる順序機械となる。この M' において、任意の $x\in\Sigma$ に対して

$$
\begin{aligned}
[p]\equiv[q]\ \rightleftharpoons&\lambda'([p],x)=\lambda'([q],x)\quad M\text{ が Mealy 機械のとき}\\
&\mathcal{O}_{\lambda'}([p],x)=\mathcal{O}_{\lambda'}([q],x)\quad M\text{ が Moore 機械のとき}\\
\rightleftharpoons&\lambda([p],x)=\lambda([q],x)\quad M\text{ が Mealy 機械のとき}\\
&\mathcal{O}_{\lambda}(p,x)=\mathcal{O}_{\lambda'}(q,x)\quad M\text{ が Moore 機械のとき}\\
\rightleftharpoons&p\equiv q\\
\rightleftharpoons&[p]=[q]
\end{aligned}
$$

となって、M に同値な M' は既約となる。 ■

次の定理は既約性の重要性を示す。

定理 4.2 既約な順序機械 $M=(Q,\Sigma,\delta,\Delta,\lambda,q_0)$ の状態数 $|Q|$ は、M と同値な機械 $M'=(Q',\Sigma,\delta',\Delta,\lambda',q_0')\equiv M$ のなかで最小

$$
|Q|=\min\left\{\,|Q'|\quad\middle|\,Q'\text{ は }M'\equiv M\text{ の状態集合}\,\right\}
$$

である。

証明　$|Q|>|Q'|$ とすると、$M\equiv M'$ であることから、$p\neq q$ かつ $p\equiv r$, $q\equiv r$ である $p,q\in Q$ および $r\in Q'$ が存在する。同値性から $p\equiv q$ となり、$p\neq q$ に注意すると M の既約性と矛盾する。 ■

定理 4.2 により、順序機械（有限オートマトン）が与えられたとき、それと同等な最小の状態数を持つ機械を構成するためには既約な機械を見つければよいことがわかった。

4.2 到達可能状態の検出

到達不可能状態は FA の動作に関与しないため、与えられた FA から到達不可能な状態とそれから遷移を取り除いて得られる FA は元の FA とは等価である。

初期状態 q_0 を到達可能状態検出木の根の名前とし、各入力記号によって根から遷移する状態を付け加え、既出の名前を持つ頂点からは伸ばさないようにして到達可能状態検出木を構成する。有限状態機械であるため、この検出木の構成は有限個の頂点を有する木として完了する。

到達不可能状態は、FA の状態集合 Q から到達可能状態を逐次的に取り出した集合 $\mathcal{R}(Q)$ を決定し、残余 $Q\backslash\mathcal{R}(Q)$ の状態として検出できる。このことを保証するの次の命題が成立する。

命題 4.1 初期状態 q_0 から任意の入力 $x \in \Sigma$ に対し $q_0 \overset{x}{\underset{M}{\Rightarrow}}{}^{*}\ p$ によって一意的に状態 p へ到達可能である DFA M について、M の到達可能状態はすべて検出できる。この命題は以下の主張と同等である。

$|x| \leqq n$ である任意の入力 $x \in \Sigma$ に対して、$p_0 \overset{x}{\underset{M}{\Rightarrow}}{}^{*}\ p$ である状態 p は到達可能状態検出木内に名前 p を持つ頂点として存在している。

証明　数 n に関する帰納法によって、命題が成立することを証明する。$n = 0$ の場合、状態 $p = q_0$ は到達可能木の根として存在している。

$n = k \geqq 0$ のときに成立していると仮定して $k+1$ での成立を示す。$|x| = k$ なる $x \in \Sigma$ によって遷移した状態を p としたとき、帰納法の仮定から、名前 p を持つ頂点は到達可能状態検出木の頂点として登場している。長さが $k+1$ の入力列 $y = xa \in \Sigma$ に対して $\delta(p, a) = p'$ とすると、名前 p' を持つ頂点は名前 p を持つ検出木の子頂点として検出木に登場していなければならない。したがって、長さ $k+1$ の任意の入力列についての命題が成立していることが示された。

例 4.1 図 4.1 左の機械 M において、状態 s には初期状態 q_0 からはいかなる入力によっても到達することができない。このために M から状態 s を除去しても M が受理する言語が変わることはない。図 4.1 右の機械 M' は、M から到達不可能状態 s を取り去って

得られる等価な機械である。

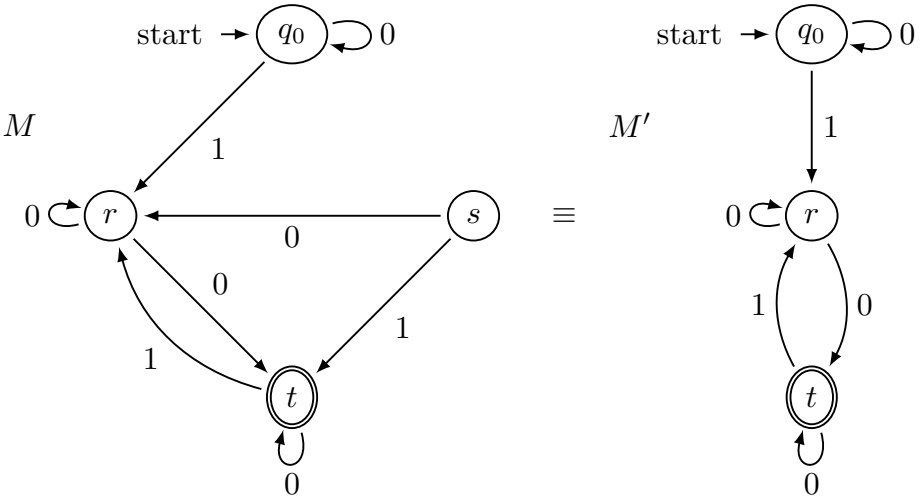


図 4.1　M の到達不可能状態 s を見いだし除去して得られる同じ言語を受理する M'

実際に機械 M の到達不可能状態が s であることを検出するための到達可能状態検出木は図 4.2 のように構成できる。初期状態 q_0 を検出木の根として⓪番目に決定し、入力 0 によって遷移する状態 q_0 を①番目、入力 1 によって遷移する状態 r を②番目として決定する。これを続けて、②番目の状態 r から 1 で遷移する④番目の r は既に②で登場しているため、④番目から検出木の枝は伸びない。また、③番目の状態 t から 0 で遷移する⑤番目の t と,1 で遷移する⑥番目の r も検出木に既出であるので、これ以上の枝を伸ばすことができずに到達可能状態検出木が完成する。この検出木の頂点から定まる集合が到達可能状態 $\mathcal{R}(Q) = \{q_0, r, t\}$ となる。到達不可能状態は $Q = \{q_0, r, s, t\}$ から到達可能状態を差し引いた $Q \backslash \mathcal{R}(Q) = \{s\}$ として求めることができる。

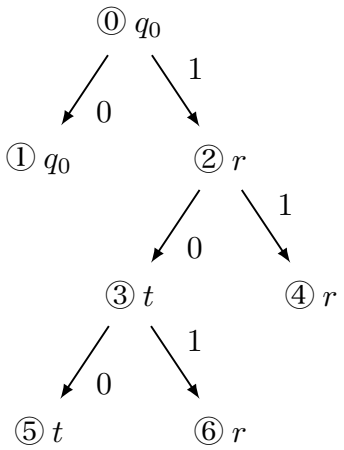


図 4.2　図 4.1 の機械 M の到達可能状態検出木。初期状態 q_0 を 0 番目の根として、入力 0,1 によって遷移する状態を下方に伸ばして到達可能状態検出木を構成。0 番目から 6 番目以降では新たな状態遷移パターン $p \overset{x}{\underset{M}{\Rightarrow}} q$ となる頂点は存在せず、到達可能状態検出木が完成して、この検出木の頂点からなる集合として到達可能状態 $\mathcal{R}(Q) = \{q_0, r, t\}$ が定まる。

4.3　2 つの機械の等価性判定

2 つの DFA M_1 と M_2 が等価であることを判定するアルゴリズムを考えよう。定義 4.1 から、同じ入力アルファベット Σ を持つ 2 つの DFA $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ と $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ が等価 $M_1 \equiv M_2$ で同じ言語を受理する $L(M_1) = L(M_2)$ であるためには、任意の入力列 $x \in \Sigma^*$ に対して

$$q_{01} \overset{x}{\underset{M_1}{\Rightarrow}} p_1 \quad \text{かつ} \quad q_{02} \overset{x}{\underset{M_2}{\Rightarrow}} p_2$$

である状態 $p_1 \in Q_1$ と $p_2 \in Q_2$ が、共に受理状態 $p_1 \in F_1, p_2 \in F_2$ であるか、または共に非受理状態 $p_1 \notin F_1, p_2 \notin F_2$ であらねばならない。もし遷移先の状態 p_1, p_2 で片方が受理状態で他方が非受理状態となるような入力列 x が存在すれば、$L(M_1) \neq L(M_2)$ となってしまう。$M_1 \equiv M_2$ であるとき、定義 4.2 より共通の入力列による遷移先の状態 p_1 と p_2 は等価 $p_1 \equiv p_2$ である。

機械 M_1 と M_2 の等価性判定は、共通の入力列によって到達する状態対について片方が受理状態、もう一方が非受理状態であるような状況が生じないことを確認すればよい。

このために、例 4.1 で到達可能状態検出木を構成したように、以下のようにして等価性判定木 (comparision tree) を構成する。確認すべき状態対は高々 $|Q_1| \times |Q_2|$ 個と有限で

あることから、この手続きは常に完了する。

まず初期状態 q_{01} と q_{02} とは等価 $q_{01} \equiv q_{02}$ として等価性判定木の根の名前とする。各入力記号によって根の 2 状態 q_{01} および q_{02} から遷移する 2 状態が等価であることを調べて $\equiv$ 記号で書く。この手続きを続けて新たに遷移した 2 状態が等価であることを調べながら等価性判定木を伸ばしていく。その際、既出の 2 状態の等価式を名前として持つ頂点が現れた場合には、そこからは枝を伸ばさないようにすると、それ以上新たな 2 状態の等価式が生じないような木として等価性判定木が完成する。

例 **4.2** 図 4.3 で与えられる機械 M_1 と M_2 の等価性を判定してみよう。

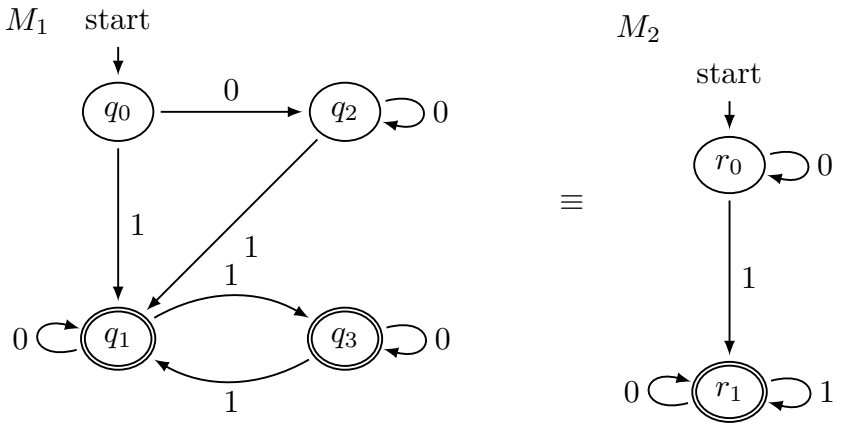


図 4.3　M_1 と M_2 は入力記号 0,1 からなる同じ言語を受理する同等な機械

まず、状態 q_i に到達する言語の正規表現 $\boldsymbol{R}_i$ に関する連立線形再帰式を考えて、M_1 と M_2 で受理される言語の正規表現を求めておこう。M_1 については

$$\begin{aligned}
\boldsymbol{R}_0 &= \boldsymbol{\varepsilon} \\
\boldsymbol{R}_1 &= \boldsymbol{R}_0\mathbf{1} + \boldsymbol{R}_1\mathbf{0} + \boldsymbol{R}_2\mathbf{1} + \boldsymbol{R}_3\mathbf{1} \\
\boldsymbol{R}_2 &= \boldsymbol{R}_0\mathbf{0} + \boldsymbol{R}_2\mathbf{0} \\
\boldsymbol{R}_3 &= \boldsymbol{R}_1\mathbf{1} + \boldsymbol{R}_3\mathbf{0}.
\end{aligned}$$

$\boldsymbol{R}_0 = \boldsymbol{\varepsilon}$ を使って、$\boldsymbol{R}_2$ について解くことができて $\boldsymbol{R}_2 = \mathbf{00}^*$、これより $\boldsymbol{R}_3 = \boldsymbol{R}_1\mathbf{10}^*$ を得る。よって、$\boldsymbol{R}_1 = \mathbf{1}+\boldsymbol{R}_1\mathbf{0}+\mathbf{00}^*\mathbf{1}+\boldsymbol{R}_1\mathbf{10}^*\mathbf{1}$. これを解いて、$\boldsymbol{R}_1 = (\mathbf{1}+\mathbf{00}^*\mathbf{1})(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*$ を得る。したがって、$\boldsymbol{R}_3 = (\mathbf{1}+\mathbf{00}^*\mathbf{1})(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*\mathbf{10}^*$. これより M_1 が受理する言語の正規表現は $\boldsymbol{R}_1 + \boldsymbol{R}_3 = (\mathbf{1}+\mathbf{00}^*\mathbf{1})(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*(\boldsymbol{\varepsilon}+\mathbf{10}^*)$ で与えられる。

M_2 については

$$\boldsymbol{R}_0 = \boldsymbol{\varepsilon} + \boldsymbol{R}_0\mathbf{0}$$
$$\boldsymbol{R}_1 = \boldsymbol{R}_0\mathbf{1} + \boldsymbol{R}_1(\mathbf{0}+\mathbf{1}).$$

これを解いて、M_2 が受理する言語の正規表現は $\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^*$ となる。

正規表現に関する関係式、特に Kleene の閉包演算の性質（演習 3.1）に注目すると、この 2 つの機械が受理する言語の正規表現が一致することを示すことができるが、見通しが芳しくない。これについては、節 4.6.1 の有限オートマトンの最小化で、統一的な方法を考えることにする。

さて、図 4.4 にあるように、⓪番目の等価式 $q_0 \equiv r_0$ から入力によって遷移した状態の等価性（共に受理状態であるか、または共に非受理状態になっている）を調べてみると①番目、②番目の等価式を得る。この操作は、以降③番目から⑧番目まで続けられるが、③、④番目や⑦, ⑧番目からは新たな等価式を生じないことが確認でき、等価性判定木は⓪番目から⑧番目の頂点を持つ木として構成できる。その結果、機械 M_1 と M_2 の等価性が示される。

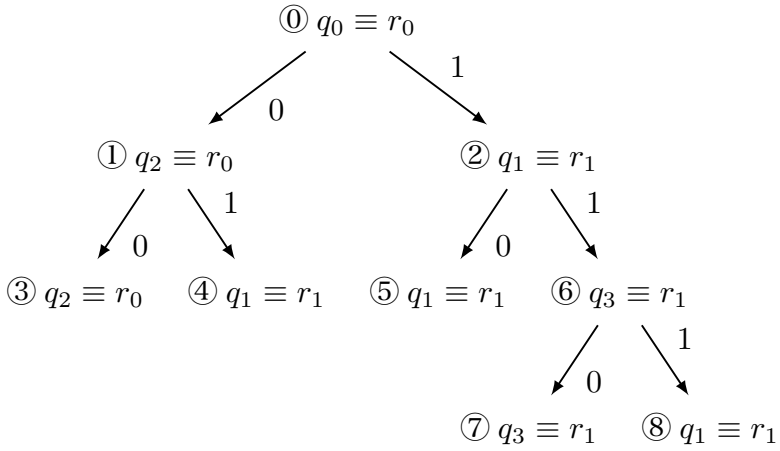


図 4.4　等価性判定木。図 4.3 の機械 M_1 の初期状態 q_0 と M_2 の初期状態 r_0 を 0 番目の等価性 $q_0 \equiv r_0$ として根とし、入力から遷移する等価式を以降同様にして 1 番目から 8 番目として得られる木からは新たな等価式は生じず、等価性判定木が得られる。

例 **4.3** M_2（図 4.3 右）は、次の M_3（図 4.5 右）とは等価ではない $M_2 \not\equiv M_3$ ことを示すことができる。

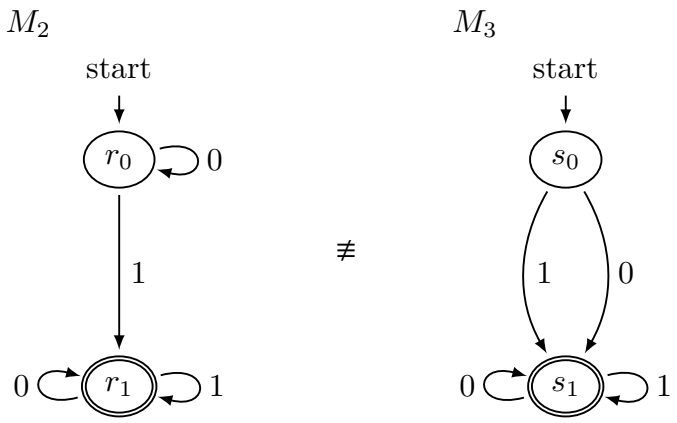


図 4.5　M_2 と M_3 は入力記号 0,1 からなる異なる言語を受理する機械

図 4.6 のように、機械 M_2 の初期状態 r_0 と M_3 の初期状態 s_0 を 0 番目の等価式 $r_0 \equiv s_0$ を根として、入力に応じた遷移状態を考える。$\delta_2(r_0, 0) = r_0 \notin F_1$ と非受理状態に遷移する一方、$\delta_3(s_0, 0)s_1 \in F_3$ と受理状態に遷移することがわかり、$r_0 \not\equiv s_1$ である。したがって、$M_2 \not\equiv M_3$ であることが判定できる。

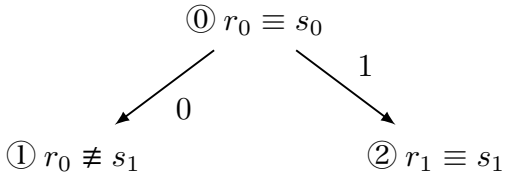


図 4.6　非等価性の判定木。図 4.5 の機械 M_2 の初期状態 r_0 と M_3 の初期状態 s_0 を 0 番目の等価式 $r_0 \equiv s_0$ を根とし、0,1 入力からの遷移状態についての等価性を調べてみる。1 番目の頂点の等価性は $r_0 \not\equiv s_1$ であることから、$M_2 \not\equiv M_3$ であることが判定できる。

実際、M_2 が受理する言語の正規表現 $\boldsymbol{R}_2$ は $\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^*$ である。一方、M_3 の各状態で遷移する言語の正規表現は、節 3.4 の線形再帰方程式の方法に従って

$$\boldsymbol{S}_0 = \boldsymbol{\varepsilon}, \quad \boldsymbol{S}_1 = \boldsymbol{S}_0(\mathbf{0}+\mathbf{1}) + \boldsymbol{S}_1(\mathbf{0}+\mathbf{1})$$

の関係にあることから、M_3 が受理する言語の正規表現は $(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})^* = \mathbf{0}(\mathbf{0}+\mathbf{1})^* + \mathbf{1}(\mathbf{0}+\mathbf{1})^*$ となる。したがって、$\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^* \neq (\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})^*$、つまり $M_2 \not\equiv M_3$ である。たとえば、1 個以上の 0 からなる言語 $\mathbf{0}^\dagger$ は M_2 では受理されないが（1 が入力されて受理状態 r_1 に遷移する必要がある）、M_3 では受理される（1 つ以上の 0 または 1 が入

力されれば受理状態 s_1 に遷移する)。

演習 4.3 図 4.1 の機械 M と M' が等価であることを等価性判定木を構成することによって確かめなさい。

4.4 状態対の等価性判定

状態対の等価性定義 4.2 から、機械 M の任意の状態対 p, q に対して、それらが共に受理状態であるか、または共に非受理状態であるときに同値であるといい $p \equiv q$ と記す。節 4.3 の機械の等価性判定木を構成したようにして、M に属する任意の状態対の等価性式を根とする等価性判定木を構成することができる。

図 4.7 で示されている 2 つの機械 M_1 と M_4 における状態の等価性を調べてみよう。直感的には、$M_1 \not\equiv M_4$ であり、2 つの機械は異なる言語を受理する。

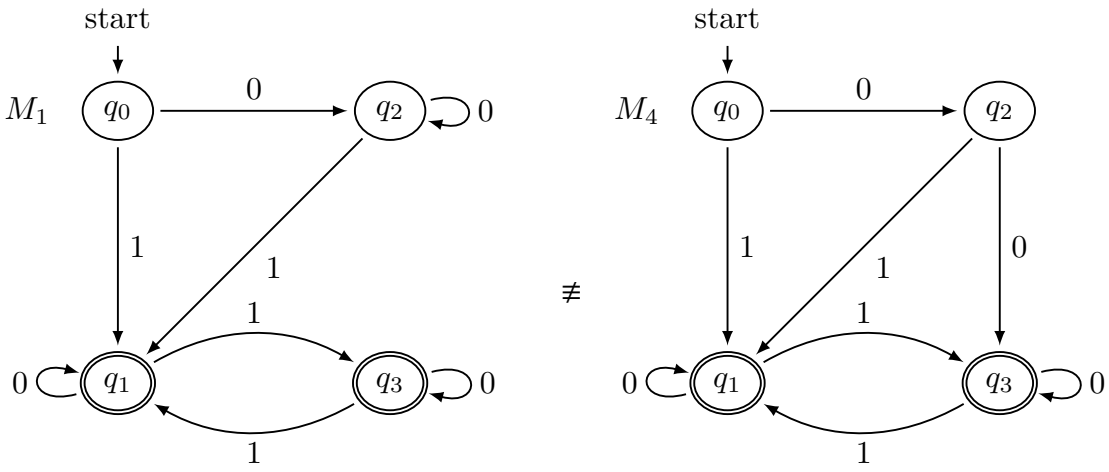


図 4.7　等価な状態を有する機械 M_1 と M_4

M_1 と M_4 において、非受理状態の集合 $\{q_0, q_2\}$ と受理状態の集合 $\{q_1, q_3\}$ に属する状態対は明らかに等価ではないことから、いずれの機械においても状態対 q_0, q_2 と q_1, q_3 の等価性を調べればよい。

演習 4.4 機械 M_4 が受理する言語の正規表現を求めて、M_1 が受理する言語の正規表現と比較してみなさい。

図 4.8 は M_1 の 2 つの状態対を根としてその等価性判定木を構成した様子である。

q_0, q_2 と q_1, q_3 の遷移から得られる等価性判定木の頂点に全て真なる等価性式が得られることから、2 つの状態対は等価であることが判定できる。これによって、2 つの状態対を 1 つにまとめても機械が受理する入力文字列は変わらないことがわかる。こうして同じ言語を受理する簡単化された機械 M_1' が図 4.9 で、例 4.2 において M_1 と等価だと示した M_2（図 4.3）と同型である（状態名称について 1 対 1 の対応がある）。節 3.3 の正規表現の表記では、M_1' したがって M_1 は正規言語 $\mathbf{0}^*(\mathbf{0}+\mathbf{1})^*$ を受理する。

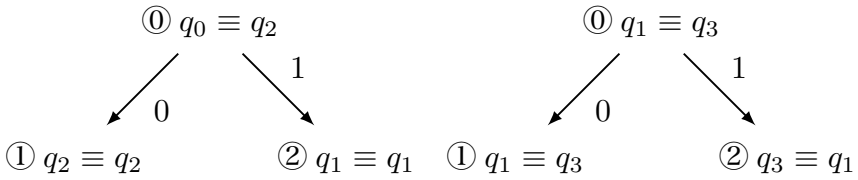


図 4.8　機械 M_1 の状態対の等価性。状態 q_0 と q_2 および q_1 と q_3 は共に等価であり、機械 M_1 は 2 つの等価な状態対を 1 つにまとめて同じ言語を受理する機械 M_1' を構成することができる。

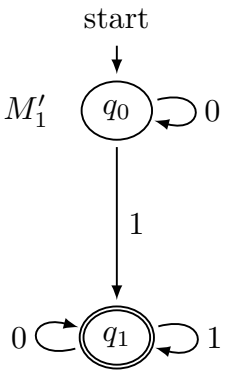


図 4.9　M_1 の 2 組の状態 q_0 と q_2 および q_1 と q_3 をまとめて得られる等価な簡略化機械 M_1'（図 4.3 の M_2 と同型）

図 4.10 は M_4 の 2 つの状態対を根としてその等価性判定木を構成した様子である。q_0, q_2 の遷移から得られる等価性判定木の頂点 q_2 と q_3 は等価でなく $q_2 \not\equiv q_3$、と q_1, q_3 の遷移からは等価性式 $q_1 \equiv q_1$ が得られる。このことから、q_0, q_2 は等価でなく、q_1, q_3 のだけが等価であることが判定できる。

これによって、1 つの状態対 q_1, q_3 を 1 つにまとめても機械が受理する入力文字列は変わらないことがわかる。こうして同じ言語を受理する簡単化された機械 M_4' を図 4.11 に

示す。節 3.3 の正規表現の表記では、M_4' したがって M_4 は正規言語 $(\mathbf{1}+\mathbf{0}(\mathbf{0}+\mathbf{1}))(\mathbf{0}+\mathbf{1})^*$ を受理する。

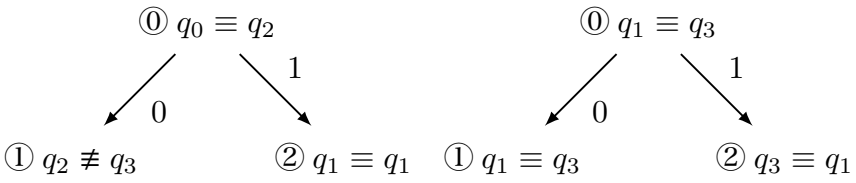


図 4.10　機械 M_4 の状態対の等価性。状態 q_0 と q_2 は等価でなく、q_1 と q_3 だけが等価で、機械 M_4 は 1 つの等価な状態対を 1 つにまとめて同じ言語を受理する機械 M_4' を構成することができる。

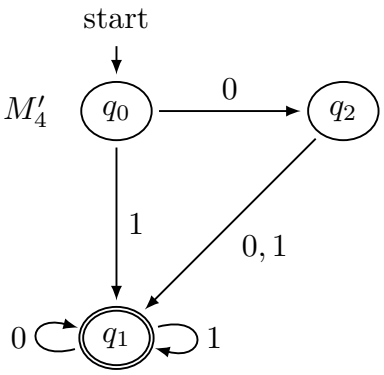


図 4.11　M_4 の 1 組の状態対 q_1 と q_3 をまとめて得られる等価な最小化機械 M_4'

M_1' および M_4' が受理する正規言語について

$$\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^* \not\equiv (\mathbf{1}+\mathbf{00}+\mathbf{01})(\mathbf{0}+\mathbf{1})^*$$

である。実際、M_4' が受理する入力文字列 00 を M_1' は受理しない。したがって、図 4.7 の機械 M_1 と M_4 は等価ではないことが示された。

演習 4.5 図 4.12 の機械 M_a, M_b, M_c は全て同等であることを確かめなさい。この 3 つの機械が受理する言語の正規表現の同等性を示しなさい。

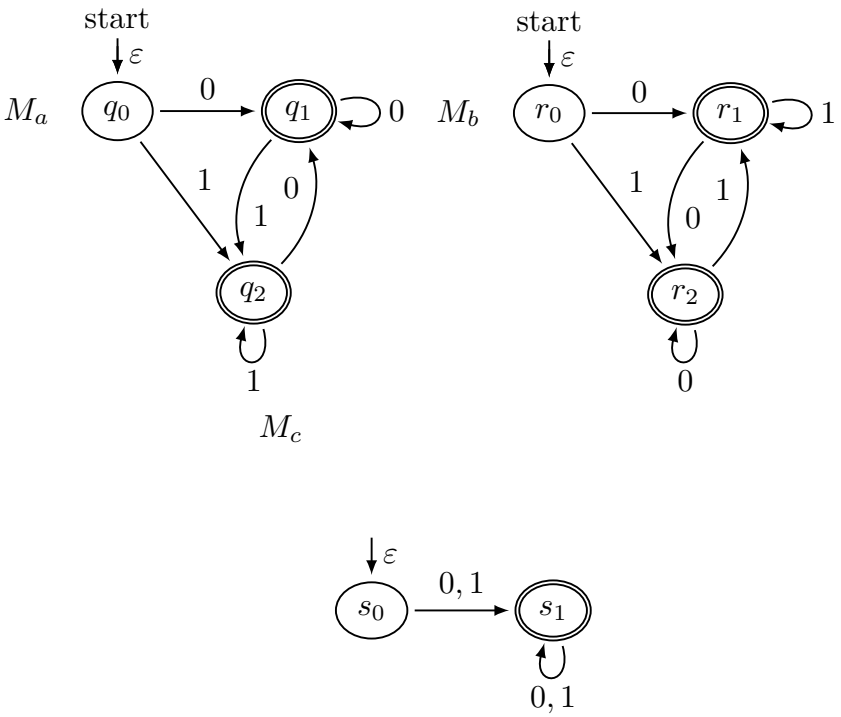


図 4.12　機械 M_a, M_b, M_c は同じ言語を受理する。

4.5　文字列の同値類

オートマトン同士やオートマトン内の状態についての等価性の議論からわかったように、ある言語を受理する DFA は多数ある。1 対 1 対応がある状態名と記号名を付け替えただけで同じ動作をするオートマトンを同型としたとき、同一言語を受理する同型でない DFA が存在する。

では、ある言語を受理する同型でない DFA のうちで状態数が最小であるような有限オートマトンを構成する操作を DFA の最小化 (minimization) または簡略化 (reduction) という。節 4.4 の状態対の等価性判定は与えられた DFA を最小化する 1 つの方法を示していた。DFA M を最小化して得られた M' は、同型を除いて一意に定まるだろうか。ここでは、DFA の等価性を入力文字列の同値関係の観点から改めて考えてみよう。

命題 4.2 (入力の同値関係) DFA $M = (Q, \Sigma, \delta, q_0, F)$ において、入力文字列 $x, y \in \Sigma$ によって同じ最終状態に到達することによって M で定まる関係 R_M を

$$x \, R_M \, y \equiv \text{ if } \delta(q_0, x) = \delta(q_0, y) \text{ for } x, y \in \Sigma^*$$

で定義する。R_M は Σ^* 上の同値関係で、その同値類の数は有限個である。

証明　R_M が同値関係であることは次の 3 つの関係を確かめればよい。

(1) 同じ文字列 x は同じ状態に到達して、$x\,R_M\,x$ であるので、R_M は反射的。
(2) y が x と同じ状態に到達して $x\,R_M\,y$ ならば、x は y と同じ状態に到達して $y\,R_M\,x$。よって $x\,R_M\,y \to y\,R_M\,x$ となり、R_M は対称的。
(3) y が x と同じ状態に到達して $x\,R_M\,y$、z が y と同じ状態に到達して $y\,R_M\,z$ ならば z は x と同じ状態に到達して $x\,R_M\,z$。よって $x\,R_M\,y \wedge y\,R_M\,z \to x\,R_M\,z$ となり、R_M は推移的。

同値関係 R_M の定める同値類は到達状態 $p \in Q$ によって決まる $[\![p]\!]_M$ であり、p に到達する入力語の集合

$$[\![p]\!]_M = \{x \in \Sigma^* \mid \delta(q_0, x) = p\}$$

を定め、$x \in [\![p]\!]_M$ である入力列は全て状態 p に到達する。逆に、状態 p に到達する 1 つの入力文字列 $x \in \Sigma^*$ は関係 R_M によって同値類 $[x]_M$

$$[x]_M = \{y \mid x\,R_M\,y, y \in \Sigma\}$$

を定めて $[\![p]\!]_M$ に一致する。したがって、1 つの状態 p に対応する入力語集合の同値類 $[\![p]\!]_M$ と p に到達する入力列 x を代表元とする同値類 $[x]_M$ と 1 対 1 に対応する。

同値関係 R_M の定める同値類の個数は DFA M の状態の個数に等しく有限で、入力文字列全体 Σ^* は有限個の同値類に直和分割される。

$$\Sigma^* = \bigoplus_{p_i \in Q} [\![p_i]\!]_M.$$

■

命題 4.3 DFA $M = (Q, \Sigma, \delta, q_0, F)$ の関係 R_M は、$x\,R_M\,y$ のとき任意の $z \in \Sigma^*$ に対して $xz\,R_M\,yz$ が成立する。

証明　$x\,R_M\,y$ なら入力 x と y は同じ状態 $\delta(q_0, x) = \delta(q_0, y)$ に到達しているため、そこから任意の $z \in \Sigma^*$ が入力されたとしても再び同じ状態 $\delta(q_0, xz) = \delta(q_0, yz)$ に遷移する。したがって、$xz\,R_M\,yz$ である。 ■

定義 **4.7** (右不変性) 関係 R が演算 $\circ$ に対して

$$x\,R\,y \quad \Rightarrow \quad x\circ z\,R\,y\circ z$$

であるとき、関係 R は右不変 (right invariant) であるという。DFA M による関係 R_M は入力文字列の連接に関して右不変な同値関係である。

命題 4.2 の証明からわかったように、入力文字列 $x, y \in \Sigma$ の M による最終状態によって定まる DFA M の同値関係 R_M の同値類は有限個で、入力文字列全体 Σ^* を最終状態によって直和分割したものである。言い換えれば、M の受理言語 $L(M)$ はその受理状態 $p_i \in F$ で定まる R_M の同値類 $[\![p_i]\!]_M$ の直和

$$L(M) = \bigoplus_{p_i\in F} [\![p_i]\!]_M$$

で表すことができる。この受理状態で定まる各同値類 $[p_i \in F]_M$ は正規表現で表されている。

例 **4.4** 図 4.13 の左側の機械 M_1 で定義される同値関係 R_{M_1} を考えよう。M_1 の 3 つの状態 q_0, q_1, q_2 に対して、初期状態 q_0 に到達する任意の入力文字列をたとえば ε、q_2 に到達する任意の入力文字列をたとえば代表元 1、q_2 に到達する任意の入力文字列をたとえば代表元 11 として、それぞれの同値類を $[\varepsilon]_{M_1}$, $[1]_{M_1}$, $[11]_{M_1}$ とする。

これらの同値類は M_1 の状態 q_0, q_1, q_2 と 1 対 1 の関係にあり、$[\![q_0]\!]_{M_1} = [\varepsilon]_{M_1}$. $[\![q_1]\!]_{M_1} = [1]_{M_1}$, $[\![q_2]\!]_{M_1} = [11]_{M_1}$ であることに注意する。

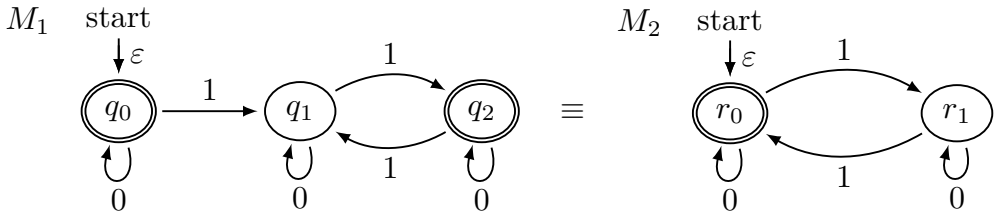


図 4.13　等価な M と M'

これらの同値類は次のような文字列集合をなし

$$[\varepsilon]_{M_1} = \{\varepsilon, 0, 00, \ldots, 0^n, \ldots\},$$
$$[1]_{M_1} = \{1, 10, 100, \ldots, 10^n, \ldots\},$$
$$[11]_{M_1} = \{11, \ldots, 0^i10^j10^k, \ldots, 0^m110^n, \ldots, 0^i10^j10^k10^\ell10^n, \ldots\}$$

$[\varepsilon]_{M_1}$ は記号 1 を含まない 0 個以上の記号 1 からなる集合、$[1]_{M_1}$ は記号 1 を 1 度だけ含む 01 文字列、$[11]_{M_1}$ は記号 1 を 2 度含む 01 文字列である。入力文字列全体 Σ^* はこれらの同値類に直和分割される。

$$\{0,1\}^* = [\varepsilon]_{M_1} \oplus [1]_{M_1} \oplus [11]_{M_1}.$$

M_1 が受理する言語 $L(M_1)$ は、受理状態から定める同値類の直和

$$L(M_1) = [\varepsilon]_{M_1} \oplus [11]_{M_1}$$

で表される。

一方、図 4.13 の右側の機械 M_2 は、演習 4.6 の方法によって M_1 と等価 $M_1 \equiv M_2$ であることを示すことができる。M_1 の場合と同様にして、$[\![r_0]\!]_{M_2} = [\varepsilon]_{M_2}$ および $[\![r_1]\!]_{M_2} = [1]_{M_2}$ を考えることができ

$$\{0,1\}^* = [\varepsilon]_{M_2} \oplus [12]_{M_2}$$

と表すことができるが、M_2 が受理する言語 $L(M_2)$ はその受理状態 r_1 から

$$L(M_2) = [\varepsilon]_{M_2}$$

である。M_1 と M_2 の等価性から、$L(M_1) = L(M_2)$ であることから、

$$[\varepsilon]_{M_1} \oplus [11]_{M_1} = [\varepsilon]_{M_2}, \qquad [1]_{M_1} = [1]_{M_2}$$

となり、R_{M_1} の同値類は R_{M_2} の同値類に含まれる $R_{M_1} \subset R_{M_2}$ ことがわかった。このような事情は、ある機械 M をより少ない状態を持つ機械 M' に簡略化する際に常に生じる。DFA M の言語定義は DFA の状態定義に依存し、一般的には互いに等価な冗長な状態が存在し、等価な状態同士を合併させた状態とする M' を構成すると、対応する同値類も合併されて合併された状態の同値類となる。いいかえると元の M の同値類は M' の細分になる。

演習 4.6 図 4.13 の機械 M_1 と M_2 が等価であることを節 4.3 の等価性判定木を構成して確かめなさい。また、M_1 の状態対 q_0 と q_2 が等価であることを節 4.4 の等価性判定木の方法で確かめ、これらを 1 つにまとめて等価な M_2 が得られることを示しなさい。

4.6 Myhill-Nerode の定理と有限オートマトンの最小化

Myhill-Nerode の定理 4.3 は正則言語を必要十分に特徴づける単一の定理で、これをつかって、ある正規言語を受理する状態数最小の DFA を構成したり、ある言語の非正規性

(どんな有限オートマトンを工夫しても受理できない言語) を証明することができる。

あらためて同値関係を再確認しておく。

定義 4.8 (同値関係) 非空集合 S 上の関係 R とは部分集合 $R \subseteqq S \times S$ で、$(x, y) \in R$ のとき、$x\,R\,y$ を記す。関係 R が同値関係とは次を満たす関係である。

(1) 反射的: $\forall x$ について、$x\,R\,x$.
(2) 対称的: $x\,R\,y$ ならば $y\,R\,x$.
(3) 推移的: $x\,R\,y$ かつ $y\,R\,z$ ならば $x\,R\,z$。

代表元 $x \in S$ を持つ同値類 (equivalence classes)E を $E\{y \,|\, x \,|\, y\} \equiv [x]$ とするとき、同値関係は S をその非連結な同値類に分割する.

$$S = \bigcup_{i \in I} E_i, \qquad E_i \cap E_j = \phi.$$

相異なる同値類の数 $|I|$ を同値関係の指数 (index) といい、指数が有限のとき、R を有限指数という。

S 上の 2 つの同値関係 R_1, R_2 があって、すべての $x, y \in S$ について $R_1 \subseteqq R_2$、つまり $x\,R_1\,y \Leftarrow x\,R_2\,y$ であるとき R_1 は R_2 の細分 (refinement) という。R_1 が R_2 の細分であるとき、R_2 の各同値類は R_1 の同値類のいくつかの差集合になっている。

さて、Σ^* 上の以下の 2 つの同値関係を考える。

定義 4.9 (DFA が定める同値関係) 与えられた DFA $M = (\Sigma, Q, \delta, q_0, F)$ に対して、関係 R_M を次で定義する。

$$x\,R_M\,y = \text{入力 } x \text{ と } y \text{ とが同じ状態に到達: } \delta^*(q_0, x) = \delta^*(q_0, y)$$

R_M の指数は高々 M の状態数 $|Q|$ であることから有限指数。

定義 4.10 (右不変な同値関係) Σ^* 上の同値関係 R が右不変 (left invariant) であるとは、$x\,R\,y$ であるとき、すべての $z \in \Sigma^*$ に対して $xz\,R\,yz$ であることから有限指数。

補題 4.2 Σ^* 上の関係 R_M は右不変な同値関係で、M で受理される言語 $L = L(M)$ は R_M の同値類のいくつかの和集合である。

証明

$$\begin{aligned}\delta(q_0, xz) &= \delta(\delta(q_0, x), z)\\ &= \delta(\delta(q_0, y), z)\\ &= \delta(q_0, yz).\end{aligned}$$

受理状態 q_f に到達する入力列の同値類を $[x]_{q_f}$ と記すと、$L(M)$ で受理状態に到達する入力列全体であることから

$$L(M) = \cup_{q_f \in F}[x]_{q_f}.$$

■

Σ^* 上の任意の言語（正則性は仮定しない）言語 L に対して、もう 1 つの同値関係 R_L が自然に定まる。

定義 4.11 (言語 L が定める Myhill-Nerode 同値関係) 任意の言語 $L \subset \Sigma^*$ に、次のように同値類 R_L を関連付ける。

$x\, R_L\, y$ とは、すべての $z \in \Sigma^*$ について、$xz \in L$ iff $yz \in L$ あるいは $xz \notin L$ iff $yz \notin L$. (4.1)

R_L を Myhill-Nerode 同値関係という。

補題 4.3 L を Σ 上の任意の言語とする。関係 R_L は Σ^* 上の右不変同値関係である。

証明　$x\, R_L\, y$ であると仮定する。このとき、$xu \in L$ iff $yu \in L$ である。$u = zv$ として

$$(xz)v \Leftrightarrow x(zv) \in L \Leftrightarrow y(zv) \in L \Leftrightarrow (yz)v \in l.$$

丁寧に R_L が右不変（定義 4.7）な同値関係であることを見るには、$x, y, z \in \Sigma$ について次の 3 つを確かめればよい。

(1) $x\, R_L\, x$ の成立は明かであるので R_L は反射的。
(2) xw, yw が共に L に属するか属さなければ、yw, xw も共に L に属するか属さないことから $x\, R_L\, y \to y\, R_L\, x$ となって、R_L は対称的。
(3) xw, yw が共に L に属するか属さなくて、さらに yw, zw も共に L に属するか属さないがであれば xw, zw は共に L に属するか属さない。よって、$x\, R_L\, y \wedge y\, R_L\, z \to x\, R_L\, z$ となり、R_L は推移的。

これより、同値関係 R_L は、$x\,R_L\,y$ なら任意の w の連接についても $xw\,R_L\,yw$ であることがわかり、したがって R_L は右不変である。 ■

任意の言語 L に関連付けられた同値関係 R_L は $Sigma^*$ を同値類に分割し、L は L の要素の対応した同値類の和集合担っている。

補題 4.4 $L \subseteqq \Sigma^*$ を任意の言語、E を任意の右不変な Σ^* 上の同値関係で L が E のいくつかの同値類の和集合になっているとする。

このとき、E は Myhill-Nerode 同値関係 R_L の細分である。

証明　$x\,E\,y$ と仮定する。このとき、すべての $z \in \Sigma^*$ に対して $xz\,E\,yz$. L の E のいくつかの同値類の和としたので、$xz \in L$ iff $yz \in L$. よって、$x\,R_L\,y$. ■

補題 4.4 の特別な場合として、L が正規言語であるときには、L を受理する DFA M で定まる同値関係 R_M を考えると次が成り立つ。

命題 4.4 L を DFA M が定める言語 $L(M)$ としたとき、任意の $x, y \in \Sigma^*$ について $x\,R_M\,y$ であれば $x\,R_L\,y$ が成立、R_M による同値類は R_L の細分である。

証明　$L = L(M)$ を仮定すると、$w \Leftrightarrow \delta^*(q_0, w) \in Q_F$. また、$x\,R_M\,y$ と仮定すると、$\delta^*(q_0, x) = \delta^*(q_0, y)$.

いま、$z \in \Sigma^*$ とする。あきらかに $\delta(q_0, xz) = \delta^*(q_0, yz)$. したがって、

$$\begin{aligned} xz \in L &\Leftrightarrow \delta^*(q_0, xz) \in Q_F \\ &\Leftrightarrow \delta^*(q_0, yz) \in Q_F \\ &\Leftrightarrow yz \in L. \end{aligned}$$

これより、$x\,R_L\,y$.

したがって、x の R_M による同値類 $[x]_M$ は、R_M による同値類 $[x]_L$ との関係において $[x]_M \subset [x]_L$ ■

次の定理 4.3 は言語 L が正規言語であるための必要十分条件を与えている。

定理 4.3 (Myhill-Nerode) L を Σ 上の言語とする。以下の命題は互いに同等である。

(1) L は正規言語である。

(2) L は、有限指数を持つ右不変同値関係で定まる同値類のある直和集合として表される。

(3) R_L を Myhill-Nerode 同値関係とすると、R_L の指数は有限である。

証明　(1)→(2)

L が正規言語とすると、L を受理する DFA$M = (Q, \Sigma, \delta, q_0, F)$ が存在する。補題 4.2（命題 4.4）から、L は右不変な有限指数を持つ同値類 R_M の同値類のいくつかの和集合になっている。

(2)→(3)

補題 4.4 から、E は R_L の細分。このとき、R_L の指数は E の指数以下。E の指数が有限であることより、R_L の指数。

(3)→(1)

L を受理する DFA $M' = (\Sigma, Q', \delta', q_0', F')$ を構成する。(2) のいうところより、

$$Q' = \{[x]_L \,|, x \in \Sigma\}.$$

である。

$$q_0' = [\varepsilon]_L,$$
$$\delta'([x]_L, a) = [xa]_L,$$
$$F' = \{[x]_L \,|\, x \in L\}$$

と定める。δ' は代表元 x の選び方に依らない。実際、R_L の右不変性から $x\, R_L\, y$ であるとき、$xa\, R_L\, ya$ となって $[xa]_L = [ya]_L$ である。さて、長さ $|x$ の帰納法より

$$\delta'(q_0', x) = \delta'([\varepsilon]_L, x) = [\varepsilon x]_L = [x]_L, \qquad \text{すべての } x \in \Sigma^*.$$

これより、$x \in L(M') \Leftrightarrow [x]_L \in F'$. したがって、有限オートマトン M' は L を受理し L が正規言語であること (1) がわかる。 ■

4.6.1　有限オートマトンの最小化

n 個の状態 Q_{NFA} を持つ NFA は一般に高々 2^n 個（Q_{NFA} の部分集合の個数）を持つ DFA で模倣できた。Myhill-Nerode の定理から $2^n (n \geqq 2)$ は最良値で、各正則言語に対して最小の状態数をもつ DFA は唯一つであることが分かる。

定理 4.4 (最小有限オートマトン) 正規言語 L を受理する最小の状態数を持つ DFA は（名前を付け替えた）同型を除いて Myhill-Nerode の定理 4.3 の証明 (3) で構成した M' ただ一つに定まる。

証明　定理 4.3 の証明で、正規言語 L を受理する DFA $M = (\Sigma, Q, \delta, q_0, F)$ によって定義される同値関係 R_M は R_L の細分で、M の状態数 $|Q|$ は M' の状態数 $|Q'|$ 以上 ($|Q| \geqq |Q'|$) である。

$|Q| = |Q'|$ のとき、M の各状態は M' の状態に次のように 1-1 に対応付けることができる。各状態 $q \in Q$ に対して $\delta(q_0, x) = q$ となる $x \in \Sigma$ が存在し（そうでなければ q は Q から除去されるべきで、より状態数の少ない FDA があることになる)、q を M' の状態 $\delta'(q'_0, x) \in Q'$ に対応させればよい。実際、$\delta(q_0, x) = \delta(q_0, y) = q$ であるとき、x と y はおなじ同値類 $[x]_M$ に含まれ、M' において $\delta'(q'_0, x) = \delta'(q'_0, y)$ となっている。 ■

4.6.2　最小有限オートマトン

与えられた DFA M は正規言語 L を定めるが、M は最小の状態数を持つわけではない。Myhill-Nerode の定理 4.3 は、M の状態が定める同値関係 R_M による同値類は、正規言語 L が定める同値関係 R_L による同値類の細分になっていることを明らかにしている。M が最小でないときには、少なくとも 2 状態 $p, q \in Q$ が存在して $\{x \in \Sigma^* \mid \delta(q_0, x) = p\}$ と $\{y \in \Sigma^* \mid \delta(q_0, x) = q\}$ の 2 つの同値類は R_L のある同値類に含まれている。

そこで、M の状態間に同値関係を導入して M' が定める状態数に縮約することを考える。2 状態 p, q の状態が区別不能 (indistinguishable) を、すべての $z \in \Sigma^*$ に対して

$$p \equiv q \Leftrightarrow \delta(p, z) \in F \text{ iff } \delta(q, z) \in F, \quad z \in \Sigma^* \tag{4.2}$$

と定めて、$p \equiv q$ と記す（この $\equiv$ が同値関係であるのは容易)。式 (4.2) は、状態 p に到達する文字列の正規表現 r_p と q に到達する文字列の正規表現を r_q と記し、

$$r_p \equiv r_q \Leftrightarrow \delta(q_0, r_p z) \in F \text{ iff } \delta(q_0, r_q z) \in F, \quad z \in \Sigma^* \tag{4.2'}$$

とも表されることに注意する。

DFA M から初期状態から到達不能な状態を取り去り、各状態に到達可能な状態（正規表現）が属する互いに区別不能な状態の最大集合（$\equiv$ による同値類）を決定して 1 つの状態に縮約し（この過程で ‘区別可能’ な状態（正規表現）が残り)、状態数最小の DFA M' の状態との間で 1-1 対応をつければよい。

したがって、課題は $\equiv$ による同値類を決定するアルゴリズムを求めることである。

定義 4.12 (DFA の最小化アルゴリズム) (0) n^2 個のすべての状態の組 $\{q, p\}$ を同値不明 (無印) とする。実際には、相異なる 2 状態 ($q_i \neq q_j$) の組 $\{q_i, q_j\}_{i \leqq j}$（配置すると下三角をなす）と対角にある同一組 $\{q_k, q_k\}$ の合計 $n \cdot (n+1)/2$ 個について、記号 $a \in \Sigma$ による状態の遷移先を調べればよい。

(1) $q_i, R_L\, q_i$（自分自身の状態は互いに同値）より、対角の組 $\{q_k, q_k\}$ には同値マーク (◯)。

(2) 受理状態 $q_f \in F$（に到達する正規表現 r_f）と非受理状態 $q_{NF} \notin F$（に到達する正規表現 r_{NF}）との組 $\{q_f, q_{NF}\}$（$\{r_f, r_{NF}\}$）は同値ではありえず、同値マーク (×)。

(3) × が未記入の状態組 $\{q_i, q_j\}$ に対して、各記号 $a \in \Sigma$ に関する遷移先の同値性をチェック。遷移先の組 $\{\delta(q_i, a), \delta(q_j, a)\}$ が同値な組への遷移であれば状態組 $\{q_i, q_j\}$ に同値マーク ◯、非同値な組への遷移であれば非同値マーク × (一方の状態遷移が未定義なときも非同値 ×)。ただし、共に状態遷移が未定義のときは同じエラー状態への遷移として ◯。不明なら無印のまま。

(4) マーク無しの各状態組 $\{q_i, q_j\}$ について (2) を繰り返す。

(5) 状態組 $\{q_i, q_j\}$ において、すべての状態遷移について同値 ◯ のとき、結果 $\Rightarrow$ は (右不変) 同値 $\equiv$、途中 1 つでも × があれば結果 $\Rightarrow$ は非同値 ×。途中に同値性が判定不能なときには、結果 $\Rightarrow$ を保留。

(6) 段階 (3) において、最後まで結果 $\Rightarrow\times$ がない（結果が保留でもよい）一連の組をなす状態 (そこに到達する正規表現) は同値関係にあり、それぞれ同じ同値類に属する（そこからの状態遷移は同じになっている)。

(7) 段階 (4) で求めた各同値関係をなす状態をグループ $[q_{i_1}, \ldots, q_{i_k}]$ を改めて 1 つの状態とし、各記号 $a \in \Sigma$ で定まる状態遷移関数 δ' を決めると DFA M' が構成される (すべての同値類に到達可能だとは限らない)。初期状態 q_0 を含む同値類が M' の初期状態。

例 4.5 図 4.14 で与えられる DFA M が受理する言語 L を正規表現として決定し、L を受理する最小の状態数を持つ DFA M' を構成しなさい。

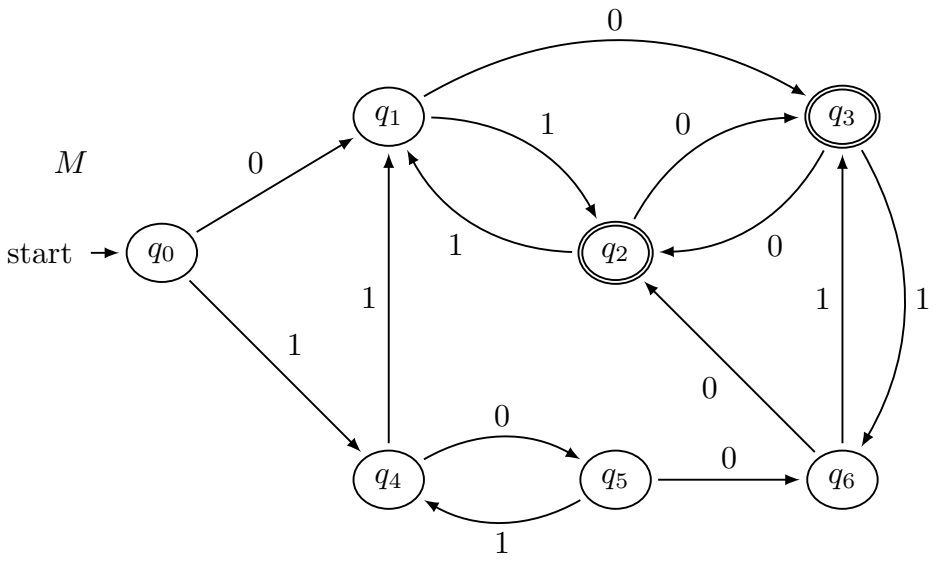


図 4.14　7 状態 $Q = \{q_0.q_1, q_2.q_3.q_4.q_5.q_6\}$ を持つ DFA M.

各状態 $q_i (i = 0, \ldots, 6)$ に到達する正規表現を $r_i (i = 0, \ldots, 6)$ とする。組 (r_k, r_k) は同値 ◯、受理状態 $q_f \in F$ とそれ以外との組 $(q_2, q_j)_{j \neq 2,3}$、$(q_3, q_j)_{j \neq 2,3}$ は同値でない × としておく。

状態の組	0 による遷移と結果	1 による遷移と結果	同値判定
(r_0, r_1)	$(r_00, r_10) = (r_1, r_3)\times$	$(r_01, r_11) = (r_4, r_2)\times$	$\Rightarrow\times$
(r_0, r_4)	$(r_00, r_40) = (r_1, r_5)$	$(r_01, r_41) = (r_4, r_1)$	$\Rightarrow$ 保留 (a)
(r_0, r_5)	$(r_00, r_50) = (r_1, r_6)$	$(r_01, r_51) = (r_4, r_4)\bigcirc$	$\Rightarrow$ 保留 (b)
(r_0, r_6)	$(r_00, r_60) = (r_1, r_2)\times$	$(r_01, r_61) = (r_4, r_3)\times$	$\Rightarrow\times$
(r_1, r_4)	$(r_10, r_40) = (r_3, r_5)\times$	$(r_11, r_41) = (r_2, r_1)\times$	$\Rightarrow\times$ 故に (a) も $\times$
(r_1, r_5)	$(r_10, r_50) = (r_3, r_6)\times$	$(r_11, r_51) = (r_2, r_4)\times$	$\Rightarrow\times$
(r_1, r_6)	$(r_10, r_60) = (r_3, r_2)$	$(r_11, r_61) = (r_2, r_3)$	$\Rightarrow$ 保留 (c)
(r_2, r_3)	$(r_20, r_30) = (r_3, r_2)$	$(r_21, r_31) = (r_6, r_1)$	$\Rightarrow$ 保留 (d)
(r_4, r_5)	$(r_40, r_50) = (r_5, r_6)$	$(r_41, r_51) = (r_1, r_4)\times$	$\Rightarrow\times$
(r_4, r_6)	$(r_40, r_60) = (r_5, r_2)\times$	$(r_41, r_61) = (r_1, r_3)\times$	$\Rightarrow\times$
(r_5, r_6)	$(r_50, r_60) = (r_6, r_2)\times$	$(r_51, r_61) = (r_4, r_3)\times$	$\Rightarrow\times$

表 4.1　DFA の最小化アルゴリズム 4.12 を図 4.14 に適用して得た状態同値性判定の結果。最後まで $\times$ がつかない結果は $\bigcirc$ である。保留 (a) は遷移先 (r_4, r_1) が $\times$ であるために、結果として $\times$ だとわかる。状態の並び $\{q_0, q_5\}$, $\{q_1, q_6\}$ および $\{q_2, q_3\}$ は同値類をなし、のこる状態 q_4 だけで 1 つの同値類を形成する。これら同値類を状態とし、記号 $0, 1$ についての状態遷移を与えて最小の DFA M' が構成できる（図 4.15）。

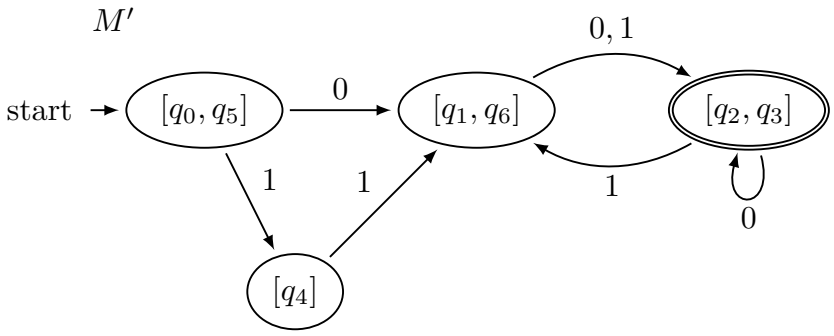
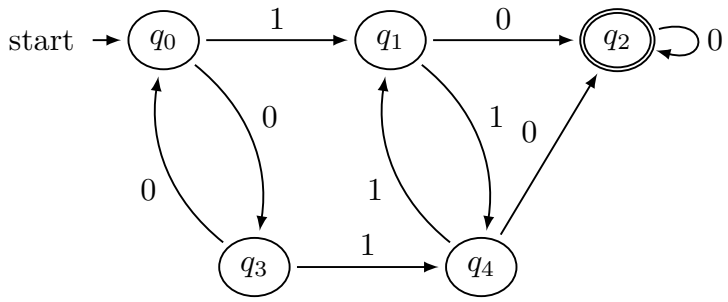


図 4.15　図 4.14 と等価な最小状態数を持つ DFA M'.

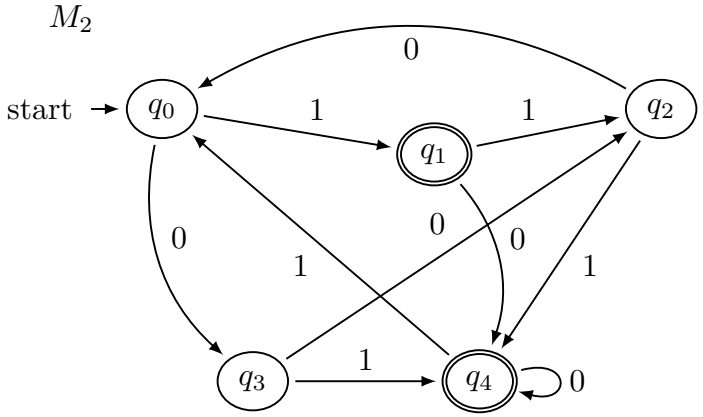
演習 4.7 DFA M_1 が受理する言語 L_1 を正規表現として決定し、L_1 を受理する最小の状態数を持つ DFA M_1' しなさい。

M_1



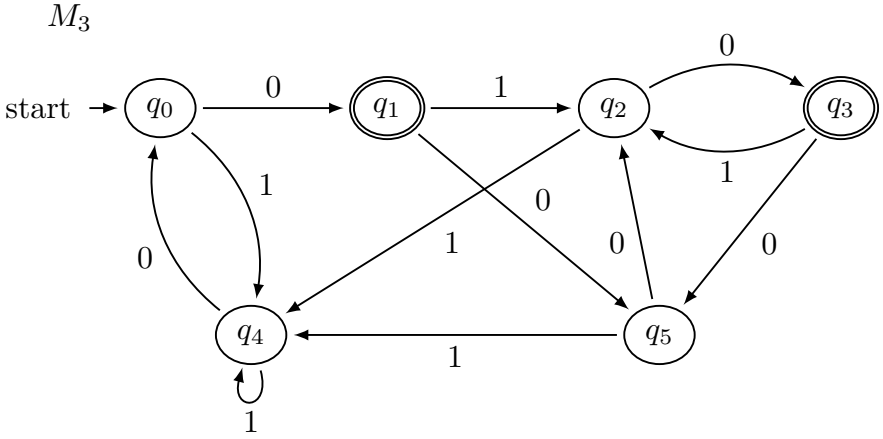
演習 4.8 DFA M_2 が受理する言語 L_2 を正規表現として決定し、L_1 を受理する最小の状態数を持つ DFA M_2' しなさい。

M_2



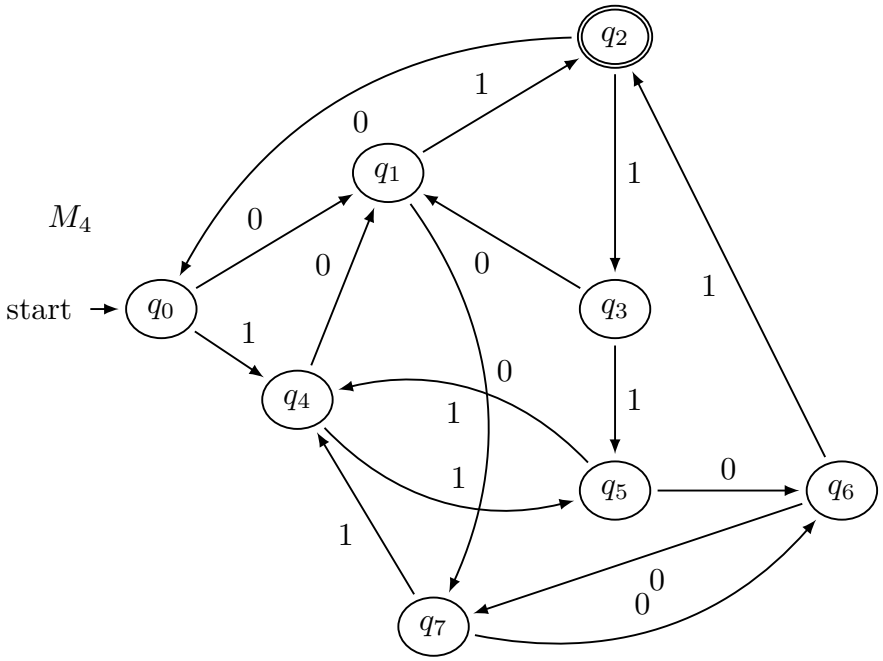
演習 4.9 DFA M_3 が受理する言語 L_3 を正規表現として決定し、L_3 を受理する最小の状態数を持つ DFA M_3' しなさい。を最小化しなさい。

M_3



演習 4.10 DFA M_4 が受理する言語 L_4 を正規表現として決定し、L_4 を受理する最小の

状態数を持つ DFA M_4' しなさい。を最小化しなさい。



4.6.3 Myhill-Nerode の応用

例 **4.6** $\Sigma = \{a, b\}$ 上の言語 $L = \{a^n b^n \mid n \geqq 0\}$ は正規言語ではない。

もし L が正規言語であれば, Myhill-Nerode の定理から、L で定まる右不変な同値関係 R_L による Σ^* の分割は有限指数を持ち、

$$a^j R_L a^k$$

であるような整数 $j < k$ が存在する。このとき、右不変性から

$$a^j b^i R_L a^k b^i$$

となる（$a^j b^i, a^k b^i$ がどちらも L に属するか、またはどちらも L に属さないかいずれかである）。しかし、$a^j b^i \in L$ のとき $a^k b^i \in L$ になり L の定義に反する。したがって、L は正規言語ではあり得ない。

この有名な L の非正規性は正規言語の反復補題 3.6 によっても示すことができる。

節 4.6 の定理 4.3 の証明の (2)→(3) にある式 (**??**) は、証明の (3)→(1) で構成するオートマトンの状態集合 Q' の個数（与えられた有限オートマトン M で受理される言語 L が

定める同値関係 R_L による Σ^* の同値類への分割個数 $|Q'| = |\Sigma^*/R_L|$）は、言語 L を受理する有限オートマトン M の最小値を与えている。

第 5 章

話題: 文字列照合

5.1 文字列照合問題

テキスト T 内に与えられた文字列パターン P が含まれているかどうか、含まれているならどこに何個ふくまれるかという問題が文字列照合（string patching problem）である。

見慣れている日本語文や英文では容易いように思えるが、DNA（デオキシリボ核酸:DeoxyriboNucleic Acid）のような 4 種類のアルファベット $\Sigma = \{A, G, C, T\}$ からなる場合の照合はどうだろうか。地球上の生物の遺伝情報を担う DNA は 2 重螺旋高分子で、A（アデニン)、G（グアニン)、C（シトシン）そして T（チミン）の 4 種類の塩基がかならず A-T と G-C とがペアになって 2 重螺旋をなしている。したがって、二重螺旋の一方の塩基配列が決まると、もう 1 本の塩基配列も自動的に決まり、この 4 文字 A, G, C, T をアルファベットとする文字列が遺伝情報を符号化していると考える。

次は DNA Data BanK of Japan（http://www.ddbj.nig.ac.jp/）で公開されている蟻（LH381539）の DNA データのほんの一部である。

```
gcaagaagca cttctcggat tatcaaatac gtgctcatcg catcgtttca gtgttgtcgt
tatactgctt gacataacat ttgcagacga gtattaatag cgaaacgtct taattgatat
tatgtattgt ggatttgaac gtacaataag gacgcgcgat ctatcatttt gtttgttcat
atattcttct tgagaactta agtaataatt tgcagtagca cagattgcga aaacagaccg
attatttttt catttgaaaa agccagtttg tatttttta tacataagac atttttacat
taattgaact tgcaatttcc tggattgctt ggttatcgat acgattgttt ttgttcttct
```

人の DNA では約 31 億塩基対があるとされている[*1]。また、厳密なパターン照合だけでなく、曖昧照合も実用的意味合いが大きい。さらに、画像や音楽データのパターン照合など多くの研究テーマがある。

文字列 $S = [1..n]$ の k 文字の接頭辞 $S[1..k]$ を S_k と記するとき、$S_0 = \varepsilon$ および $S_n = S$ であることを思い起こしておこう。

定義 5.1 (文字列照合) アルファベット Σ 上のテキスト T は長さ $n = |T|$ の配列 $T = T[1..n] = t_1 t_2 \ldots t_n$、パターン P は長さ $m = |P| < n$ の配列 $P = P[1..m] = p_1 p_2 \ldots p_m$ に対して、パターン P がテキスト T に照合（マッチ）するとは、シフト $0 \leqq s \leqq n - m$ が存在して

$$T[s + 1..s + m] = P[1..m]$$

であることである。文字列照合問題とは次の接尾辞関係

$$P \sqsupset T_{s+m}$$

を満たすシフト $0 \leqq s \leqq n - m$ を見つけることである（図 5.1）。

[*1] ヒトゲノム計画は 1984 年に提案されて、1991 年から解読作業が始まり、2003 年 4 月 14 日に一応の解読完了が宣言されている。ヒトの遺伝子の推定値は現在 2 万 3 千個あまりである（ヒトの高度な複雑さを思うと想像以上に少ない）。さらに驚くべきことは、2006 年にイネ科植物の遺伝子が人より多く、また下等とされるウニの遺伝子数がヒトとほぼ同じで、しかも 2/3 以上がヒトと共通していることが判明したことだ（The Genome of the Sea Urchin Strongylocentrotus purpuratus: http://science.sciencemag.org/content/314/5801/941.abstract）。従来の生物進化に伴う遺伝描像—無限に多様であり得るヒトの遺伝子情報は膨大であり、高度に進化した生物の遺伝子はそれなりに増加し複雑化するなるなど—は大きな変更が迫られている。

いわゆる DNA 鑑定ではヒトゲノムの塩基配列のすべてを調べるわけではない。同じ塩基配列が繰り返して存在する DNA 内の縦列反復配列部位が人によって異なることを利用して個人識別を行う方法が一般的で、同じ形の別人が現れる確率は 0 ではないと算定されており、現段階では絶対的鑑定方法だとは言えないようだ。

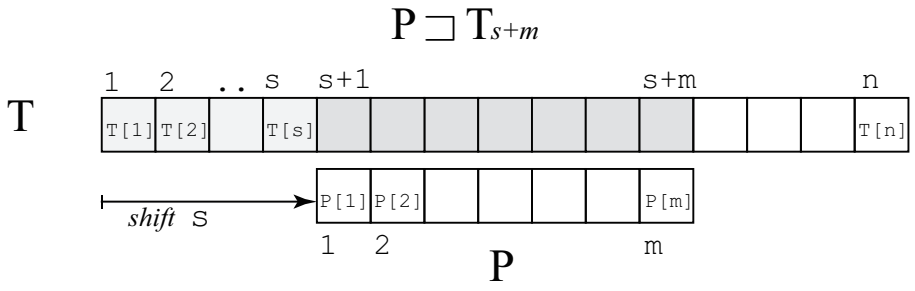


図 5.1　テキスト T の $s+m$ 文字接頭辞 T_{s+m} は m 文字からなるパターン P を接尾辞としてもつ。

5.2　文字照合オートマトン

テキスト T を 1 文字ずつ読み込みながら与えられた文字列パターン P があるかどうか判定する文字列照合オートマトンを考えてみる。目的のオートマトン $M = (Q, \Sigma, \delta, q_0, F)$ はアルファベット Σ からなるテキスト T を入力し、パターン $P = P[1..m]$ を見いだしたときに受理状態となるように構成する。

M の内部状態 Q を

$$Q = \{0, 1, \ldots, m\}$$

とする。その時点で読み込んだテキスト文字列 $T[1..i]$ がその接尾辞としてパターンの長さ q の接頭辞 $P_q = P[1..q]$ としてマッチしてしている $P_q \sqsupset T[1..i]$ ように M の状態 q を定める（$P_0 = \varepsilon$ とする）。また、1 文字もマッチしていない状態が初期状態 0 で、0 からは唯一入力 $P[1]$ によって状態 1 に遷移する。また、パターン $P[1..m]$ にマッチした状態 m は唯一の受理状態集合 F をなす。状態遷移関数 $\delta : Q \times \Sigma \to Q$ を

$$\delta(q, a) = \sigma_P(P_q a), \quad a \in \Sigma$$

で定める。$\sigma_P : \Sigma^* \to Q$ は接尾辞関数で、次のように定義する。

定義 5.2 (パターン P の接尾辞関数 σ_P)

$$\sigma_P(x) = \max_k \{k \mid P_k \sqsupseteq x\}, \qquad x \in \Sigma.$$

$\sigma_P(x)$ の値は、文字列 x の接尾辞であるようなパターン P の最長の接頭辞 P_k の長さである（P_k は x の真の接尾辞である必要はなく $P_k = x$ であっても構わないので $\sqsupseteq$ を使った）。

このとき、次の性質に注意する。

$$\sigma_P(x) = 0 \quad \text{iff } P_k \sqsupset x \text{ である } P_k \text{ が存在しない},$$
$$\sigma_P(x) = m \quad \text{iff } P \sqsupset x,$$
$$\text{if } x \sqsupset y, \quad \sigma_P(x) \leqq \sigma_P(y).$$

$\delta(q, a) = \sigma_P(P_q a)$ による状態遷移は図 5.2 のようになる。

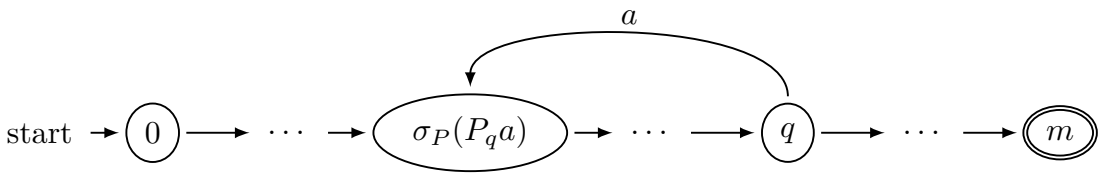


図 5.2　文字列照合オートマトンの状態遷移。パターン $P_q = P[1..q]$ までマッチしている状態 q にあるとき、さらに文字 $a \in \Sigma$ を読み込んだとき接尾辞関数で定まる状態 $\sigma_P(P_q a)$ に遷移する。

接尾辞関数 $\sigma_P : \Sigma^* \to Q$ からわかるように、文字列照合オートマトン M の動作は、パターン $P = P[1..m]$ だけで決定される。

例 5.1 長さ $m = 5$ のパターン $P =$ `c r i c k` を受理する文字列照合オートマトンを構成してみよう [10, p.76]。$q = 0, \ldots, m$ として遷移先の状態 $\delta(q, a), a \in \Sigma$ を求めていく。

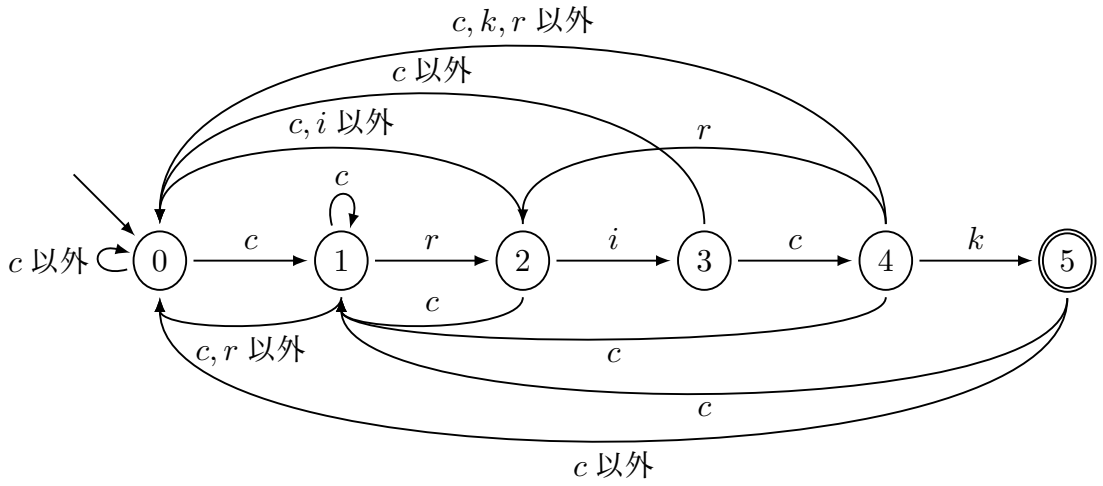


図 5.3　パターン $P =$ `c r i c k` を受理する文字列照合オートマトン

まず、空接頭辞 $P_0 = \varepsilon$ を使って、$\delta(0, c) = \sigma_P(c) = 1$、また文字 $s \neq c$ については $\delta_P(0, s) = \sigma_P(s) = 0$. 次に、$\delta(1, c) = \sigma_P(cc) = 1$, $\delta(1, r) = \sigma_P(cr) = 2$、文字 $s \neq c, r$ については $\delta(1, s) = \sigma_P(cs) = 0$.

同様にして、$\delta(2, i) = \sigma_P(cri) = 3$, $\delta(2, c) = \sigma_P(crc) = 1$、文字 $s \neq c, i$ については $\delta(2, s) = \sigma_P(crs) = 0$.

$\delta(3, c) = \sigma_P(cric) = 4$、文字 $s \neq c$ については $\delta(3, s) = \sigma_P(cris) = 0$. $\delta(4, k) = \sigma_P(crick) = 5$, $\delta(4, c) = \sigma_P(cricc) = 1$、$\delta(4, r) = \sigma_P(cricr) = 2$、文字 $s \neq c, k, r$ については $\delta(4, s) = \sigma_P(crics) = 0$.

最後に、$\delta(5, c) = \sigma_P(crickc) = 1$, 文字 $s \neq c$ については $\delta(5, s) = \sigma_P(cricks) = 0$.

この結果から、パターン $P =$ `c r i c k` を受理する文字列照合オートマトンは図 5.3 のようになる。

一般に、パターン $P = P[1..m]$ の文字列照合オートマトンの遷移関数 δ は次のアルゴリズムから求めることができる。

ソースコード 5.1　文字列照合オートマトンの遷移関数を定めるアルゴリズム

```
1  String-Match-FA-transition(パターン P)
2  m = length(P);
3  for q = 0 to m do
4      for a ∈ Σ do
5          k = min (m + 1, q + 2);
6          repeat do
7                  k = k + 1;
8              until P_k ⊐ P_q a;
9          δ(q, a) = k;
10 return δ
```

演習 5.1 長さ m のパターン $P = P[1..m]$ の文字列照合オートマトンの遷移関数を求めるソースコード 5.1 の計算量は $\mathcal{O}(m^3|\Sigma|)$ であることを示しなさい。

以上から、指定したテキスト T に対して与えたパターン P に関する文字列照合オートマトンを使った判定は次のソースコード 5.2 に従って動作する。

ソースコード 5.2　文字列照合オートマトンの動作アルゴリズム

```
1  String-Match-FA(テキスト T, パターン P)
2  n = length(T);
```

```
m = length(P);
δ = String-Match-FA-transition(P);
q = 0;
for i = 1 to n do
    q = δ(q, T[i]);
    if q == m then
        print パターン P が T 内にシフト i - m でマッチ
```

5.3 文字列照合アルゴリズム

次のコード 5.3 はテキストの先頭からシフト s を 0 から $n-m$ までずらしながらパターン文字列を検査して文字列照合を行うアルゴリズムである。

ソースコード 5.3 素朴な文字列照合アルゴリズム

```
Naive-String-Match(テキスト T, パターン P)
n = length(T);
m = length(P);
for s = 0 to n - m do シフト数を 0からn-m まで変化
    q = 0; マッチ数を 0に初期化
    while q + 1 ≦ m do
        while P[q + 1] == T[i = s + 1 + q] do
            q = q + 1; マッチ数
        if q == m then
            print シフト s でパターンP が見つかった;
if q < m then
    print T にはパターンP を含まない;
```

演習 5.2 与えられたパターン P の長さが $m = |P|$、指定されたテキスト T の長さを $n = |T|$ とするとき、素朴な文字列照合アルゴリズムの計算量は $\mathcal{O}(mn)$ であることを示しなさい。

図 5.4 は、与えたテキスト b a c b a b a b a a b c a b a 内に 7 文字からなる指定したパターン a b a a b c a を照合する場合を示している。先頭から $s = 4$ だけシフトして $T[s+1=5] = a$ からパターン P を照合したとき、$P[1..3]$ までマッチ（マッチ数 $q = 3$）し、次の $P[q+1=4] \neq T[i=8]$ で不一致となった様子である。

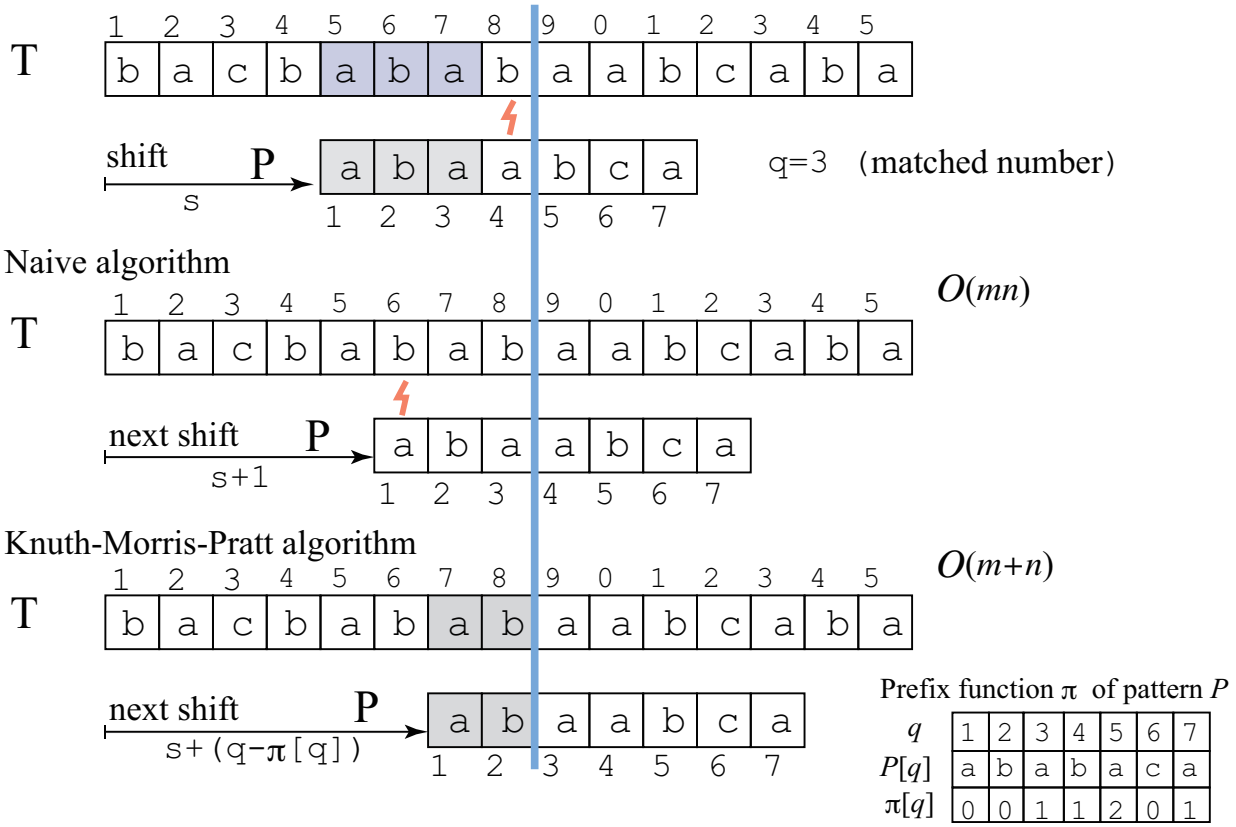


図 5.4　文字列照合と Knuth-Morris-Pratt アルゴリズム

テキスト T を位置 $i = 8$ までスキャンしているとき、パターン P の先頭から $q = 3$ 文字までマッチし、$q + 1 = 4$ 番目の文字で不一致であるとする（図 5.4 の上段）。コード 5.3 の素朴な文字列照合アルゴリズムでは、次にシフト $s = 4$ を 1 文字分増やして $s \to s + 1 = 5$、$T[i = 8]$ とからパターン P の文字 $P[q - 1 = 3]$ との比較を試みる（図 5.4 の中段）。しかし、この +1 だけのシフトには意味があるだろうか。

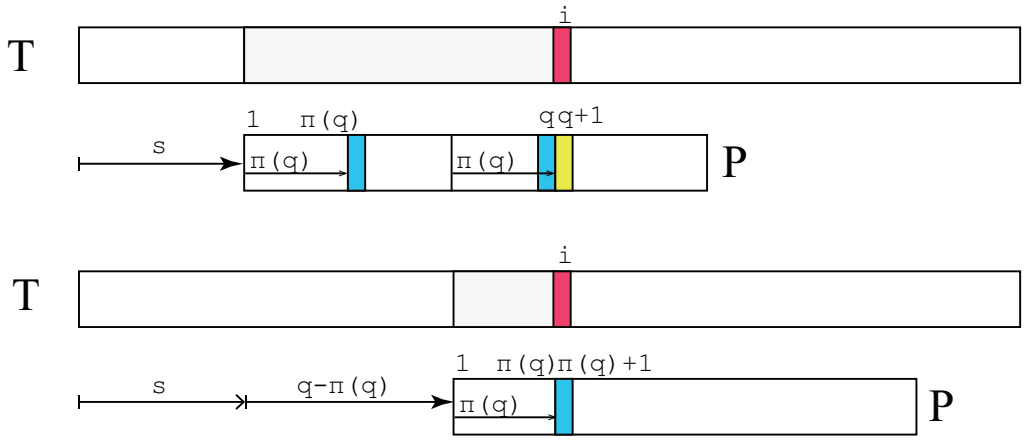


図 5.5　Knuth-Morris-Pratt アルゴリズム。パターン P の接頭辞関数 π を使うと、+1 文字ずつシフトして照合する素朴なアルゴリムに比べて文字列照合を高速化できる。接頭辞 P_q に着目したとき、その接頭辞 $P_k = P[1..k]$ が P_q の同じ長さ k の接尾辞 $P[m+1-k..m]$ に一致しているような最大の k を接頭辞関数値 $\pi(q)$ とする。このとき、次にテキスト文字 $T[i]$（文字種は赤）と照合するパターン P の文字は $P[\pi(q)+1]$（文字種は青）である。

このことについて検討するために、図 5.5 のような場合を考えてみる。図 5.5 では、テキスト T を位置 i までスキャンしているときに（$T[i]$ の文字種は赤）、パターン P の先頭から q 文字までマッチし、$q+1$ 番目の文字で不一致（$P[q+1]$ の文字種は黄）である様子を表している。このとき接頭辞 $P_q = P[1..q]$ に着目したとき、P_q の長さ k の接頭辞 $P_k = P[1..k]]$ が P_q の同じ長さ k の接尾辞 $P[m+1-k..m]$ に一致しているとしよう。このとき、長さ $\pi(q)$ をそのような長さ k の最大長 $\max_k\{k \,|\, k < m \text{ and } P_k \sqsubset P_q\}$ であるとする。$\pi(1) = 0$ と定義しておく。こうした関数 $\pi : \{1, \ldots, m\} \to \{0, \ldots, m\}$ を接頭辞関数という。

そのとき次にすべきことは、テキスト位置 i の文字 $T[i]$ をパターン P を +1 文字ずらして $P[q+1]$ と比較する素朴なアルゴリズムに依るのではなく（$\pi(q)$ の選び方から、結果は不一致となる）、$q - \pi(q)$ ずらして $P[\pi(q)+1]$（文字種は青）と比較することである。$q - \pi(q) > 1$ であれば +1 だけずらして検査する素朴なアルゴリズムに比べて文字列照合を高速化することができる。

5.4 Knuth-Morris-Pratt のアルゴリズム

まず、パターン $P = P[1..m]$ の接頭辞関数 $\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m\}$ を次で定義しておこう。

定義 5.3 (P の接頭辞関数)

$$\begin{aligned}\pi(q) &= P_q \text{ の '真の' 接尾辞となるような } P \text{ の最長の接頭辞の長さ} \\ &= \max_k\{k \mid k < q \text{ and } P_k \sqsupset P_q\}\end{aligned}$$

図 5.6 はパターン P の接頭辞関数値 $\pi(q)$ との関係を示している。パターン P の長さ $q \in \{1, \ldots, m\}$ の接頭辞 P_q に対して、P_q の長さ k の接頭辞 $P_q[1..k]$ が P_q の真の接尾辞 ($P_q[1] \neq P_q[m+1-k]$) でもあるようなものを考える。そのような接頭辞 $P_q[1..k]$ の最大の長さ $K = \max_k\{k \mid P_q[m+1-k] \sqsupset P_q\}$ が接頭辞関数の値 $\pi(q) = K$ である。

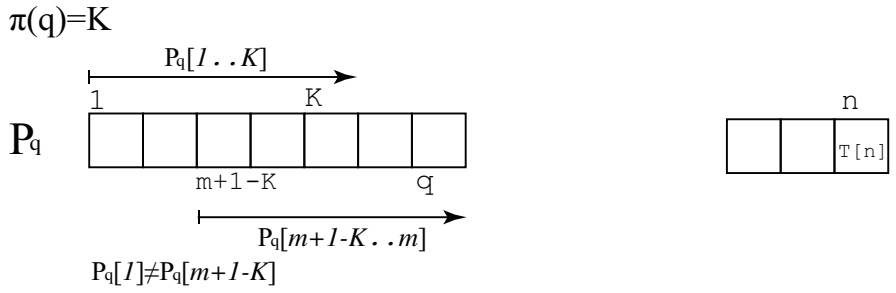


図 5.6　接頭辞関数 $\pi : \{1, \ldots, m\} \to \{0, \ldots, m\}$. パターン P の長さ $q \in \{1, \ldots, m\}$ の接頭辞 P_q において、その長さ k の接頭辞 $P_q[1..k]$ が P_q の真の接尾辞 ($P_q[1] \neq P_q[m+1-k]$) でもあるような最大の長さ K が接頭辞関数の値 $\pi(q) = K$ である。

パターン P の接頭辞関数 π はパターン P だけで決まり、指定するテキスト T には独立である。$P =$ a b a a b c a の場合、次のようになる。

P	a	b	a	a	b	c	a
q	1	2	3	4	5	6	7
$\pi(q)$	0	0	1	1	2	0	1

表 5.1　パターン $P =$ a b a a b c a の接頭辞関数表

パターン P が与えられたとき、その接頭辞関数 π は次のように Prefix-function 関数として計算できる。

ソースコード 5.4　パターン P の接頭辞関数表作成アルゴリズム

```
1  Prefix-function(パターン P)
2  m = length(P);
3  π(1) = 1;
4  k = 0;
5  for q = 2 to m do
6      while k > 0 and P[k + 1] ≠ P[q] do
7          k = π(k);
8      if P[k + 1] == P[q] then
9          k = k + 1;
10     π(q) = k
11 return π
```

演習 5.3 与えられたパターン P の長さが $m = |P|$ とするとき、その計算量は $\mathcal{O}(m)$ であることを示しなさい。

ソース 5.4 でパターン P の接頭辞関数 π を計算しておくと、次の Knuth-Morris-Pratt アルゴリズム 5.5 を使って文字照合を行うことができる [18]。

ソースコード 5.5　Knuth-Morris-Pratt アルゴリズム

```
1  Knuth-Morris-Pratt(テキスト T, パターン P)
2  n = length(T);
3  m = length(P);
4  π = Prefix-function(P);
5  π(1) = 1;
6  q = 0; マッチした文字数
7  for i = 1 to n do テキストを先頭からスキャン
8      while q > 0 and P[q + 1] ≠ T[i] do 次の文字がマッチしない
9          q = π(q);
10     if P[q + 1] == T[i] then
11         q = q + 1; 次の文字がマッチ
12     if q == m then
13         print パターン P がテキスト T を i - m だけシフトして見つかった
14         q = π(q) 次のマッチを探す
```

演習 **5.4** 与えられたパターン P の長さが $m = |P|$、指定されたテキスト T の長さを $n = |T|$ とするとき、Knuth-Morris-Pratt のアルゴリズムの計算量は $\mathcal{O}(m + n)$ であることを示しなさい。

次は与えたテキスト T = b a c b a b a b a a b c a b a 内のパターン P = a b a a b c a を Knuth-Morris-Pratt アルゴリズムを使って照合する過程を表している。括弧内の q の値は i に関する for ブロック終了時にマッチしている文字数を表している。

```
i=1
[1]2 3 4 5 6 7 8 9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
 a b a a b c a    (q=0)
 1 2 3 4 5 6 7
i=2
 1[2]3 4 5 6 7 8 9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
  [a]b a a b c a   (q=1)
   1 2 3 4 5 6 7
i=3
 1 2[3]4 5 6 7 8 9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
     a b a a b c a   (q=0)
     1 2 3 4 5 6 7
i=4
 1 2 3[4]5 6 7 8 9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
       a b a a b c a   (q=0)
       1 2 3 4 5 6 7
i=5
 1 2 3 4[5]6 7 8 9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
        [a]b a a b c a   (q=1)
         1 2 3 4 5 6 7
```

```
i=6
 1 2 3 4 5[6]7 8 9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
        [a b]a a b c a   (q=2)
         1 2 3 4 5 6 7
i=7
 1 2 3 4 5 6[7]8 9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
        [a b a]a b c a   (q=3)
         1 2 3 4 5 6 7
i=8
 1 2 3 4 5 6 7[8]9 0 1 2 3 4 5
 b a c b a b a b a a b c a b a
            [a b]a a b c a   (q=2)
             1 2 3 4 5 6 7
i=9
 1 2 3 4 5 6 7 8[9]0 1 2 3 4 5
 b a c b a b a b a a b c a b a
            [a b a]a b c a   (q=3)
             1 2 3 4 5 6 7
i=10
 1 2 3 4 5 6 7 8 9[0]1 2 3 4 5
 b a c b a b a b a a b c a b a
            [a b a a]b c a   (q=4)
             1 2 3 4 5 6 7
i=11
 1 2 3 4 5 6 7 8 9 0[1]2 3 4 5
 b a c b a b a b a a b c a b a
            [a b a a b]c a   (q=5)
             1 2 3 4 5 6 7
i=12
 1 2 3 4 5 6 7 8 9 0 1[2]3 4 5
 b a c b a b a b a a b c a b a
```

```
            [a b a a b c]a   (q=6)
             1 2 3 4 5 6 7
i=13
 1 2 3 4 5 6 7 8 9 0 1 2[3]4 5
 b a c b a b a b a a b c a b a
            [a b a a b c a]   (q=7) find
             1 2 3 4 5 6 7
i=14
 1 2 3 4 5 6 7 8 9 0 1 2 3[4]5
 b a c b a b a b a a b c a b a
                       [a b]a a b c a   (q=2)
                        1 2 3 4 5 6 7
i=15
 1 2 3 4 5 6 7 8 9 0 1 2 3 4[5]
 b a c b a b a b a a b c a b a
                       [a b a]a b c a   (q=2)
                        1 2 3 4 5 6 7
```

第 6 章

文脈自由言語

6.1 形式文法

形式文法 $G = (\Sigma_N, \Sigma_N, P, S)$ とは、非終端記号集合 Σ_N, 終端記号集合 Σ_T ($\Sigma_N \cap \Sigma_T = \phi$) と開始記号 $S \in \Sigma_N$ と生成規則 P の組で、生成規則の要素は次の形の書き換え規則からなる。

$$P = \left\{ (\Sigma_N \cup \Sigma_T)^\dagger \to (\Sigma_N \cup \Sigma_T)^* \right\}$$

定義 6.1 $\alpha \in \Sigma_N$, $u, v, \beta \in (\Sigma_N \cup \Sigma_T)^*$ において、次のように文形式 (sentential form) を定める。

(1) $S \in \Sigma_N$ は文形式。
(2) 文形式 u, v からなる文形式 $u\alpha v$ が文形式であり、$\alpha \to \beta \in P$ のとき、$u\beta v$ もまた文形式。
(3) 手続き (1), (2) だけで得られるものだけが文形式である。

終端記号だけのみからなる文形式 $w \in \Sigma_T^*$ を文 (sentence) という。

定義 6.2 文法 G において、文形式 $u\alpha v \in (\Sigma_N \cup \Sigma_T)^\dagger$ を書き換え規則 $\alpha \to \beta \in P$ で書き替えて $u\beta v$ を導くことを導出 (derivation) と称し、次のように記す。

$$u\alpha v \underset{G}{\Rightarrow} u\beta v$$

関係 $\underset{G}{\Rightarrow}$ は $(\Sigma_N \cup \Sigma_T)^*$ 上の 2 項関係である。常に文形式内の最左の非終端記号に対して生成規則を適用するとき最左導出、最右の非終端記号に対して生成規則を適用するとき

最右導出という。このとき、u を前文脈 (pre context)、v を後文脈 (post context) と称する。

$(\Sigma_N \cup \Sigma_T)^*$ 上の関係 $\underset{G}{\Rightarrow}$ の反射的推移閉包を $\overset{*}{\underset{G}{\Rightarrow}}$ と記す。$(\Sigma_N \cup \Sigma_T)^*$ 上の文形式列 $w_0, w_1, \ldots, w_n$ が

$$w_0 \underset{G}{\Rightarrow} w_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} w_n$$

の導出関係にあるとき、w_0 は w_n を導出するといい、$w_0 \overset{*}{\underset{G}{\Rightarrow}} w_n$ と表す。

文法 G が与えられたとき、開始記号 S から導出される終端記号 Σ_T 上の記号列 $w \in \Sigma^*$ 全体

$$L(G) = \{w \in \Sigma^* \,\big|\, S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

は文法 G が生成する言語である。

6.2 正規文法

6.3 文脈依存文法

文法 $G = (\Sigma_N, \Sigma_T, P, S)$ が文脈依存文法 (CSG: Context Sensitive Grammar) とは、書き換え規則の有限集合 $P = \{s \to t\}$ が

$$s \in (\Sigma_N \cup \Sigma_T)^\dagger, t \in (\Sigma_N \cup \Sigma_T)^*, \qquad |s| \leqq |t|$$

であって、開始記号 $S \in \Sigma_N$ を持つ文法である。

例 **6.1** 次の文脈依存文法 $G_1 = (\{S, A, B\}, \{a, b\}, P_1, S)$ が導出する言語 [15, 例題 5.10]。

$$\begin{aligned} P_1 = \{&S \to aSBA \,\big|\, abA, \\ &AB \to BA, \\ &bB \to bb, \\ &bA \to ba, \\ &aA \to aa\}. \end{aligned}$$

G_1 は次のような導出が可能である。

$$S \Rightarrow aSBA \Rightarrow aabABA \Rightarrow aabBAA \Rightarrow aabbAA \Rightarrow aabbaA \Rightarrow aabbaa.$$

演習 **6.1** 例 6.1 の文脈依存文法 G_1 が生成する言語 $L(G_1)$ を帰納法によって示しなさい。

例 **6.2** 次の句構造文法 $G_2 = (\{S, A, B, C\}, \{a\}, P_2, S)$ が導出する言語 [10, 例 5.1]。

$$\begin{aligned} P_2 = \{&S \to BAB, \quad BA \to BC, \\ &CA \to AAC, \quad CB \to AAB \\ &A \to a, \quad B \to \varepsilon\}. \end{aligned}$$

G_2 は次のような導出が可能である。

$$\begin{aligned} &S \Rightarrow BAB \Rightarrow BCB \Rightarrow BAAB \Rightarrow BCAB \Rightarrow BAACB \Rightarrow BAAAAB \Rightarrow BCAAAB \\ &\Rightarrow BAACAAB \Rightarrow BAAAACAB \Rightarrow BA^6CB \Rightarrow BA^8B \Rightarrow A^8 \Rightarrow a^8 \end{aligned}$$

演習 **6.2** 例 6.2 の文法 G_2 が生成する言語 $L(G_2)$ を帰納法によって示しなさい。

例 **6.3** 言語 $L(G_2)$ は句構造文法よりも狭いクラス属する次の文脈依存文法 $G_3 = (\{S, A, B_1, B_2, C, D, E\}, \{a\}, P_3, S)$ でも生成可能である [10, 5 章演習 5.1]。ただし、$L(G_3) = L(G_2) - \{aa\}$.

$$\begin{aligned} P_3 = \{&S \to aa \,|\, B_1AAB_2, \quad B_2B_2 \to aa, \\ &B_1A \to aB_2 \,|\, CDEA, \quad B_2A \to aB_2, \\ &EA \to DDE, \quad EB_2 \to AB_2, \\ &DA \to AA, \quad CA \to B_1A\}. \end{aligned}$$

$$\begin{aligned} &S \Rightarrow B_1AAB_2 \Rightarrow CDEAAB_2 \Rightarrow CDDDEAB_2 \Rightarrow CDDDDDEB_2 \\ &CDDDDDAB_2 \Rightarrow CDDDDAAB_2 \Rightarrow CDDDAAAB_2 \Rightarrow CDDAAAAB_2 \\ &\Rightarrow CDAAAAAB_2 \Rightarrow CAAAAAAB_2 \Rightarrow B_1AAAAAAB_2 \Rightarrow aB_2AAAAAB_2 \Rightarrow aaB_2AAAAB_2 \\ &\Rightarrow \cdots \Rightarrow aaaaaAB2B2 \Rightarrow aaaaaaB_2B_2 \Rightarrow aaaaaaaa. \end{aligned}$$

演習 **6.3** 文法 G_3 が生成する言語 $L(G_3)$ を帰納法によって示しなさい。

例 **6.4** 次の文脈依存文法 $G_4 = (\{S, A, B, C, T_a, T_b, T_c\}, \{a, b, c\}, P_4, S)$ が導出する言語 [9, 例 15.7]。

$$
\begin{aligned}
P_4 = \{&S \to ABCS \,|\, ABT_c,\\
&CA \to AC, \quad BA \to AB,\\
&CB \to BC, \quad CT_c \to T_c c,\\
&BT_c \to T_b c, \quad BT_b \to T_b b,\\
&AT_b \to T_a b, \quad AT_a \to T_a a,\\
&T_a \to a\}.
\end{aligned}
$$

$$
\begin{aligned}
&S \Rightarrow ABCS \Rightarrow ABCABT_c \Rightarrow ABACBT_c \Rightarrow ABABCT_c \to AABBCT_c \to AABBT_cc\\
&\Rightarrow AABT_cc \Rightarrow AAT_bbcc, \Rightarrow AT_abbcc \Rightarrow T_aabbcc \Rightarrow aabbcc
\end{aligned}
$$

演習 **6.4** 例 6.4 の文脈依存文法 G_4 が生成する言語 $L(G_4)$ を帰納法によって示しなさい。

例 **6.5** 次の文脈依存文法 $G_5 = (\{S, B\}, \{a, b, c\}, P_5, S)$ が導出する言語。

$$
\begin{aligned}
P_5 = \{&S \to aSBc \,|\, abc,\\
&AB \to BA,\\
&cB \to Bc,\\
&bB \to bb\}.
\end{aligned}
$$

$$S \Rightarrow aSBc \Rightarrow aabcBc \Rightarrow aabBcc \Rightarrow aabbcc.$$

演習 **6.5** 文法 G_5 が生成する言語は例 6.4 の文法 G_4 が生成する言語と同じであることを示しなさい。

6.4 文脈自由言語

6.4.1 文脈自由文法

定義 **6.3** 文法 $G = (\Sigma_N, \Sigma_T, P, S)$ が文脈自由文法 (CFG: context free grammar) とは、生成規則 P が

$$A \to \alpha, \quad A \in \Sigma_N, \alpha \in (\Sigma_N \cup \Sigma_T)^*$$

の形の有限集合からなり、文形式 uAv（$u, v \in (\Sigma_N \cup \Sigma_T)^*$）において A の前後の文脈に無関係に置き換え規則が適用できて、

$$uAv \underset{G}{\Rightarrow} v\alpha v$$

と導出できるような文法である。置き換え規則 $A \to \alpha$ において、。$\alpha = \varepsilon$ のときは **ε-生成**という。非終端記号 A を左側に持つ複数の置き換え規則 $A \to \alpha_1, \ldots, A \to \alpha_n$ があるとき、記号 $|$ を or の意味で使って

$$A \to \alpha_1 \,\big|\, \ldots \,\big|\, \alpha_n$$

と表すことがある。

例 **6.6** 文脈自由文法 $G_1 = (\{S\}, \{0,1\}, P_1, S)$

$$P_1 = \{S \to 0S1,\ S \to \varepsilon\}.$$

G_1 の導出は次のようになる。

$$S \underset{G_1}{\Rightarrow} 0S1 \underset{G_1}{\Rightarrow} 00S11 \overset{*}{\underset{G_1}{\Rightarrow}} 0^nS1^n \underset{G_1}{\Rightarrow} 0^n1^n.$$

これより、言語

$$L = \{0^n1^n \,\big|\, n \geqq 0\}$$

は G_1 によって生成される文脈自由言語である。

例 **6.7** 文脈自由文法 $G_2 = (\{S, A, B\}, \{0,1\}, P_2, S)$

$$\begin{aligned} P_2 :\quad & S \to 0B \,\big|\, 1A, \\ & A \to 0 \,\big|\, 0S \,\big|\, 1AA, \\ & B \to 1 \,\big|\, 1S \,\big|\, 0BB. \end{aligned}$$

演習 **6.6** 文法 G_2 が生成する言語を帰納法によって示しなさい。

文脈自由文法での導出 $u \underset{G}{\Rightarrow} v$ とは、G の P 内の生成規則 $A \to \alpha$ の左辺に登場する非終端記号 A が文形式 u 内のどこかにあれば、その場所によらずに記号列 α に置き換えて文形式 v が得られる。

Σ_T 上の言語 $L \subseteq \Sigma_T^*$ に対して、L を生成する文脈自由文法 G が存在するとき、L を**文脈自由言語** (CFL: context free language) という。言語階層に関する結果から、文脈依存言語はその真の部分集合として文脈自由言語を含む。したがって、文脈自由言語は文脈依存文法によって生成されるが、その逆、文脈自由文法によって生成できない言語集合が存在する。

定義 6.4 文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ に対する導出木 (derivation tree)（解析木 (parse tree) または構文木 (syntax tree) ともいう）を次で定義する。

(1) 各 $a \in \Sigma_N \cup \Sigma_T$ に対して、記号 a をラベルに持つ 1 つの頂点だけからなる木（図 6.1）は導出木である。

図 6.1　1 頂点 $a \in \Sigma_N \cup \Sigma_T$ だけからなる導出木

(2) ε-生成規則 $A \to \varepsilon (A \in \Sigma_N)$ に対して木（図 6.2）は導出木である。

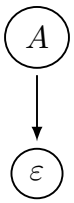


図 6.2　ε-生成規則 $A \to \varepsilon$ の導出木

(3) 木の根のラベルが $A_1, \ldots, A_n \in \Sigma_N \cup \Sigma_T$ である導出木を $T_1, \ldots, T_n$ とする。G の生成規則 $P : A \to A_1 \ldots A_n$ に対して、A を根とする図 6.3 のような木は導出木である。

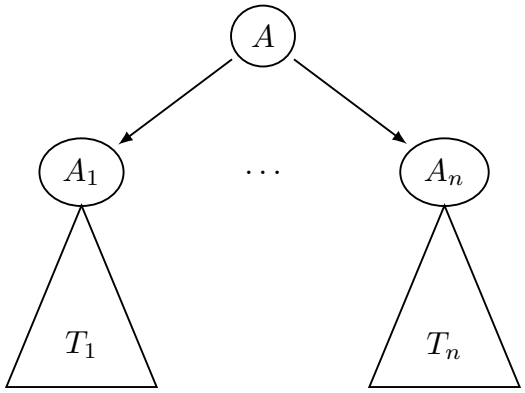


図 6.3　生成規則 $A \to A_1 \ldots A_n$ からの導出木

(4) 手順 (1),(2),(3) を使って定義される有限木だけが導出木である。

演習 6.7 例 6.7 で与えられた文脈自由文法 G_2 の導出木を書いてみなさい。

文法 G の導出 $A \overset{*}{\underset{G}{\Rightarrow}} \alpha, (A \in \Sigma_N, \alpha \in (\Sigma_N \cup \Sigma_T)^*)$ において、各導出で適用された生成規則（被置き換え記号は 1 度に 1 つだけ）を定めることによって導出木 T は一意に定まり、T の葉を左から右へ走査した記号列が α となっている。導出 $A \overset{*}{\underset{G}{\Rightarrow}} \alpha$ が、その各段階において記号列の最左にある非終端記号が置き換えられて得られているとき**最左導出** (leftmost derivation)、その各段階において記号列の最右にある非終端記号が置き換えられて得られているとき**最右導出** (rightmost derivation) という。

次の補題から、文脈自由文法の導出は最左導出だけを考えるだけで十分である。

補題 6.1 導出 $x \overset{*}{\underset{G}{\Rightarrow}} y$ を文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ の導出とする。このとき、x から y への最左導出がある。

証明

■

定義 6.5 文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ がある語 $w \in L(G)$ に対して 2 つ以上の異なる導出木を持つとき、G は**曖昧** (ambiguous) という。

例 6.8 加法と乗法からなる式を生成する文脈自由文法 $G_3 = \{\{S, T\}, \{a, b, +, *\}, P_3, S\}$

$$
\begin{aligned}
P_3 : \quad & S \to S + S \,|\, S * S \,|\, T, \\
& T \to x \,|\, y.
\end{aligned}
$$

G_3 は**中置き式** (infix) の加乗法の式を生成する。$x * y + x$ の導出は、$(x * y) + x$ または $x * (y + x)$ と 2 つの導出木を持ち、曖昧である。

演習 6.8 例 6.8 の文脈自由文法 G_3 で生成される文の導出木が 2 つ以上ある別の文例を挙げなさい。

演習 6.9 次の文脈自由文法 $G_a = (\{S, A, B\}, \{a, b\}, P, S)$ga 曖昧（ある語の導出過程を表す導出木が複数ある）であることを示しなさい。

$$
P_a : \quad A \to AC \,|\, CB, \quad A \to aAb \,|\, ab, \quad B \to bB \,|\, ba, \quad C \to ac \,|\, a.
$$

[ヒント] 導出 $aabba$ の導出過程を複数上げることができる。

文脈自由言語 L は、それを生成するどんな文脈自由文法 G も曖昧であるとき本質的に曖昧 (inherently ambiguous) という。本質的に曖昧な文脈自由言語が存在する [1]。

6.4.2 文脈自由文法の例

生成する言語が非自明な文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ を考えてみよう。

演習 6.10 文脈自由文法 $G_4 = (\{S, A, B, C\}, \{a\}, P_4, S)$

$$\begin{aligned} P_4: \quad & S \to BAB \,|\, BA \\ & A \to 0 \,|\, 0S \,|\, 1AA, \\ & B \to 1 \,|\, 1S \,|\, 0BB \end{aligned}$$

が定める言語 $L(G_4)$ を決定しなさい。

演習 6.11 $\Sigma_T = \{a, b\}$ 上の任意の回文 (palindrome)

$$\left\{ w \,|\, w = w^R, w^R \text{ は } w \text{ の反転} \right\}$$

を生成する文脈自由文法 G_7 を構成しなさい。

演習 6.12 $\Sigma_T = \{a, b\}$ として、言語

$$\left\{ ww^R \,|\, w \in \Sigma_T^*, w^R \text{ は } w \text{ の反転} \right\}$$

を生成する文脈自由文法 G_6 を構成しなさい (節 7.3 の例 7.5 節参照)。

演習 6.13 $\Sigma_T = \{a, b, c\}$ として、言語

$$\left\{ wcw^R \in \Sigma_T^* \,|\, w \in \{a, b\}^*, w^R \text{ は } w \text{ の反転} \right\}$$

を生成する文脈自由文法 G_5 を構成しなさい。

演習 6.14 次の文脈自由文法 $G_8 = (\{S, T\}, \{a, b\}, P_8, S)$ が定める言語 $L(G_8)$ を決定しなさい。

$$\begin{aligned} P_8: \quad & S \to aSb \,|\, T, \\ & T \to bTa \,|\, \varepsilon \end{aligned}$$

[ヒント]

$$L(G_8) = \left\{ a^n b^m a^m b^n \,\middle|\, n \geqq 0, m \geqq 0 \right\}.$$

演習 6.15 次の文脈自由文法 $G_N = (\{S, A, B\}, \{a, b\}, P_N, S)$ が定める言語 $L(G_N)$ を決定しなさい。

$$
\begin{aligned}
P_N : \quad & S \to aB \,|\, bA \,|\, \varepsilon \\
& A \to aS \,|\, bAA \\
& B \to bS \,|\, aBB.
\end{aligned}
$$

[ヒント] G_N は a と b の出現回数が等しい言語を生成する（演習 7.9 参照）。

$$L(G_N) = \left\{ w \in \{a, b\}^* \,|\, N_a(w) = N_b(w) \right\}$$

非終端記号を S だけ 1 使う文法規則

$$P' : \quad S \to aSbS \,|\, bSaS \,|\, \varepsilon$$

でも $L(G_N)$ が生成できることに注意。

演習 6.16 次の文脈自由文法 $G_{abc} = (\{S, S_1, S_2, A, C\}, \{a, b, c\}, P_{abc}, S)$ が定める言語 $L(G_{abc})$ を決定しなさい。

$$
\begin{aligned}
P_{abc} : \quad & S \to S_1C \,|\, AS_2, \\
& S_1 \to aS_1b \,|\, \varepsilon, \\
& S_2 \to bS_2c \,|\, \varepsilon, \\
& A \to aA \,|\, \varepsilon, \\
& C \to cC.
\end{aligned}
$$

[ヒント] G_{abc} は次の言語を生成する。

$$L(G_{abc}) = \left\{ a^i b^j c^k \,|\, i = j \text{ または } j = k, \quad i, j, k \geqq 0 \right\}$$

演習 6.17 次の文脈自由文法 $G_{ab} = (\{S, B\}, \{a, b\}, P_{ab}, S)$ が定める言語 $L(G_{ab})$ を決定しなさい（演習 7.10 参照）。

$$
\begin{aligned}
P_{ab} : \quad & S \to aSbB \,|\, \varepsilon, \\
& B \to b \,|\, \varepsilon.
\end{aligned}
$$

[ヒント]

$$L(G_{ab}) = \left\{ a^i b^j \,|\, 0 \leqq i \leqq j \leqq 2i \right\}.$$

演習 **6.18** 文脈自由文法 $G_9 = (\{S, T, U, X\}, \{a, b, c\}, P_9, S)$

$$
\begin{aligned}
P_9 : \quad & S \to XSX \,|\, T, \\
& T \to cUc, \\
& U \to XUX \,|\, a \,|\, b \,|\, \varepsilon, \\
& X \to a \,|\, b.
\end{aligned}
$$

が定める言語 $L(G_9)$ を決定しなさい。
[ヒント]

$$L(G_9) = \left\{ xcycz \,\middle|\, |x| = |z|, x, y, z \in \{a, b\}^2 \right\}.$$

演習 **6.19** 次の CFG を生成する文法を構成しなさい。

(1) $L_1 = \{a^n b^n a^m \,|\, m, n \geqq 1\}$
(2) $L_2 = \{a^n b^m a^m \,|\, m, n \geqq 1\}$

演習 **6.20** 次の文脈自由言語を生成する文法を定めさない。

(1) $L_1 = \{a^n b^m \,|\, 1 \leqq n, m = n + 1\}$
(2) $L_2 = \{a^n b^m \,|\, 1 \leqq m, n = m + 1\}$
(3) $L_3 = \{a^n b^m \,|\, 1 \leqq n \leqq m\}$
(4) $L_4 = \{a^n b^m \,|\, 1 \leqq m \leqq n\}$
(5) $L_5 = \{a^n b^m \,|\, m, n \geqq 1, m \neq n\}$

演習 **6.21** $\Sigma_N = \{S, A, B\}, \Sigma_T = \{a, b, c\}$ とし、CFG $G_1 = (\Sigma_N, \Sigma_T, P_1, S)$ と $G_2 = (\Sigma_N, \Sigma_T, P_2, S)$ を次のように定める。

$$P_1 : \quad S \to aAbB, \quad A \to aAb \,|\, \varepsilon, \quad B \to cB \,|\, c,$$

$$P_2 : \quad S \to AbBc, \quad A \to aA \,|\, a, \quad B \to bBc \,|\, \varepsilon.$$

(1) $L(G_1), L(G_2)$ はどのようなものか。
(2) $L(G1) \cup L(G_2)$ を生成する CFG を構成しなさい。
(3) $L(G1) \cap L(G_2)$ はどのような言語か。

ヒント) 演習 6.1、演習 6.19

6.5　文脈自由文法の簡約化

文脈自由言語 L が与えられているとき、その言語を生成する CFG G（$L = L(G)$）は唯一つに定まらない。CFG に冗長な記号や生成規則を保つ場合、それらを見つけて除去して簡約化することを考える。

正規言語は Myhill-Nerode の定理（4.6）にあったように（したがって、その文法型も）、状態数を最小化して簡約する方法は（同型を除いて）ただ一つに定まるが、文脈自由文法では簡約化は唯一に定まるとは限らない。

例 **6.9** 無意味な ε-規則や単位記号の置き換えは簡約できる。簡約化は一通りに決まるわけではない。

(1) $P_1 : S \to aA, \quad A \to bA \,|\, \varepsilon.$
書き換え $A \to \varepsilon$ は、右辺に A が登場する全ての規則を書き直して ε-規則を、削除できる。
$\Rightarrow S \to aA \,|\, a, \quad A \to bA \,|\, ba.$

(2) $P_2 : S \to aA, \quad A \to aB \,|\, B, \quad B \to bB \,|\, b.$
単位規則 $A \to B$ があれば、右辺に B が登場する全ての規則を書き直して、削除できる。
$\Rightarrow S \to aA, \quad A \to aB \,|\, bB \,|\, b, \quad B \to bB \,|\, b.$
$\Rightarrow S \to aA \,|\, aB, \quad A \to aB, \quad B \to bB \,|\, b.$
$\Rightarrow S \to aA \,|\, abB \,|\, ab, \quad A \to aB, \quad B \to bB \,|\, b.$

演習 **6.22** 例 6.9 の CFG の生成規則 P_1, P_2 の簡約によって、生成される CFL は変わらないことを示しなさい。

CFG の簡約化には、無用記号、ε-生成規則、単位生成規則を見つけて生成規則を書き換える。

6.5.1　無用記号の除去

文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ において、開始記号 S からの導出において非終端記号 $X \in \Sigma_N$ が有用 (useful) とは次の 2 条件を満たすことである。

生記号 $X \overset{*}{\Rightarrow} w, \quad w \in \Sigma_T.$ (6.1)

到達可能記号 $S \overset{*}{\Rightarrow} \alpha X \beta, \quad \alpha, \beta \in (\Sigma_T \cup \Sigma_T)^*$ (6.2)

記号 $X \in \Sigma_N$ について有用な導出がないとき、非終端記号 X を無用 (useless) という。$L(X) = \phi$ である非終端記号を死記号 (dead symbol) という。

無用記号は、次の 2 段階を順に経ることによって見つけることができる。

(1) 生記号 (live symbol) を逐次的に見つけ尽くして死記号の集合を求める。開始記号以外の死記号は無用である。死記号を含む生成規則は除去できる。
(2) 到達可能記号（$\in \Sigma_N \cup \Sigma_T$）を逐次的に見つけ尽くして残された記号は開始記号から到達不能記号である。到達不能記号を含む生成規則は除去できる。

例 **6.10** 次の生成規則

$$P: \quad S \to aAB, \quad A \to aBB, \quad B \to ab, \quad B \to cCD, \quad D \to d.$$

(1) P の生成規則で、右辺が終端記号列 $\in \Sigma_T$ である規則の左辺の記号 B, D は自明な生記号集合 $N_1 = \{B, D\}$ となる。これより、A を左辺に持つ規則も終端記号列を導出することがわかり生記号集合 $N_2 = N_1 \cup \{A\}$ が求まる。さらに、これより、開始記号 S も生記号であることがわかり生記号集合 $N_3 = \{S, A, B, D\}$ が求められる。P において、右辺が $(\Sigma_T \cup N_3)^*$ であるような規則はなく、N_3 以上の生記号集合はないことがわかる、$\Sigma_N - N_3 = \{C\}$ が死記号である。P から死記号 C を含む生成規則 $B \to cCD$ を取り除いて

$$P': \quad S \to aAB, \quad A \to aBB, \quad B \to ab, \quad D \to d$$

を得る。

(2) 開始記号は自明な到達可能記号の集合 $R_1 = \{S\}$. S を左辺とする規則から、到達可能記号の集合は拡大され $R_2 = R_1 \cup \{a, A, B\}$ となる。さらに A, B を左辺に持つ生成規則から到達可能記号集合 $R_3 = R2 \cup \{b\} = \{a, b, S, A, B\}$ を得る。R_3 の記号を左辺に持つ規則はないため、R_3 以上の到達可能集合はない。これより、非終端記号 D および終端記号 d は到達不能。P' から $\{d, D\}$ を含む規則を除去して

$$P'': \quad S \to aAB, \quad A \to aBB, \quad B \to ab$$

を得る。

演習 **6.23** 例の生成規則 P から生成される言語を定めなさい。

6.5.2 ε-生成規則の除去

CFG $G = (\Sigma_N, \Sigma_T, P, S)$ の生成規則 P において、$A \to \varepsilon$ は ε-生成規則という。$\varepsilon \notin L(G)$ であるとき、ε-生成規則は本質的でなく、除去することができる。

CFG G が次の条件のどちらかを満たす時、G は ε-無し (free) という。

- P には ε-生成規則がない（$\varepsilon \notin L(G)$）。
- P 内の唯一の ε-生成規則は $S \to \varepsilon$ で（$\varepsilon \in L(G)$）、開始記号 S は P の右辺には現れない。

CFG G は次の段階を経て、ε-無し文法 G' に簡約化できる。

(1) 消去可能記号 (nullable)（$A \overset{*}{\Rightarrow} \varepsilon$ が可能な非終端記号）の集合 N_ε をすべて見つける。

(2) ε-生成の除去して簡約化した生成規則 P' を作る。

(a) 生成規則 P から、右辺に消去可能記号 ($\in N_\varepsilon$) を含まない規則を P' の規則とする。

(b) P 内の右辺に消去可能記号な記号を含む規則

$$A \to \alpha_0 E_1 \alpha_1 \cdots E_n \alpha_n, \quad E_i \in N_\varepsilon, \alpha_i \in (\Sigma_T \cup (\Sigma_N - N_\varepsilon))^*$$

に対し、次のような $X_i = E_i$ または $X_i = \varepsilon (i = 1, \ldots, n)$ とした可能な組み合わせを P' の要素とする。

$$A \to \alpha_0 X_1 \alpha_1 \cdots X_n \alpha_n, \quad \text{ただし } \alpha_0 X_1 \alpha_1 \cdots X_n \alpha_n \not\equiv \phi.$$

(c) $S \in N_\varepsilon$（$\varepsilon \in L(G)$）のとき、P' の要素として

$$S' \to \varepsilon, S' \to S, \quad S' \notin N_\varepsilon, N' = N \cup \{S'\}.$$

一方、$S \notin N_\varepsilon$（$\varepsilon \notin L(G)$）のとき、$N' = N, S' = S$.

以上から、$S' \to \varepsilon$ 以外の ε-生成規則は除去される。

例 **6.11** 次の CFG $G = (\{S, A, B, C\}, \{a, b, c\}, P, S)$

$$P: \quad S \to aAC \,|\, BC, \quad A \to AA \,|\, B, \quad B \to b \,|\, C, \quad C \to \varepsilon.$$

(1) 消去可能記号集合

右辺に ε を持つ自明な消去可能記号集合 $N_1 = \{C\}$. 右辺に N_1 の要素からなる規則から、$N_2 = N_1 \cup \{B\}$. これより、右辺に N_2 の要素からなる規則から、$N_3 = N_2 \cup \{S, A\}$. N_3 はこれ以上拡大できず、$N_\varepsilon = N_3 = \{S, A, B, C\}$.

(2) ε-生成を除去した P' の規則。

(a) $B \to b$.

(b) $S \to a \,|\, aA \,|\, aC \,|\, aAC \,|\, B \,|\, C \,|\, BC, \quad A \to A \,|\, AA \,|\, B, \quad B \to C$.

(c) $S' \to \varepsilon \,|\, S$.

これより、C は死記号（無用）となり、$S \to aC \,|\, aAC \,|\, C \,|\, BC, \quad B \to C$ は除去される。

以上から、無用記号 C を含む規則は除去して

$$P' : \quad S' \to \varepsilon \,|\, S, \quad S \to a \,|\, aA \,|\, B, \quad A \to A \,|\, AA \,|\, B, \quad B \to b.$$

演習 **6.24** 上の例の CFG G から定まる言語 $L(G)$ を決定しなさい。

6.5.3 単位規則の除去

CFG G の生成規則 P において、$A \to B$（$A, B \in \Sigma_N$）の形の規則を単位規則 (unit production) といい、次の段階を経て除去可能で、新しい生成規則 P' を作る。

(1) 各 $A \in \Sigma_N$ について、A から単位規則だけで導出される A を含む集合 N_A を次のようにして逐次的に求める。$N_A^{(0)} = \{A\}$ から、

$$N_A^{(i)} = N_A^{(i-1)} \cup \left\{ B \,\middle|\, X \to B, \quad X \in N_A^{(i-1)} \right\}$$

が $N_A^{(k)} = N_A^{(k-1)}$ となるまで繰り返して $N_a = N_A^{(k)}$ とする。CFG G は既に ε-無しとなっているため

$$N_A = \left\{ B \,\middle|\, A \overset{*}{\Rightarrow} B \right\}$$

である。

(2) 各 $A \in \Sigma_N$ について、$B \in N_A$ かつ $B \to \alpha \in P$ となる $alpha$

$$A \overset{*}{\Rightarrow} B \Rightarrow \alpha, \quad |\alpha| \neq 1$$

を求め、$A \to \alpha$ の形すべてだけを P' の要素とする。

演習 **6.25** 次の CFG の生成規則 P を簡約化して ε-生成規則を除去しなさい。

(1) $S \to aA, \quad A \to aA \,|\, bB, \quad B \to bB \,|\, \varepsilon.$
(2) $S \to aAB, \quad A \to a \,|\, aA \,|\, Bb, \quad B \to bB \,|\, \varepsilon.$
(3) $S \to aA, \quad A \to b \,|\, aA \,|\, B, \quad B \to cB \,|\, \varepsilon.$
(4) $S \to aAB, \quad A \to aA \,|\, AB \,|\, \varepsilon, \quad B \to bB \,|\, \varepsilon.$

演習 **6.26** 次の CFG の生成規則 P と等価な単位規則を持たない生成規則を作りなさい。

(1) $S \to aA, \quad A \to aA \,|\, B, \quad B \to bB \,|\, \mathrm{b}\,.$
(2) $S \to aA, \quad A \to aA \,|\, B, \quad B \to bB \,|\, bA \,|\, b.$
(3) $S \to A \,|\, B, \quad A \to aA \,|\, a, \quad B \to bB \,|\, b.$

演習 **6.27** 次の CFG の生成規則 P を簡約化して ε-生成規則を除去しなさい。

(1) $S \to AB \,|\, a, \quad A \to a.$
(2) $S \to A \,|\, C, \quad A \to aC \,|\, bSa \,|\, b, \quad B \to SB \,|\, BAa, \quad C \to AS \,|\, bC.$
(3) $S \to AC \,|\, CB, \quad A \to aAb \,|\, bDa \,|\, ab, \quad B \to bBa \,|\, ba, \quad C \to aC \,|\, C \,|\, a, \quad D \to SE$
$E \to D, \quad F \to aFb \,|\, ab.$
(4) $S \to A, \quad A \to B \,|\, CD \quad B \to DE, \quad C \to aCb \,|\, ab, \quad D \to aC \,|\, a, \quad E \to bEa \,|\, ba.$
(5) $S \to AB, \quad A \to ABAC \,|\, B \,|\, a, \quad B \to C \,|\, b \,|\, \varepsilon \quad C \to B \,|\, C \,|\, \varepsilon$
(6) $S \to A, \quad A \to AB \,|\, a, \quad B \to C \,|\, b, \quad C \to A \,|\, c$
(7) $S \to aAbB, \quad A \to aAb \,|\, \varepsilon, \quad B \to cB \,|\, c$
(8) $S \to AbBc, \quad A \to aA \,|\, a, \quad B \to bBc \,|\, \varepsilon$

6.6 文脈自由文法の標準形

6.6.1 Chomsky 標準形

定義 **6.6** 文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ において、すべての生成規則が次の形式であるとき、G は **Chomsky** 標準形 (Chomsky normal form) であるという。

(1) $A \to BC, \quad A, B, C \in \Sigma_N$
(2) $A \to a, \quad a \in \Sigma_T$

(3) $S \to \varepsilon$

ただし、(3) $S \to \varepsilon \in P$ のとき、(1) において $B, C \in \Sigma_N - S$.

定理 6.1 ε を含まない任意の文脈自由言語 L は、Chomsky 標準形を持つ文法 G によって生成される。L が ε を含むとき、ε-生成規則は

$$S \to \varepsilon \qquad (??')$$

のみを含み、G には他の ε-生成規則はない。

証明　概略) 次の手順で Chomsky 標準形を構成できる。

(1) ε を含まない任意の CFG G に対し、いつでも ε-生成規則や単位規則を含まない等価な文法 G_1 を求めることができる。

(2) その生成規則で $A \to X_1 X_2 \cdots X_m$ の形を考える（$m \geqq 2$)。$X_i = a \in \Sigma_T$ のとき、新しい非終端記号 $\bar{X}_i$ を追加して Σ'_N を更新し、同時に文法規則において X_i を $\bar{X}_i$ に置き換えたうえで、Chomsky 標準形規則 $\bar{X}_i \to a$ を導入して P' を更新する。こうして得られる ε を含まない任意の文脈自由文法 $G_2 = (\Sigma'_N, \Sigma_T, P', S)$ は、その規則 P' が

$$A \to a, \quad a \in \Sigma_T,$$
$$A \to B_1 B_2 \ldots B_m, \quad B_i \in \Sigma'_N$$

という形になる。

(3) $m \geqq 3$ であるような P' の生成規則 $A \to B_1 B_2 \ldots B_m$ について、新しい非終端記号 $D_1, D_2, \ldots, D_{m-2}$ を加えて、その生成規則を次のように書き換えることができる。

$$\begin{aligned} &A \to B_1 B_2 \ldots B_m \\ &\quad A \to B_1 D_1 \\ &\quad D_1 \to B_2 D_2 \\ &\quad \vdots \\ &\quad D_{m-3} \to B_{m-2} D_{m-2} \\ &\quad D_{m-2} \to B_{m-1} D_{m-1} \end{aligned}$$

(4) こうして定めた非終端記号集合 Σ''_N と生成規則 P'' から決まる $G_3 = (\Sigma''_N, \Sigma_T, P'', S)$ が求める文法で、$L = L(G_3)$.

証明の概要から、生成規則に Chomsky 標準形の形式になっているものは流用する。その他の形については、たとえば、$X, A, B, C \in \Sigma_N$, $a \in \Sigma_T$ のとき

$$A \to aB \quad \Rightarrow \quad X \to AB, A \to a$$

$$A \to ABC \quad \Rightarrow \quad X \to AY, Y \to BC.$$

例 **6.12** $\Sigma_N = \{S, A, B, C\}, \Sigma_T = \{a, b\}$,

$$P: \quad S \to AaC \,|\, CbBa, \quad A \to aAb \,|\, ab, \quad B \to bB \,|\, b, \quad C \to Ca \,|\, a.$$

を Chomsky 標準形に書き換える。

$$S \to AaC \Rightarrow S \to AS_1, S_1 \to S_2C, S_2 \to a$$
$$S \to CbBa \Rightarrow S \to CS_3, S_3 \to S_4S_5, S_5 \to BS_2$$
$$A \to aAb \Rightarrow A \to S_2A_1, A_1 \to AS_4$$
$$A \to ab \Rightarrow A \to S_2S_4$$
$$B \to bB \Rightarrow B \to S_4B$$
$$C \to Ca \Rightarrow C \to CS_2.$$

演習 **6.28** 次の生成規則で定まる CFG を Chomsky 標準形に書き換えなさい。ただし、$0, 1, 2, a, b, +, *, (,), x$ は終端記号。

(1) $S \to AB, \quad A \to A1 \,|\, A2, \quad B \to B0 \,|\, 0.$
(2) $S \to Ab, \quad A \to aAb \,|\, ab.$
(3) $S \to Sb \,|\, aSb \,|\, ab.$
(4) $S \to aA \,|\, Bb, \quad A \to aA \,|\, aAb \,|\, ab, \quad B \to Bb \,|\, aBb \,|\, ab.$
(5) $S \to S + S \,|\, S * S \,|\, (S) \,|\, x.$

6.6.2 Greibach 標準形

文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ において、

$$A \Rightarrow \alpha \overset{*}{\underset{G}{\Rightarrow}} \beta A \gamma, \quad \alpha, \beta, \gamma \in (\Sigma_N \cup \Sigma_T)^*$$

であるとき、A は**再帰的** (recursive) であるという（節 6.7 の自己埋込的の定義 6.8 と比較）。

$\gamma = \varepsilon$ のときは、A は**右再帰的** (right recursive) という。一方、$\beta == \varepsilon$ のとき

$$A \underset{G}{\Rightarrow} \alpha \overset{*}{\underset{G}{\Rightarrow}} A\gamma, \quad \alpha, \gamma \in (\Sigma_N \cup \Sigma_T)^*$$

であるような 1 段階以上の導出があるとき、A は**左再帰的** (left recursive) という。とくに、生成規則 P が $A \to A\alpha'$（$\alpha' \in (\Sigma_N \cup \Sigma_T)^*$）を含むとき、$A$ は直接左再帰的であるという。

定義 6.7 文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ において、すべての生成規則が次の形式であるとき、G は **Greibach 標準形** (Greibach normal form) であるという。

(1) $A \to aB_1B_2 \cdots B_m, \quad a \in \Sigma_T, B_1, \ldots, B_m \in \Sigma_N$
(2) $A \to a, \quad a \in \Sigma_T$
(3) $S \to \varepsilon$

ただし、(3) $S \to \varepsilon \in P$ のとき、(1) において $B_i \in \Sigma_N - S$.

Gleibach 標準形は左再帰的ではない。左再帰的生成規則であっても、それが生成する言語を変えることなく左再帰的「でない」生成規則である Gleibach 標準形に変換することができる。

このために左再帰的規則を除去する必要がある。文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ のある生成規則

$$A \to \alpha_1 B \alpha_2, \quad B \in \Sigma_N, \alpha_1, \alpha_2 \in (\Sigma_N \cup \Sigma_T)^*$$

が除去したい規則である時、B を左辺に持つ置き換え規則すべて

$$B \to \beta_1 \cdots \beta_r$$

を使って、次の r 個の規則で置き換えればよい。

$$A \to \alpha_1\beta_1\alpha_2 \,|\, \cdots \,|\, \alpha_1\beta_r\alpha_2.$$

A に着目したとき、A の直接左再帰性を除去するには次のようにする。置き換え後も A で始まる r 個の規則と置き換え後に A で始まらない s 個の規則

$$A \to A\alpha_i, \quad \alpha_i \in (\Sigma_N \cup \Sigma_T)^*, i = 1, \ldots, r$$
$$A \to \beta_j, \quad j = 1, \ldots, s$$

があるとする。直接左再帰性を除去するには、新しく記号 B を導入して、先の r 個の直接左再帰的な規則を、次の $s + 2r$ 個の新しい規則で置き換えれば良い。

$$A \to \beta_j B, \quad j = 1, \ldots, s$$
$$B \to \alpha_i \,|\, \alpha_i B, \quad i = 1, \ldots, r.$$

以上、これからのような置き換え規則の除去によっても、文法は等価で生成される言語は変わらない。

6.7 自己埋込と反復補題

定義 6.8 文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ が**自己埋込み的** (self-embedding) とは、ある非終端記号 $A \in \Sigma_N$ が存在して

$$A \overset{*}{\underset{G}{\Rightarrow}} \alpha A \beta, \qquad \alpha, \beta \in (\Sigma_N \cup \Sigma_T)^\dagger, \quad \big(|\alpha|, |\beta| \geqq 1\big)$$

となる導出が可能であることである。この A は自己埋込み的という。

言語が自己埋込というのは、それを生成する「すべての」文法が自己埋め込みであるときである（L を生成するある G が自己埋め込みだというだけでは不十分)。

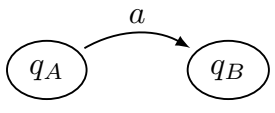
文脈自由文法で A について自己埋込みであるとき、A の前後の文脈には依存せずに導出可能であることから、任意の $k \geqq 1$ について

$$A \overset{*}{\underset{G}{\Rightarrow}} \alpha A \beta \overset{*}{\underset{G}{\Rightarrow}} \alpha\alpha A \beta\beta A \overset{*}{\underset{G}{\Rightarrow}} \ldots \overset{*}{\underset{G}{\Rightarrow}} \alpha^k A \beta^k, \qquad k \geqq 1$$

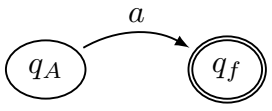
を導出できる。

ところで、正規言語を生成する決定有限オートマトン ($Q = \{q_A, q_B, \ldots\}, \Sigma_T, \delta, q_0, F = \{q_f\}$) に 1-1 対応する正規文法 G_R の (右線形) 生成規則は次の形を持つ。

$$A \to aB \quad \Leftrightarrow \quad \delta(q_A, a) = q_B \qquad \equiv$$



$$A \to a, a \in \Sigma_T^* \quad \Leftrightarrow \quad \delta(q_A, a) = q_f \qquad \equiv$$



記号 A からの導出は $A \overset{*}{\underset{G_R}{\Rightarrow}} a_1 \ldots a_k C$ となり非終端記号は文形式の最後にだけ現れるために、正規文法自己埋込にはなり得ない。

ただし、ある 1 つの文法が自己埋込み的であるといっても、それが生成する言語が非正規言語であるとは限らない。実際、自己埋込み的文法のいくつかは正規言語を生成する。

例 **6.13** 文法 $G_{R0} = (\{S\}, \{a\}, \{S \to aSa, S \to aa, S \to a\}, S)$ は自己埋込み的である。G_{R0} が生成する言語

$$L(G_{R0}) = \{a^k \mid k \geqq 1\}$$

は正整数個の a 持つ文字列となる。一方、文法 $G'_{R0} = (\{S\}, \{a\}, \{S \to aS, S \to a\}, S)$ は $L(G_{R0})$ と同じ言語を生成するが、文法 G'_{R0} は自己埋込み的ではない。

演習 **6.29** 文法 G_{R0} または文法 G'_{R0} が生成する言語を受理する有限オートマトンを構成してみなさい。これより、これら文法が生成する言語は正規であることを示しなさい。

例 **6.14** 文法 $G_{R1} = (\{S\}, \{a\}, \{S \to aSa, S \to a\}, S)$ は自己埋込み的である。G_{R1} が生成する言語

$$L(G_{R1}) = \{a^{2k+1} \mid k \geqq 0\}$$

は奇数個の a 持つ文字列となる。これを受理する 2 状態の有限オートマトンを容易に構成できるために $L(G_{R1})$ は正規言語である。$L(G_{R1})$ と同じ言語を生成する自己埋込み的でない文法を定めなさい。

演習 **6.30** 文法 G_{R1} またはそれと同等な文法が生成する言語を受理する有限オートマトンを構成してみなさい。

定義 **6.9** ある言語 L を生成するすべての文法が自己埋込み的であるとき、L は**自己埋込み的言語**という。

したがって言語の自己埋込み性をいうためには言語を生成する文法の 1 つが自己埋込みであるというだけでは不十分である。

実際、次を証明することができる。

定理 **6.2** すべての非正規な文脈自由言語は自己埋込み的で、すべての自己埋込み的言語は非正規である [6, Theorem 2.8]。言語が正規であるとは、それが文脈自由な非自己埋込み的であるときに限る [5, Thorem 5.5]。

証明　ここでは、2 番目の主張を証明する。

どんな正規言語も自己埋込み的でないことから、言語 L が自己埋込み的であれば、それは正規ではないことになる。

逆に、L が非自己埋込みでない文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ によって生成されると仮定する。このとき、正規文法に書き換え可能であることを証明できる。一般性を失うことなく、各 $A \in \Sigma_N$ について S から導出される文形式が A を含むと仮定してよい（そうでない場合、A や A を含む規則は $L(G)$ を変えることなく除去することができる）。以下、2 つの場合にわけて考える。

[場合 1]. 各 $A \in \Sigma_N$ において、A からの導出が S を含む場合。

生成規則 P において、右辺に非終端記号を含むすべての規則は次の 4 つの型 (i) $A \to \alpha B \beta$, (ii) $A \to \alpha B$, (iii) $A \to B\beta$, (iv) $A \to B$ のいずれかである（$A, B \in \Sigma_N$, $\alpha, \beta \in (\Sigma_N \cup \Sigma_T)^\dagger$）。

P が (i) の型を含むとき、仮定よりその導出はある語 $\alpha_1, \alpha_2, \beta_1, \beta_2$ を使って

$$A \Rightarrow \alpha B \beta \overset{*}{\Rightarrow} \alpha\alpha_1 S \beta_1 \beta \overset{*}{\Rightarrow} \alpha\alpha_1\alpha_2 A \beta_2\beta_1\beta.$$

$\alpha, \beta \neq \varepsilon$ であるため G は自己埋込みとなってしまい、この型の規則は P にはない。

同様な矛盾は P が (ii),(iii) の型を含むときにも生じる。P が型 (ii) を 1 つ含むとき、P に含まれる型 (ii) のすべての規則において、α は Σ_T 上の語である。さもないと P は型 (i) や (iii) の規則も含んでしまう。これは、G が右線形文法であることを意味する。おなじく、P が型 (iii) の規則を含むとすると G は左線形になり、これら両者から $L(G)$ は正規言語である。以上のことより、P が型 (i)-(iii) の規則を含まなければ $L(G)$ は正規言語である。

[場合 2]. $A_1 \overset{*}{\Rightarrow} \alpha S \beta$ となる $\alpha, \beta \in (\Sigma_N \cup \Sigma_T)^\dagger$ がないような非終端記号 A_1 がある場合。

$L(G)$ が正規言語であることの証明は非終端記号の数 n に関する帰納法で行う。$n = 1$ のとき、$S \overset{*}{\Rightarrow} S$ しかならない。$n = k$ で成立すると仮定する。Σ_N の非終端記号の数を $k+1$ として、元の P から S を含むすべての規則を除いた生成規則 P' を持つ文法

$$G_1 = (\Sigma_N - \{S\}, \Sigma_T, P', A_1)$$

また、P から左辺に A_1 を持つすべての規則を取り除いた生成規則 P'' を持つ文法で A_1 を終端記号と考えた

$$G_2 = (\Sigma_N - \{A_1\}, \Sigma_T \cup \{A_1\}, P'', S)$$

を考える。$L(G_1)$ および $L(G_2)$ の両方は、帰納法の仮定あるいは [場合 1] のいずれかより正規言語である。$L(G)$ は $L(G_2)$ 内の A_1 を $L(G_1)$ で置き換えたものであることに注意すると、これより $L(G)$ は正規言語であることがわかる。したがって $n = k+1$ について主張が証明され、帰納法が完成する。

■

これより、自己埋込み性は文脈自由言語の特徴であることがわかる。これに関連して任意の文脈自由言語が満たすべき必要な条件として次がある。

補題 6.2 (CFL に対する反復補題) L が任意の文脈自由言語とする。このとき、ある定数 n が存在して、長さが n 以上の L 内の語 z は次の性質を満たすように $z = uvwxy$ と分解できる [1, 補題 6.1]。

(1) $|vx| \geqq 1$,
(2) $|vwx| \leqq n$,
(3) 任意の $i \geqq 0$ について $uv^iwx^iy \in L$.

証明　$L - \{\epsilon\}$ を生成する Chomsky の標準形文法を G とする。語 $z \in L(G)$ が十分に長ければ、z の構文木の葉への経路は長くなる。正確には、h に関する帰納法によって次がわかる：ある語の構文木（高々 2 分木）T_S の高さが h 以下（その構文木は長さ h 以下の経路しか含まない）であれば、その語の長さは高々 2^{h-1} である。実際、$h = 1$ のときには、明らか。$h > 1$ のとき成立していると仮定する。根 S からの左右の部分 2 分木 T_ℓ と T_r の高さは $h - 1$ を越えないことより、帰納法の仮定からそれらの左右の 2 分木 T_ℓ, T_r が生成する語の長さは 2^{h-2} を越えない。したがって、木全体 T_S は長さ 2^{h-1} 以下の語を生成する。

さて、G の非終端記号の個数が $k = |\Sigma_N|$ とするとき、$n = 2^k + 1$ と選ぶ。語 $z \in L(G)$ で $|z| \geqq n$ であるとき、$|z| > 2^k$ であるため、z を横出する構文木には長さ $k + 1$ 以上の経路が存在する。その経路 p 上には $k + 2$ 個以上の頂点を含み、S からの経路端点である葉以外のラベルは非終端記号であり、その数は $k + 1$ 以上ある。鳩ノ巣原理からある記号 $A \in \Sigma_N$ が二回以上する。

これより、経路 p の頂点においては次の条件を満たす 2 頂点 v_1, v_2 が存在する：

(1) 経路 p 上の 2 頂点は同じラベル A を持つ。$v_1 = v_2 = A$.
(2) v_1 は v_2 よりも根 S に近い。
(3) v_1 から葉までの経路の長さは $k + 1$ を越えない。

頂点 v_1 を根とする 2 分木 T_{v_1} は長さ 2^k 以下の部分語 z_1 を導出する（T_{v_1} 内には長さ $k + 1$ を超える経路がない）。

経路 p を v_1 から更に下った頂点 v_2 を根とする 2 分木 T_{v_2} が生成する語 w を考えると、

$z_1 = uwx$ である。ここで、u, x は共に ε とはなり得ない。Chomsky 標準形である G において z_1 の導出に最初に使われる規則は $A \to BC$ の形をしており、部分二分木 T_{v_2} は B を根とする部分木か、あるいは C を根とする部分木の真に含まれているからである（どちらの部分木も ϵ でない語を導出する）。

以上から、次の A の導出があることがわかる：

$$A \overset{*}{\underset{G}{\Rightarrow}} vAx, \quad A \overset{*}{\underset{G}{\Rightarrow}} w, \quad |vwx| \leqq 2^k = n$$

この A の自己埋込み性から、各 $i > 0$ に対して

$$A \overset{*}{\underset{G}{\Rightarrow}} v^i A x^i$$

であることもわかる。これより根 S から導出される語 z はある u,y について $z = uvwxy$ と表すことができる。 ■

6.7.1 反復補題の応用

正規言語版の反復補題（節 3.6）と同様に、文脈自由文法の反復補題は、ある言語が文脈自由 (CFL) ではないことの証明に利用できる。CFL の反復補題は、目的の言語が文脈自由でないことを示す際に次のような手順で使われる [1, 節 7.2.3]。

(1) CFL でないことを示したい言語 L を選ぶ。
(2) 自由に n を決める。
(3) L に属する長さ n 以上の語 z を選ぶ。
(4) $z = uvwxy$ のように分割する（ただし、$|vwx| \leqq n$、$vx \neq \varepsilon$ を満たすように）。
(5) ある i を選ぶと $uv^iwx^iy \notin L$ であることを示す。
(6) L は CFL でないことを結論できる。

ただし、反復補題はすべての非 CFL について有効であるわけではない [1, Ogden の補題（I, p.167)]。

まず、次の文脈自由文法の例を確認しておこう。

演習 6.31 Greibach 型の文脈自由文法 $G = (\{S, B, C, D, E\}, \{a, b, c\}, P, S)$ を考える [10, 例 4.18]：

$$
\begin{aligned}
P = \{ \quad & S \to aB, \quad B \to b, \quad S \to aC, \quad C \to c, \\
& S \to aDB, \quad D \to aB, \quad D \to aDB, \quad S \to aEC, \\
& E \to aC, \quad E \to aEC \quad \}
\end{aligned}
$$

このとき G が生成する言語は

$$L(G) = \{a^i b^i \mid i \geqq 1\} \cup \{a^j c^j \mid j \geqq 1\}$$

であることを示しなさい。

しかしながら、文脈自由文法の自己埋込み性だけでは 3 個以上の部分に関する制御はできなくなる。次はその例である。

例 **6.15** 言語

$$L = \{a^k b^k c^k, \mid k \geqq 1\}$$

は文脈自由言語ではい。L を生成する文脈自由文法 $G = (\Sigma_N, \Sigma_T, P, S)$ が存在すると矛盾することがわかり [1, 例 6.1] 、L は文脈自由言語でないことを証明できる。

L を生成する CFG G が存在すると仮定する。CFL の反復補題で L だけで決まる定数を n とし、語 $z = a^n b^b c^n \in L$ を考える。反復補題を満たすように、$z = uvwxy$ と分解しておく。v, x を取り除いた（補題で $v^i, x^i (i = 0)$）uwy も L に属している。ここで、部分語 v, x が $a^n b^b c^n$ のどの場所に位置しているかを考えてみる。

$|vwx| \leqq n$ であることから、語 vx が a と c の両方を含むことはありえない。

(i) v と x が a しか含まないとき：$N_b(uwy) = N_c(uwy) = n$ である。ここで、語 s に対して $N_a(s)$ は s 内の記号 a の数を表している。反復補題から $|vx| \geqq 1$ であることから、$N_a(uwy) < n$. よって、uwy は $a^j b^j c^j$ の形をしていないことになるが、$uwy \in L$ より矛盾。

(ii) v と x が b しか含まないとき：(i) と同様の議論が使える。

(iii) v と x が a と b を含むとき：$N_c(uwy) > N_a(uwy), N_b(uwy)$ となり、$uwy \notin L$.

(iii) v と x が b と c を含むとき：(iii) と同様に $uwy \notin L$.

以上から、L は文脈自由となりえないことがわかった。

■

演習 **6.32** 文脈自由文法の反復補題をつかって、言語

$$L = \{ww \mid w \in \{a, b\}^*\}$$

は文脈自由言語でないことを証明しなさい [9, 例 18.3]。

反復補題はすべての非 CFL の証明に使えるわけではなく、次の言語の非 CFL 性を示すことができない [1, p.167]。

$$L = \{a^i b^j c^k d^\ell \,|, i = 0 \text{ または } j = k = \ell\}.$$

補題 6.3 (Ogden 1968) 文脈自由言語 L は、定数 n（反復補題と同じでよい）が存在して次の条件を満たす。

L 内の長さ n 以上の任意の語 z について、z 中の n 個以上の記号に目印（マーク）が付いているとする。このとき、次の条件を満たすように z を $z = uvwxy$ の形に表すことができる

(1) vx の中に少なくとも 1 つの目印がある
(2) vwx は高々 n 個の目印しか含まない
(3) 任意の $i \geqq 0$ について $uv^i wx^i y \in L$.

z のすべて文字に目印がつくと、CFL の反復補題になる。

第 7 章

プッシュダウンオートマトン

7.1　プッシュダウンオートマトン

実は、PDA はスタックヘッドが読み取るマス目がスタックの底であるかを認識することはできない。このため、動作前の PDA のスタックは空であるとし、動作開始時に空スタックに特別な記号 $Z_0 \in \Gamma$ をプッシュして、スタックヘッドが Z_0 を読み取るとるときには Z_0 の下にはもはやスタック文字列はない（したがってスタックの底）と PDA に認識させる工夫をする。この特別な役割のための Z_0 をスタック初期記号という。その具体的な利用法は例 7.2 に示す。

定義 7.1 プッシュダウンオートマトン (PDA: push down automaton) M を次の 7 組

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

で定義する。Q は非空な状態集合、Σ は入力アルファベット、Γ はプッシュダウンスタックのアルファベット、$q_0 \in Q$ は初期状態、$F \subseteq Q$ は受理状態、$Z_0 \in \Gamma$ はプッシュダウンスタックの初期記号である。$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$ とする。節 7.3 で明らかになることであるが、スタック記号集合 Γ は入力アルファベット Σ を含んでいるとした方が都合がよい（$\Sigma \subseteq \Gamma$）。混乱を避けるために、当面は入力記号を小文字にし、対応するスタック記号を大文字としておく配慮をすることがある。

δ は $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$ から $Q \times \Gamma^*$ の部分集合への非決定性関数（集合値を取る）$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow 2^{Q \times \Gamma^*}$ で

$$\delta(p, a_\varepsilon, Z) = \big\{(q_1, \gamma_1), \ldots, (q_{\mathrm{n}}, \gamma_n)\big\}, \quad a_\varepsilon \in \Sigma_\varepsilon, p, q_i \in Q, Z \in \Gamma_\varepsilon, \gamma_i \in \Gamma^* \quad (7.1)$$

と表される。すなわち、状態が $p \in Q$ でスタックトップ (stack top) が $Z \in \Gamma$ であるよ

うな **PDA** 状況 (p, Z) であるとき、入力 $a_\varepsilon \in \Sigma_\varepsilon$ を読みとると、状態 q_i とスタックトップが（置き換えられた）文字列 γ_i で定まる新たな PDA 状況の組集合 $\{(q_i, \gamma_i)\}$ に（集合濃度が 2 以上のときには非決定的に）遷移する (節 7.1.1 参照)。最初に空スタックに開始記号 Z_0 を積ませる動作をする PDA を考えたり、動作をさせる前に Z_0 がスタックトップに用意されている PDA を考えることもある（本質的な違いはない)。

PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ は、その内部状態とスタックという内部記憶装置を有する機械装置と見なすことができる。M の状態数は有限 ($^\#Q < +\infty$) だが、積み上がるスタック文字列 $\gamma \in \Gamma^*$ の高さ ($|\gamma|$) には制限はない。

節 7.1.4 の時点表示で改めて説明するように、PDA の計算状況は、状態 $q \in Q$ におけるスタック列 $\gamma \in \Gamma^*$ の組 (q, γ) で表される。空スタック $\gamma = \phi$ となったときは、それから先の動作が定義されておらず、動作は停止 (halt) する。

現在状況から変化があり得るのはスタック列 γ が空でなく、$\gamma = Z\gamma', \gamma' \in \Gamma^* (Z \in \Gamma)$ であるときで、次に入力記号を読み込むかどうかは状態 q とスタックトップ $Z \in \Gamma$「だけ」によって決まる。(q, Z) を状況 (q, γ) のモードという。

PDA の状態遷移（の 1 つ）を次のようにいつくかの表し方がある（図 7.1 も参照)。

$$(q_i, \gamma_i) \in \delta(p, a_\varepsilon, Z) \equiv \quad (p, Z) \xrightarrow{a_\varepsilon} \{(q_i, \gamma_i)\}_i \tag{7.2}$$

$$\equiv \quad \left\{ p \xrightarrow{a_\varepsilon, Z \to \gamma_i} q_i \right\}_i \quad \to \quad (p, Z) \overset{a_\varepsilon}{\vdash} \{(q_i, \gamma_i)\}_i. \tag{7.3}$$

これらは、例 7.2 のように、状態遷移図として PDA の動作を図示するために用いられる。PDA の状態 p とスタックトップが Z である状況 (q, Z) が入力文字 a_ε によって、状態 q_i への遷移が Z の文字列 γ_i へ書き換えを伴うとして表しており、これらはすべて同等な意味を持つ。すなわち、入力 a_ε によって状態 p である PDA のスタックトップ Z がポップされ記号列 γ_i の左端が新たにスタックトップになるようにプッシュされ、状態は q_i に遷移する。このとき、入力文字列を読む入力ヘッドは右に動いて次の文字を読む準備をする。

$\gamma_i = \phi$、空スタックになったときは PDA は動作を停止して、以降の文字列は読み込まない。また、$n = 0$ ($\delta(p, a_\varepsilon, Z) = \phi$ となる）ときには、(p, a_ε, Z) からの遷移は無定義とする。したがって、入力語を読みきらないうちに空スタックから更にポップしようとしたり、入力語を読み切ってしまうと、PDA は停止すると考える。

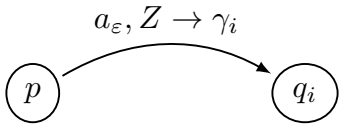


図 7.1　プッシュダウンオートマトンの状態遷移図。状態 p から入力 $a_\varepsilon \in \Sigma_\varepsilon$ とスタックヘッド Z に応じて、状況の 1 つ $(q_i, \gamma_i) \in \delta(p, a_\varepsilon, Z)$ へ遷移する様子

7.1.1　プッシュダウンオートマトンの決定・非決定性

文字列受理機械の決定性は有限オートマトンの場合（定義 2.5）と同様に、PDA においても決定性概念があることを確認しておこう。

PDA M において計算状況（時点表示）$p, Z\gamma' (Z \in \Gamma, \gamma' \in \Gamma^*)$ において、次に入力記号を読み込むかどうかはモード (p, Z) だけで一意に定まる。しかし、そこから唯一つのモードへ移行するわけではない。

このとき、外部からの入力文字列 $w \in \Sigma^*$ は PDA の入力ヘッダによって 1 文字ずつ読み取られるのであるが、PDA の挙動は読み取った文字 $a_\varepsilon \in \Sigma_\varepsilon$ とその時の PDA の内部状態 $p \in Q$ とスタックトップ $Z \in \Gamma_\varepsilon$ の組 (p, Z)（計算状況）によって定まる。この事情は、式 (7.1) で表される状態遷移関数 $\delta(p, a_\varepsilon, Z), p \in Q, a_\varepsilon \in \Sigma_\varepsilon, Z \in \Gamma_\varepsilon$ や遷移関係の表し方の式 (7.2) などから確認できる。

定義 7.2 PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ が**決定性** (deterministic) を持つとは、式 (7.2) で表されるモード (p, Z) からの遷移において、次の条件が成り立つことである：

(1) 任意の $p \in Q, Z \in \Gamma$ および $a_\varepsilon \in \Sigma \cup \{\varepsilon\}$ に対して、$\delta(p, a, Z)$ は 2 つ以上の要素を含まない

$$\left|\delta(p, a, Z)\right| \leqq 1, \qquad p \in Q, a \in \Sigma_\varepsilon, Z \in \Gamma_\varepsilon.$$

言い換えると、$(p, Z) \xrightarrow{a_\varepsilon} (q, \alpha)$ が δ にあるとき、$(q, \alpha) \neq (q', \alpha')$ であるような遷移 $(p, Z) \xrightarrow{a_\varepsilon} (q', \alpha')$ は δ にはない（高々 1 つしかない）。

(2) ε-遷移

$$(p, Z) \xrightarrow{\varepsilon} (q, \beta)$$

が 1 つだけあって、任意の $p \in Q, Z \in \Gamma$ に対して、$\delta(p, \varepsilon, Z) \neq \phi$ であるとき、どんな $b \in \Sigma$ についても $(p, Z) \xrightarrow{b} (q, \beta')$ であるような遷移は δ にはない。つまり、

(p, Z) からある 1 状態への ε-遷移があるときには、実入力 $b \in \Sigma$ による (p, Z) からの遷移がない（ε-遷移と実入力による動作との間の選択可能性はない)。

決定性でない PDA を非決定的 (non-deterministic) という。PDA は特にわらない限り、一般的には非決定的である。

決定性 PDA では、各モードについて入力が無くて（ε-動作）も実入力であっても、次のモードが高々 1 つだけ定義され、空スタックでない限り「次モード」が唯 1 つに定まる。

PDA の決定性・非決定性は、PDA の挙動を表す状態遷移図から容易にわかる。入力 a による PDA 状況に対する応答あるいは状態遷移が一意的

$$(p, Z) \overset{a}{\vdash} (q, \gamma) \quad \equiv \quad p \overset{a, Z \to \gamma}{\longrightarrow} q, \qquad p, q \in Q,\ a \in \Sigma_\varepsilon, Z \in \Gamma_\varepsilon, \gamma \in \Gamma^*$$

でないような遷移が 1 つでもあれば、PDA は非決定的である。たとえば、次のような PDA 状況の応答

$$(p, Z) \overset{a}{\vdash} \{(q_1, \gamma_1), (q_2, \gamma_2)\}$$

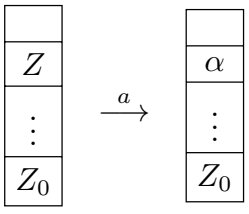
を持つ PDA は非決定的である。

7.1.2 単純決定プッシュダウンオートマトン

特に簡単な PDA として、Γ 上のスタック列 $\gamma \in \Gamma^*$ と入力記号 $a \in \Sigma$ だけでスタック列が $\delta : \Gamma \times \Sigma \to \Gamma^*$ によって変化するような（PDA の内部状態を考えなくてもよい）PDA $M = (\Gamma, \Sigma, \delta, Z_0)$ を考えることができる（内部状態 Q はスタック記号 Γ で表されると考えてもよい)。

スタックトップ $Z \in \Gamma$ は入力 $a \in \Sigma$ によって $\alpha \in \Gamma^*$ に置き換わる (α の先頭文字がスタックトップになる):

$$\delta(Z, a) = \alpha \quad \Leftrightarrow \quad Z \overset{a}{\to} \alpha.$$



計算モードの推移を定める δ が、ε-動作なしであり集合値をとらずにただ一つの計算モードを決めるために、この PDA を単純決定プッシュダウンオートマトンという。スタックが空になると動作を停止する。

この推移を

$$Z \underset{M}{\overset{a}{\Rightarrow}} \alpha$$

と記すことができ、さらに

$$\alpha_1 \underset{M}{\overset{a_1 \ldots a_n}{\Rightarrow^*}} \alpha_{n+1} = \alpha_1 \underset{M}{\overset{a_1}{\Rightarrow}} \alpha_2 \underset{M}{\overset{a_2}{\Rightarrow}} \alpha_3 \ldots \alpha_n \underset{M}{\overset{a_n}{\Rightarrow}} \alpha_{n+1}$$

と短縮して表す。

例 **7.1** $\Gamma = \{Z_0, A, B\}, \Sigma = \{a, b\}$ について $M = (\Gamma, \Sigma, \delta, Z_0)$

$$\delta : Z_0 \overset{a}{\to} A, \quad A \overset{a}{\to} AB,$$
$$A \overset{b}{\to} \varepsilon, \quad B \overset{b}{\to} \varepsilon.$$

は

$$L(M) = \left\{ a^i b^i \,|\, i \geqq 1 \right\}$$

を受理する

演習 **7.1** 単純決定オートマトン $M = (\{Z_0, A, B\}, \{a, b, c\}, \delta, Z_0)$

$$\delta = \left\{ Z_0 \overset{c}{\to} \varepsilon, \quad Z_0 \overset{a}{\to} Z_0 A, \quad A \overset{a}{\to} \varepsilon, \quad Z_0 \overset{b}{\to} Z_0 B, \quad B \overset{b}{\to} \varepsilon. \right\}$$

は

$$L(M) = \left\{ xcx^R \,|\, x \in \{a, b\}^* \right\}$$

を受理することを説明する。

単純決定性文脈自由文法

文脈自由文法 $G = (\Sigma_N, \Sigma_N, P, S)$ において、生成規則 P が、任意の $A \in \Sigma_N$ について

$$A \to a\alpha, \qquad a \in \Sigma_T, \alpha \in \Sigma_N^*$$

だけであるとき（$A \to a\alpha \in P$ であるとき、$\alpha \neq \beta \in \Sigma^*$ であるいかなる β について $A \to a\beta \notin P$）、文脈自由文法 G は単純決定という。

単純決定文法 $G = (\Sigma_N, \Sigma_N, P, S)$ と単純決定プッシュダウンオートマトン $M = (\Gamma, \Sigma, \delta, Z_0)$ との間には 1 対 1 の対応があり、直接に次のように構成（逆構成も）できる：

$$\gamma = \Sigma_N, \quad Z_0 = S, \Sigma = \Sigma_T$$

および、$A \longrightarrow a\alpha \in P$ $(a \in \Sigma_T, \alpha \in \Sigma_N^*)$ であるときにだけ $A \xrightarrow{a} \alpha \in \delta$.

$$\begin{array}{l} A \to a\alpha \\ \text{単純決定文法} \end{array} \quad \Leftrightarrow \quad \begin{array}{l} A \xrightarrow{a} \alpha \\ \text{単純決定 PDA} \end{array}$$

演習 7.2 演習 7.1 の単純決定 PDA から同等な単純決定文法 G を構成し、その導出過程を調べなさい。

演習 7.3 単純決定文法 $G = (\{S, A, B\}, \{a, b\}, P, S)$、

$$P: \quad \{S \to aA, \quad A \to b, \quad A \to aAB, \quad B \to b\}$$

から定まる言語 $L(G)$ を決定し、同等な単純決定 PDA を構成してその動作を調べなさい。

7.1.3 PDA の受理

ここでは PDA の入力文字列の受理を受理状態と空スタックによって定義する。PDA による入力文字列の受理には次の 3 つ

- 状態が受理状態で、かつスタックが空
- 状態が受理状態
- スタックが空

が考えられる。節 7.2 で、この 3 つの条件のどれによって PDA の受理を定めても、受理される言語クラスは変わらないことが示される [1, 定理 5.1 と定理 5.2]。ここでは、以下に「受理状態と空スタック」によって PDA の受理を定めることにする。

PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ において、実入力列 $w = w_1 w_2 \dots w_n \in \Sigma^*$ を受理する動作は実入力文字数である n 回でなく、$m \geq n$ 回必要である。m 回のうち $m - n$ 回は ε-動作する。

実入力列 $w = w_1 w_2 \dots w_n$ の間に $m - n$ 個の ε を挟み込んだ列を $a_0 a_1 \dots a_{m-1} \in \Sigma_\varepsilon^*$ と表す。実入力 $w_1, w_2, \dots, w_n$ のどこに $m - n$ 個の ε を挟み込むかは任意（非決定的）で、PDA の最初と最後の受理条件を満たし、$a_0 a_1 \dots a_{m-1}$ の各「入力」a_i ごとに PDA の状態遷移関数と整合した動作が行われた結果として、$w = w_1 w_2 \dots w_n$ が受理されればよいと考える。$m - n$ 個の ε-動作をどこに挿入するかの自由度があるのは、PDA の非決定性によるものである。

定義 7.3 (PDA の受理 (1)) PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ において、入力列 $w = w_1 w_2 \dots w_n \in \Sigma^*$ が PDA M に最終状態の集合 F で受理されるのは次の条件 (1)(2)(3)

を満たす $w = a_0 a_1 \ldots a_{m-1}\,(m \geq n)$ となる $a_0, a_1, \ldots, a_{m-1} \in \Sigma \cup \{\varepsilon\}$、$p_0, p_1, \ldots, p_m \in Q$ および $z_0, z_1, \ldots, z_m \in \Gamma^*$ が存在することである：

(1) $p_o = q_0, p_m \in F$.

(2) $z_0 = \varepsilon, z_m = \varepsilon$.

(3) $0 \leqq i \leqq m-1$ に対して、$(p_{i+1}, \gamma) \in \delta(p_i, a_i, Z)$. ただし、$Z \in \Gamma \cup \{\varepsilon\}, \gamma, \beta \in \Gamma^*$ が存在して

$$z_i = Z\beta, \quad z_{i+1} = \gamma\beta.$$

例 **7.2** 図 7.2 で示される状態遷移を有する PDA を考える（空スタックにまず開始記号 Z_0 を積ませている）。入力記号 $\Sigma = \{a, b\}$, スタック記号 $\Gamma = \{Z_0, A, B\}$ である。この PDA は a, b からなる言語

$$\begin{aligned} L &= \left\{ ww^R \,|\, w \in \{a,b\}^*, w^R \text{ は } w \text{ の反転文字列} \right\} \\ &= L(G = (\{S\}, \{a,b\}, \{S \to aSa \,|\, bSb \,|\, \varepsilon\}, S)) \end{aligned}$$

を受理することが、以下のようにしてわかる。

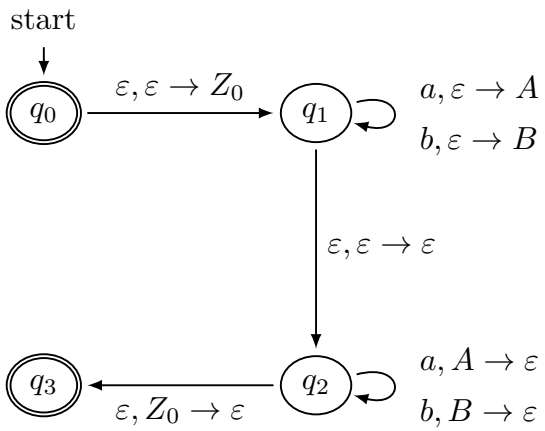


図 7.2　言語 $L = \left\{ ww^R \,|\, w \in \{a,b\}^* \right\}$ を受理する PDA。同じ言語を受理する図 7.7 の PDA も参照。

スタート時に ε-動作によって、まず入力なしで $q_0 \overset{\varepsilon, \varepsilon \to Z_0}{\longrightarrow} q_1$ によって空スタックにスタック初期記号 Z_0 がプッシュされ、状態 q_0 から q_1 に状態遷移する。

次いで、$q_1 \overset{a, \varepsilon \to A}{\longrightarrow} q_1$ によって、入力記号 a に対してはスタックに記号 A をプッシュ、または $q_1 \overset{a, \varepsilon \to A}{\longrightarrow} q_1$ によって、入力記号 b に対してはスタックに記号 B をプッシュする。

この間、状態は q_1 に留まる。

ただし一方、この間いつでも、ε-動作によって入力なしで $q_1 \overset{\varepsilon,\varepsilon\to\varepsilon}{\longrightarrow} q_2$ によって、いつでも状態 q_1 から q_2 に状態遷移することを許している。つまり、この PDA は a, b からなる入力文字列のすべての箇所で入力文字なしの ε-動作によって q_2 への状態遷移をいつでも許していることに注意する。

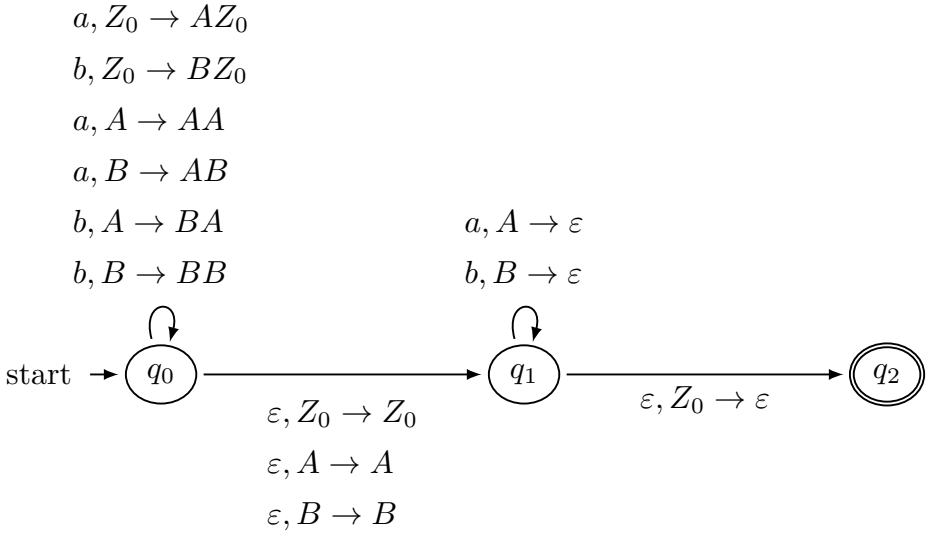
状態 q_2 では、入力記号 a に対してはスタックに記号 A をポップするような状態遷移 $q_2 \overset{a,A\to\varepsilon}{\longrightarrow} q_2$、または入力記号 b に対してはスタックに記号 B をポップする状態遷移 $q_2 \overset{b,B\to\varepsilon}{\longrightarrow} q_2$ をすることで状態 q_2 にとどまりながらスタックトップをポップし続ける。

目的の語 ww^R ($w \in \{a, b\}^*$) の受理のためには、w と w^R の境目で状態遷移 $q_1 \overset{\varepsilon,\varepsilon\to\varepsilon}{\longrightarrow} q_2$ した場合にだけ、状態 q_2 において入力文字 a または b に対して、積み上がっているスタック記号をスタックトップから対応したスタック記号 A または B をポップしてスタックの底（スタックヘッドが初期記号 Z_0 を認識する）に到達することである。つまり、語 w によって積み上がったスタックを w の逆順 w^R の入力によって、スタックトップを Z_0 とすることになる。

スタックヘッドがスタック初期記号 Z_0 を見つけたときには、ε-動作によって入力記号なしで $q_2 \overset{\varepsilon,Z_0\to\varepsilon}{\longrightarrow} q_3$ と受理状態 q_3 に状態遷移し Z_0 がポップされ、スタックが空になる。

こうして ww^R が受理状態と空スタックにって文字列 ww^R が受理される。

演習 7.4 開始記号 Z_0 が既にスタックトップに積まれた状態から動作する PDA を考えてみよう。この PDA の動作を調べて受理する言語を決定しなさい。



演習 **7.5** a, b, c からなる言語

$$L = \left\{ w\, c\, w^R \mid w \in \{a,b\}^*, w^R \text{ は } w \text{ の反転文字列} \right\}$$

を受理する PDA を構成し、その動作を説明しなさい。合わせて L を生成する文脈自由文法を与えなさい（演習 6.13 参照）。

7.1.4 時点表示

PDA に入力される記号列は左端から入力ヘッドによって読み取られていく（既読入力列となる）。PDA の動作は状態間遷移 $p \overset{a,Z\to\gamma}{\longrightarrow} q$ のような局所的な表し方の他に、PDA 全体の挙動を次のように時点表示として表すこともできる。

定義 7.4 (時点表示) PDA の現在の到達状態 $q \in Q$、未読入力記号列 $w \in \Sigma^*$、スタック記号列 $\gamma \in \Gamma^*$（スタックトップは左端）の 3 組 (q, w, γ) を**時点表示** (instantaneous configuration) という。

いま、これから入力ヘッドが未読記号 a（その右側以降の未読文字列は $x \in \Sigma^*$）を読み、同時にスタックヘッドはスタックトップ Z（その下のスタック記号列を $\beta \in \Gamma^*$）を読むとする。

このとき、PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ の動作が $(q, \alpha) \in \delta(p, a, Z)$ であることを、時点表示の遷移を $Q \times \Sigma^* \times \Gamma^*$ 上の関係 $\underset{M}{\vdash}$ として次のように表す。

$$(p, aw, Z\beta) \underset{M}{\vdash} (q, w, \gamma\beta)$$

このとき、遷移 $(q, \alpha) \in \delta(p, a, Z)$ によって、PDA M は時点表示 $(p, aw, Z\beta)$ から $(q, w, \alpha\beta)$ へ合法的に動作するという（$a = \varepsilon$ のときは入力記号が読み込まれずに動作する ε-遷移である）。M の**計算**とは、合法的な時点表示列である。

関係 $\underset{M}{\vdash}$ の反射的推移閉包を $\overset{*}{\underset{M}{\vdash}}$ と書くことにする。このとき、時点表示の列が

$$(p_0, w_0, \alpha_0) \underset{M}{\vdash} (p_1, w_1, \alpha_1) \underset{M}{\vdash} \dots \underset{M}{\vdash} (p_n, w_n, \alpha_n)$$

であれば、

$$(p_0, w_0, \alpha_0) \overset{*}{\underset{M}{\vdash}} (p_n, w_n, \alpha_n)$$

と書くことができる。到達に要するステップ数が k であるときには $\overset{k}{\underset{M}{\vdash}}$ と記すこともある。

注意 7.1 ここで定義した PDA は非決定性で、時点表示 $(p_i, w_i, \alpha_i) \underset{M}{\vdash} (p_{i+1}, w_{i+1}, \alpha_{i+1})$ は可能な動作列の経路の 1 つを表しているに過ぎず、$(p_0, w_0, \alpha_0) \overset{*}{\underset{M}{\vdash}} (p_n, w_n, \alpha_n)$ は時点表示 (p_0, w_0, α_0) から (p_n, w_n, α_n) に到達する 1 つの経路を表している（複数あるかもしれない)。

計算における推移には次の注意が必要である [1, 第 2 版節 6.1]

定理 7.1 PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ において
$(p, x, \alpha) \overset{*}{\underset{M}{\vdash}} (q, y, \beta), \quad x, y \in \Sigma^*, \alpha, \beta \in \Gamma^*$
ならば、任意の $w \in \Sigma^*$ と $\gamma \in \Gamma^*$ に対して
$(p, xw, \alpha\gamma) \overset{*}{\underset{M}{\vdash}} (q, yw, \beta\gamma).$

証明　$(p, xw, \alpha\gamma)$ から $(q, yw, \beta\gamma)$ への計算ステップ数に関する帰納法。M の計算は w に左右されず $(p, xw, \alpha\gamma) \overset{*}{\underset{M}{\vdash}} (q, yw, \beta\gamma)$ は正当。

合法的計算列では、各時点表示の未読入力列に任意の入力列またはスタック列に任意のスタック列を付け加えても計算は合法的。

しかし、この逆については共通する未来の未読記号列 w を計算過程から取り去ることは可能だが、スタックの底からみて共通のスタック記号列 γ については削除できない（γ を必要とする場合があり得る)。

定理 7.2 PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ において
$(p, xw, \alpha) \overset{*}{\underset{M}{\vdash}} (q, yw, \beta), \quad x, y, w \in \Sigma^*, \alpha, \beta \in \Gamma^*$
ならば
$(p, x, \alpha) \overset{*}{\underset{M}{\vdash}} (q, y, \beta).$

時点表示の言葉で PDA の受理は次のように定義される。

定義 7.5 (PDA の受理 (2)) PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ が受理状態と空スタックによって入力記号を受理することで定まる言語 $L(M)$ を次にように定義する。

$$L(M) = \left\{ w \in \Sigma^* \,\middle|\, \text{ある } q_f \in F \text{ について } (q_0, w, Z_0) \overset{*}{\underset{M}{\vdash}} (p_f, \varepsilon, \varepsilon) \right\}.$$

ここでは、最初の時点表示を (q_0, w, Z_0) は、動作直後にはスタックの底に初期記号 Z_0 を置いてある（動作直後に ε-動作でからスタックに Z_0 をプッシュした）とみなしている

(図 7.2 との整合性を保っている)。

7.2 PDA 受理の等価性

節 7.1.3 で PDA の受理について触れたように、PDA の入力文字列受理の定義には次の 3 つ

(1) PDA M_1 が 受理状態に達し、かつスタックが空（ここで採用している定義）
(2) PDA M_2 が受理状態に達して受理
(3) M_3 のスタックが空になって受理

がある。それぞれは ε-動作によって開始時にスタックの底に開始記号 Z_0 が置かれるとする。これら 3 つの条件のどれを採用しても受理される言語クラスは変わらないことを説明しよう。

M_2 が M_1 を、M_3 が M_2 を、M_1 が M_3 を模倣することができるため、これらの受理条件は等価である [9, p.261]。

7.2.1 M_2 は M_1 を模倣する

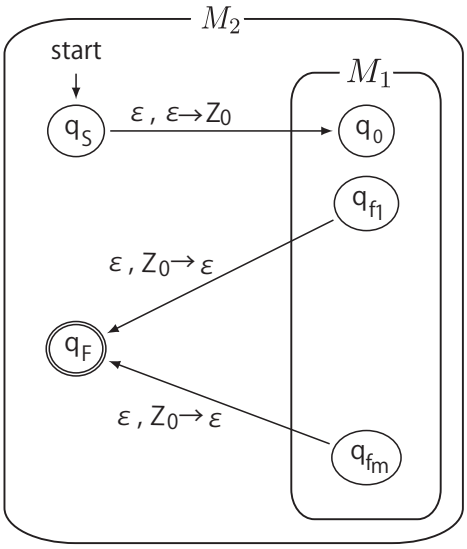


図 7.3　M_2 が M_1 を模倣する。M_1 は状態が受理状態かつスタックが空で入力を受理する PDA で、初期状態 q_0 から始まり、受理状態 q_{f_1} から q_{F_m} を持つ。このとき M_1 を模倣する M_2 は、新たに初期状態 q_S と受理状態 q_F を追加した上で、示された ε-動作を加えて得られる PDA である。M_1 では q_0 を初期状態から出発して受理状態 $\{q_{f_i}\}, i = 1, \ldots, m$ のどれかに到達したときには空スタックとなったときに受理とする。M_2 は初期状態 q_S から出発して ε-動作で q_0 に遷移し、かつスタックの底に Z_0 を追加する。$\{q_{f_i}\}$ のいずれかに達した時、ε-動作で Z_0 をポップして空スタックとして受理状態 q_F に達して受理することで M_1 を模倣する。M_1 の受理条件 (1) は M_2 における受理条件 (2) と等価である。

7.2.2 M_3 は M_2 を模倣する

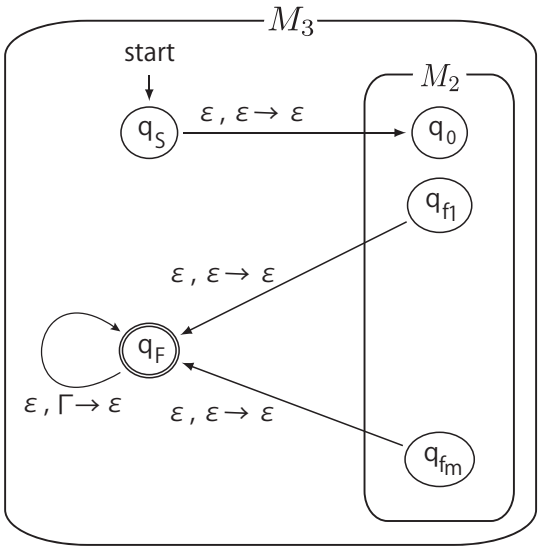


図 7.4　M_3 が M_2 を模倣する。M_2 が受理状態に達したとき、M_3 は ε-動作で受理状態 q_F に遷移させ、その後 ε-動作で任意のスタック記号をポップして空スタックになるまで続けて M_2 を模倣する。M_3 として、新たな初期状態 q_S を導入せずに $q_S = q_0$ としてよく、また、この図のように M_3 で新たに追加した $F_3 = \{q_F\}$ を M_3 の受理状態とぜずに受理状態集合 F_3 を ϕ とし、空スタックになったときにだけ受理するようにもできる。

7.2.3 M_1 は M_3 を模倣する

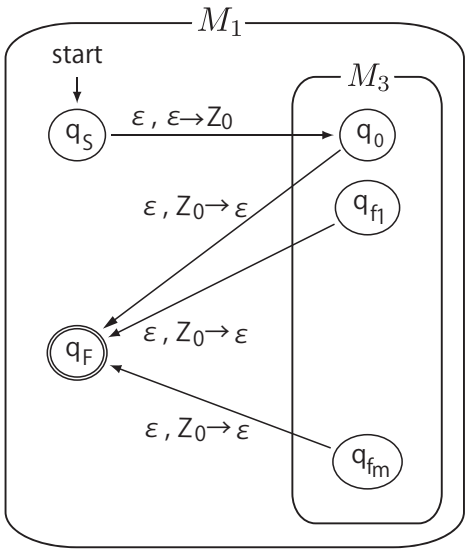


図 7.5　M_1 が M_3 を模倣する。M_3 が空スタックだけで受理するとき M_1 がこれを模倣するには、M_1 は受理状態かつ空スタックで受理しなければならない。M_1 でプッシュ・ポップされる $Z_0' \in \Gamma_1$ は M_3 のスタック記号集合 Γ_3 には含まれない記号である。

次の例では、入力アルファベット $\Sigma = \{a, b\}$ とスタック記号にそれぞれ対応する A と B を導入しましたが、混乱しなければ $\Gamma = \{Z_0, a, b\}$ としても構いません。

例 **7.3** $Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{Z_0, A, B\}, F = \{q_2\}$ とする PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ を次のように定める。

q	a	Z	$\delta(q,a,Z)$
q_0	a	Z_0	$\{(q_0, AZ_0), (q_1, AZ_0\}$
q_0	b	Z_0	$\{(q_0, BZ_0), (q_1, BZ_0\}$
q_0	a	A	$\{(q_0, AA), (q_1, AA\}$
q_0	a	B	$\{(q_0, AB), (q_1, AB\}$
q_0	b	A	$\{(q_0, BA), (q_1, BA\}$
q_0	b	B	$\{(q_0, BB), (q_1, BB)\}$
q_1	a	A	$\{(q_1, \varepsilon)\}$
q_1	b	B	$\{(q_1, \varepsilon)\}$
q_1	ε	Z_0	$\{(q_2, \varepsilon)\}$

表 7.1　q を内部状態、a を入力文字,Z をスタックトップしたときの遷移関数 δ

演習 7.6 例 7.3 で与えられた PDA M の入力文字列 aabaabaab に対する時点表示列を書いてみなさい。

演習 7.7 例 7.3 で与えられた PDA M に対して、

$$L(M) = \{ww^R \,|\, w^R \text{ は } w \text{ の反転文字列、} w \in \{a,b\}^*\}$$

であることを示しなさい（演習 6.12 参照)。

7.3　PDA の例

この節でも、入力アルファベット $\Sigma = \{a, b, \dots\}$ とスタック記号にそれぞれ対応する $A, B, \dots$ を導入しましたが、混乱しなければ $\Gamma = \{Z_0, a, b, \dots\}$ としても構いません。

例 7.4 図 7.6 の状態遷移図を持つ PDA $\big(\{q_0,q_1,q_2,q_3\}, \{a,b\}, \{Z_0,A\}, \delta, Z_0, \{q_0,q_3\}\big)$ は、文脈自由文法の例 6.6 $\big(\{S\}, \{a,b\}, \{S \to aSb \,|\, \varepsilon\}, S\big)$ が生成する言語 $\{a^i b^i \,|\, i \geqq 0\}$ を受理する。

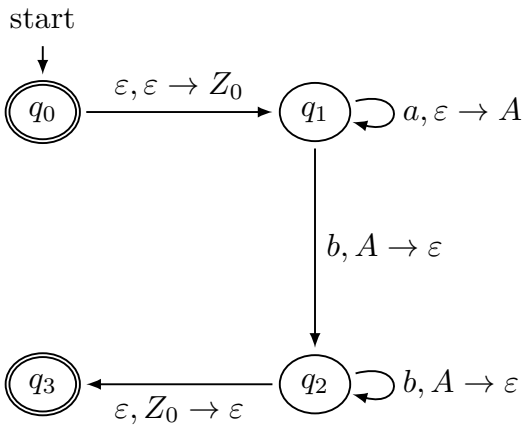


図 7.6　言語 $\{a^i b^i \mid i \geqq 0\}$ を受理する PDA。文脈自由文法の例 6.6 参照。

例 **7.5** 図 7.2 の PDA が受理する言語 $L = \{ww^R \mid w \in \{a,b\}^*, w^R \text{ は } w \text{ の反転文字列}\}$ は、CFG 文法 $G = (\Sigma_N, \Sigma_T, P, S)$

$$\Sigma_N = \{S\}, \quad \Sigma_T = \{a, b\},$$
$$P : S \to aSa \mid bSb \mid \varepsilon.$$

によって生成されることに注意しよう [9, p.172]。

この CFG は図 7.7 の PDA $M = (Q = \{q_0.q_1, q_2\}, \Sigma = \{a,b\}, \Gamma = \{Z_0, S, A, B\}, \delta, q_0, Z_0, F = \{q_2\})$ の動作に対応していることを確かめよう。スタック初期記号を S としている。図 7.7 では、状態遷移 $q_1 \overset{\varepsilon, C \to \gamma}{\longrightarrow} q_1$ で、スタックトップの非終端記号 C に対して最左導出 $C \to \gamma$ を適用する。また、終端記号 c を読んだ時にスタックトップが終端記号に対応した C のときはそれをポップして $q_1 \overset{c, C \to \varepsilon}{\longrightarrow} q_1$ と、次の最左導出を行うことを繰り返す。これらに対応する生成規則 P は

$$P : \{C \to \gamma\}$$

と表されていると考える（定理 7.3 参照）。

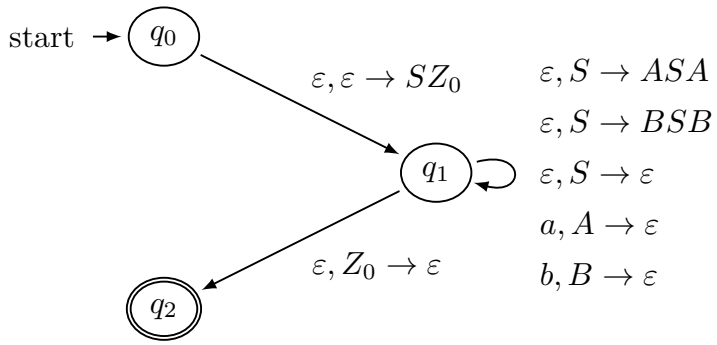


図 7.7　言語 $L = \{ww^R \mid w \in \{a,b\}^*\}$ を受理する PDA。同じ言語を受理する図 7.2 の PDA も参照。

演習 7.8 図 7.7 の PDA は、置き換え規則 $P : S \to aSa \mid bSb \mid \varepsilon$ を持つ CFG と同じ言語 $\{ww^R \mid w \in \{a,b\}^*\}$ を受理することを説明しなさい [9, 図 20.3]。

演習 7.9 $\Sigma = \{a,b\}, \Gamma = \{Z_0, A, B\}$ についての状態遷移図 7.8 を持つ PDA はどんな文字列を受理するか説明しなさい [9, 図 19.8]。
[ヒント] 図 7.8 の PDA が受理する言語 $\{w \in \{a,b\}^* \mid N_a(w) = N_b(w)\}$(例 6.15 参照)

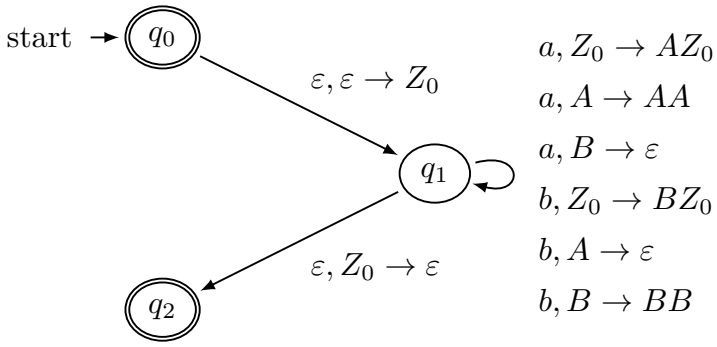


図 7.8　$\{w \in \{a,b\}^* \mid N_a(w) = N_b(w)\}$ を受理する PDA

演習 7.10 $\Sigma = \{a,b\}, \Gamma = \{Z_0, A\}$ についての状態遷移図 7.9 を持つ PDA はどんな文字列を受理するか説明しなさい [9, 問題 19.3]。
[ヒント] 図 7.9 の PDA が受理するためには、最後に連続的に b を読み込んで A をポップして Z_0 だけを残す必要がある。

このとき、$|b^j|$ は $|a^i|$ より $j - i$ だけ長い。あらかじめ a を i 回読み込んで A 達をプッシュする際、入力 b 用にさらに A を余計に $(j - i)$ 回プッシュしておくことが必要になる。

a を読んだときに A と AA のどちらをプッシュするかは非決定的に決める。

この PDA で受理される言語は $L = \left\{a^i b^j \,\middle|\, 0 \leqq i \leqq j \leqq 2i\right\}$(例 6.17 参照)。$L$ は次の $G_=(\{S, B\}, \{a, b\}, P, S)$ で生成できる：

$$\begin{aligned} P: \quad & S \to aSbB \,|\, \varepsilon, \\ & B \to b \,|\, \varepsilon. \end{aligned}$$

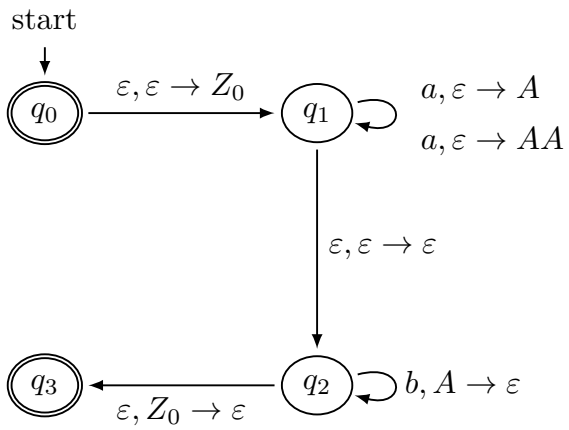


図 7.9　言語 $\{a^i b^j \,|\, 0 \leqq i \leqq j \leqq 2i\}$ を受理する PDA

演習 7.11 $\Sigma = \{a, b, c\}, \Gamma = \{Z_0, A, B\}$ についての状態遷移図 7.10 を持つ PDA はどんな文字列を受理するか説明しなさい [9, 問題 19.2]。

[ヒント] 図 7.10 の PDA が受理する言語 $\left\{a^i b^j c^k \,|\, i = j \text{ または } j = k, \quad i, j, k \geqq 0\right\}$(例 6.16 参照)

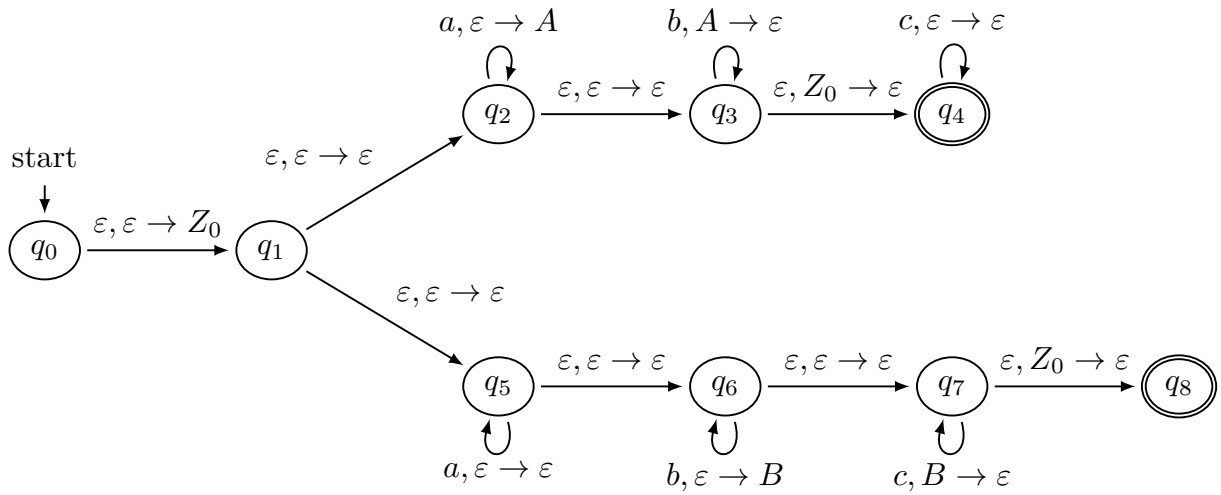


図 7.10　$\Sigma = \{a, b, c\}$ からなるどんな文字列を受理する PDA か。

演習 7.12 $\Sigma = \{a, b, c\}, \Gamma = \{Z_0, A, B\}$ についての状態遷移図 7.11 を持つ PDA はどんな文字列を受理するか説明しなさい [3, 例 2.10]。
[ヒント] 図 7.11 の PDA が受理する言語 $\left\{a^i b^j c^k \,|\, i = j \text{ または } i = k,\ i, j, k \geqq 0\right\}$

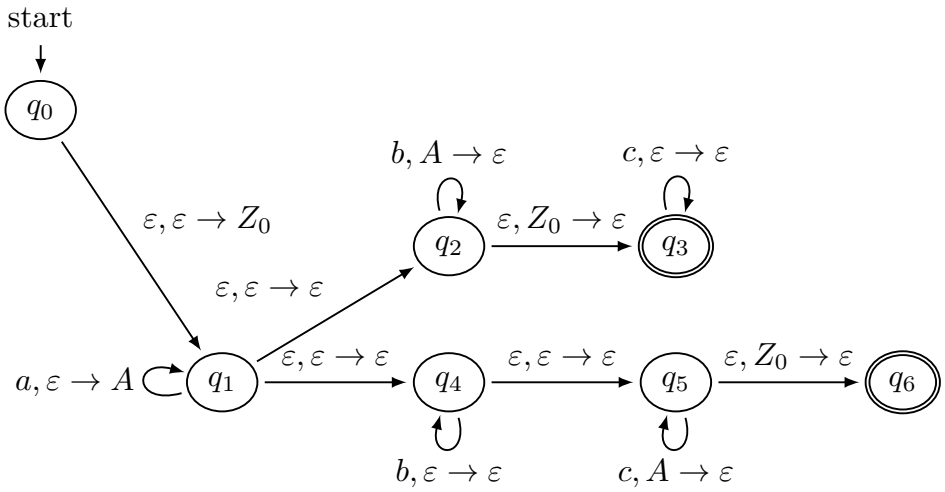


図 7.11　$\Sigma = \{a, b, c\}$ からなるどんな文字列を受理する PDA か。

7.4　CFG と PDA の関係

この節では、入力アルファベット（非終端記号）$\Sigma = \{a, b, \dots\}$ もスタック記号に入れて $\Gamma = \{Z_0, \cdots, a, b, \dots\}$ とする（いままでは、あえて混乱を避けるために $a \in \Sigma$ に対応するスタック記号 $A \in \Gamma$ を導入してきた）。

図 7.7 の例が示すように、与えられた CFG $G = (\Sigma_T, \Sigma_N, P, S)$ が生成する PDA M は次のようにして直接構成することができるする（図 7.12）。

(1) 開始状態 q_0 から、スタックにスタック初期化記号 Z_0 さらに G の開始記号 S をプッシュして、状態 q_1 に遷移。
(2) スタックトップが非終端記号 $A \in \Sigma_N$ のとき、非決定的に G の書き換え規則 $P : A \to \alpha$（$\alpha \in (\Sigma_T \cup \Sigma_N)^*$）を選んで、$A$ を α で置き換える（A をポップして α をプッシュ）。状態は q_1 に留まる。
(3) スタックトップが終端記号 $a \in \Sigma_T$ のとき、もし入力ヘッドが読み込もうとする入力記号が a ならば、スタックトップ a をポップし、入力ヘッドを 1 増すだけ右に移動して次の入力文字を読む準備をする。
(4) スタックトップが Z_0 のとき、ε-動作で受理状態 q_2 に遷移。

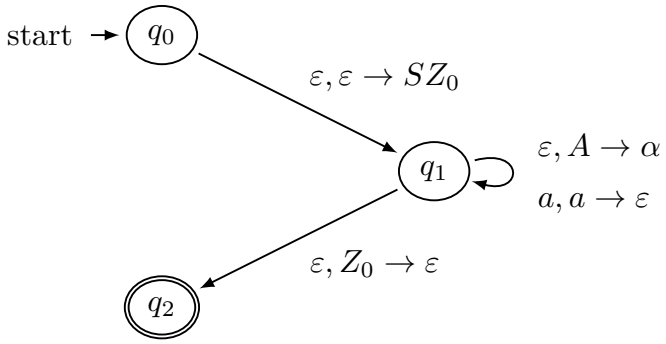


図 7.12　CFG $G = (\sigma_T, \sigma_N, P, S$ が生成する CFL $L = L(G)$ を受理する PDA M。

定理 7.3 文脈自由言語を生成する文法自由文法 G に対して、$L(G)$ を受理する PDA M が存在する。

証明　CFG $G = (\Sigma_T, \Sigma_N, P, S)$ から構築される PDA を $M = (Q = \{q_0, q_1, q_2\}, \Sigma = \Sigma_T, \Gamma = \Sigma_T \cup \Sigma_N \cup \{Z_0\}, \delta, q_0, Z_0, F = \{q_2\})$ とする。

状態遷移関数 δ を、図 7.12 のように、$P : A \to \alpha, (\alpha \in \Sigma_T \cup \Sigma_N)^*$ に対して

$$q_1 \overset{\varepsilon, A \to \alpha}{\longrightarrow} q_1,$$

すべての $a \in \Sigma_T$ に対して

$$q_1 \overset{a, a \to \varepsilon}{\longrightarrow} q_1.$$

こうして定めた PDA M は CFG G が生成する終端記号列に対して ε-動作を伴って非決定的に動作する。ただし、この PDA M は CFG が生成する文字列の導出木を決定するわけではない。非決定性による同時並行的計算でうち、正しい導出木について行って PDA 計算だけが受理に成功する。

CFG G が生成する言語を受理するにはこの PDA の非決定性が本質的である。 ■

定理 7.4 PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F$ で受理される任意の言語 $L(M)$ に対して、$L(M)$ を生成する文脈自由文法 $G = (\Sigma_N, \Sigma, P, S)$ を構成することができる。

証明　G の構成方法: HMU(定理 6.14) で帰納法によって証明 [1, 定理 5.4]。

G の非終端記号集合 Σ_N は開始記号 S 以外に、記号 $[p, A, q]$（ただし、$p, q \in Q$,$A \in \Gamma$）を集めたものである。ここで、PDA M の時点表示である内部状態 p とそのスタック列状態 A による入力列 x の計算において

$$(p, x, A) \overset{*}{\underset{M}{\vdash}} (q, \varepsilon, \varepsilon)$$

あるとき（M が入力列によって状態 p で始まり q に至る動作に置いてスタックから A を消去)、かつそのときに限り G の導出において

$$[p, A, q] \overset{*}{\underset{G}{\Rightarrow}} x.$$

このための次のように生成規則 P を作る:

(1) すべての p について、初期状態 q_0 から p に達してスタック初期記号 Z_0 を消去して文字列 x を受理するようなすべての x を生成するための規則

$$S \to [q_0, Z_0, r]$$

(2) $\delta(p, a, A) \ni (q_1, B_1 \dots B_n)(a \in \Sigma_\varepsilon, B_i \in \Gamma)$ であるとき、任意の状態列 $q_2, \dots, q_{n+1} \in Q$ に対して生成規則

$$[p, A, q_{n+1}] \to a[q_1, B_1, q_2] \dots [q_n, B_n, q_{n+1}]$$

特に、$\delta(p, a, A) \ni (q_1, \varepsilon)$ のときは

$$[p, A, q_1] \to a.$$

こうして構成された $G = (\Sigma_N, \Sigma, P, S)$ において、$L(G) = L(M)$ である。

演習 7.13 言語 $\left\{ww^R \,|\, w \in \{a, b\}\right\}$ は非決定性 PDA を使って、スタックの連続積み上げと、その折返しに対応する連続ポップの動作パスが存在することによって構成することができた。しかしながら、補題 6.2 の CFG の反復補題から、言語 $\{ww \,|\, w \in \{a, b\}^*\}$ は CFG では生成できない（演習 6.32）。スタックを有する PDA でこの言語を受理できないことを説明しなさい。

第 8 章

Turing 機械

1936 年に A.M.Turing の論文 [21] において、「機械的な手順」による計算を単純な有限的な基本動作に帰着させたものを今日では **Turing** 機械 (TM: Turing machine) と呼んでいる。Turing はこの機械によって様々な関数が計算できること（計算可能数）を示すと同時に、計算不可能な例も与え（Turing 機械が停止するかどうかは決定不可能）Gödel の不完全性定理と同等な結果を得た。

Turing 機械 TM は「機械的に実行できる手順」を定義する計算モデルとして数学的に導入されたが、「機械的に実行できる手順」と称するものが厳密に定義されているわけではないため、TM で計算できる問題のクラスと「機械的に実行できる手順」で計算できる問題のクラスとは比べようがなく必ずしも一致しない（後者が前者を含んでいる）。そこで Church-Turing がこれらが一致すると考えることを提唱した。

定義 8.1 (Church-Turing の提唱) 機械的な手順で計算されるクラスを TM で計算されるクラスとみなす (Church-Turing thesis)：

機械的な手順で計算されるクラス = TM で計算されるクラス

TM で計算可能数を帰納関数と呼ぶと、Church-Turing の提唱とは直観的な概念「計算できる関数」を帰納的関数のクラスと同一視することである。

8.1 Turing 機械の定義

TM は有限内部状態 Q を有し、入力文字列を載せたマス目に区切られた片側（また両側無限）テープをヘッドを記号を読み、内部状態に応じてテープ記号を書き換えヘッドを

左右に移動させながら動作を繰り返す。最初に与えた入力文字列の以外のテープのマス目は空白記号で埋められている。

定義 8.2 Turing 機械 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, q_H)$ を有限状態集合 Q、初期状態 $\boldsymbol{q}_0 \in Q$, 停止状態 $q_H \in Q$、入力記号 $\Sigma \subset \Gamma$（テープ記号 Γ）, 空白 $B \in \Gamma$（$B \notin Q$）として、状態遷移関数 $\delta : Q \times \Gamma \to Q \times \gamma \times D$ を次で定義する。

$$\delta(q, a) \mapsto (p, b, D), \qquad d \in \{L, R\}.$$

内部状態 p のときヘッドがテープ記号 a を読むと、状態 q に遷移しテープ記号を b に書き換えてヘッドを D に移動する（図 8.1）。TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, h)$ に対して、**5** つ組集合 K を

$$K = \left\{ pabDq \,|\, \delta(q, a) = (p, b, D) \right\}$$

を与えることによって定めることもできる。

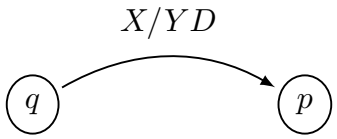


図 8.1　TM の状態遷移 $\delta(q, a) \mapsto (p, b, D)$：ヘッド移動 $D \in \{L, R\}$ または $D \in \{\leftarrow, \rightarrow\}$

8.2　TM の計算状況（時点表示）

定義 8.3 TM M において、テープ上に記号列 $X_1 X_2 \ldots X_n$ が書かれており、他のマス目はすべて空白記号 B で埋められていて、内部状態が q でヘッドが記号 X_i の位置にあることを

$$X_1 \ldots X_{i-1} q X_i \ldots X_n = \alpha q X_i \beta, \quad \alpha = X_1 \ldots X_{i-1}, \beta = X_{i+1} \ldots X_n$$

この表現 $\alpha q X_i \beta$ を TM の計算状況 (configuration) または時点表示 (instantaneous description) という。

定義 8.4 (TM の計算) 計算状況の推移列 $\underset{M}{\vdash}$ を TM の計算といい、次のようになる：

(1) $(q, X_i) = (p, Y, L)$（ヘッドが左移動）⇒

$$X_1 \dots X_{i-1} q X_i x_{i+1} \dots X_n \underset{M}{\vdash} X_1 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$$

ただし、次の例外的場合がある。

- $i = 1$ はヘッドが左の空白記号 B へ動くとき

$$q X_1 X_2 \dots X_n \underset{M}{\vdash} p B Y X_2 \dots X_n$$

- $i = n$ かつ $Y = B$ のとき

$$X_1 X_2 \dots X_{i-1} q X_n \underset{M}{\vdash} X_1 X_2 \dots X_{n-2} p X_{n-1}$$

(2) $\delta(q, X_i) = (p, Y, R)$（ヘッドが右移動）⇒

$$X_1 \dots X_{i-1} q X_i X_{i+1} \dots X_n \underset{M}{\vdash} X_1 \dots X_{i-2} X_{i-1} Y p X_{i+1} \dots X_n$$

ただし、次の例外がある。

- $i = n$ のとき X_{i+1} は空白記号 B

$$X_1 X_2 \dots X_{n-1} q X_n \underset{M}{\vdash} X_1 X_2 \dots X_{n-1} Y p B$$

- $i = 1$ かつ $Y = B$ のとき

$$q X_1 X_2 \dots X_n \underset{M}{\vdash} p X_2 \dots X_n$$

定義 8.5 (入力の受理) TM M の初期状態 q_0 において入力記号列 $a_1 \dots a_m \in \Sigma^*$ を初期計算状況 $\mathcal{C}_0 = q_0 a_1 \dots a_m$ とする。このとき、計算状況の列 $\mathcal{C}_0, \mathcal{C}_1, \cdots, \mathcal{C}_r$、

$$\mathcal{C}_i \vdash_M \mathcal{C}_{i+1}, \qquad 0 \leqq i < r$$

があり、$\mathcal{C}_r = \alpha_r q_H \beta_r$（$\alpha_r, \beta_r \in \Gamma^*, q_H \in F$）となって受理状態に達したとき、TM は入力 $a_1 \dots a_m$ を受理して計算を停止したという。TM の受理状態集合 $F = \{q_H\}$ を停止集合ともいう。

TM M において、ある計算状況 $\mathcal{C}_k$ に対して次の計算状況 $\mathcal{C}_{k+1}$

$$\mathcal{C}_k \vdash_M \mathcal{C}_{k+1}$$

が存在しないとき、TM は動作を停止する. 結果として入力は受理されない。

ただし、TM が入力を受理しないときは、かならずしも停止するわけではない。

定義 8.6 TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, q_H)$ が受理認識する言語 $L(M)$ を次で定義する。

$$L(M) = \left\{ x \in \Sigma^* \mid q_0 x \vdash_M^* \alpha q_H \beta \quad q_H \in F, \alpha, \beta \in \Gamma^* \right\}$$

8.3 Turing 機械の例

8.3.1 $\{a^n b^n \mid n \geqq 1\}$ を受理する TM

TM $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, X, Y, B\}, \delta, q_0, B, \{q_4\})$ の状態遷移を次の表 8.1 で定める。

状態	a	b	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	(q_1, a, R)	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	(q_2, a, L)	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

表 8.1　$\{a^n b^n \mid n \geqq 1\}$ を受理する TM の状態遷移表

表 8.1 で定義される TM は図 8.2 で状態遷移図として表すことでもできる。

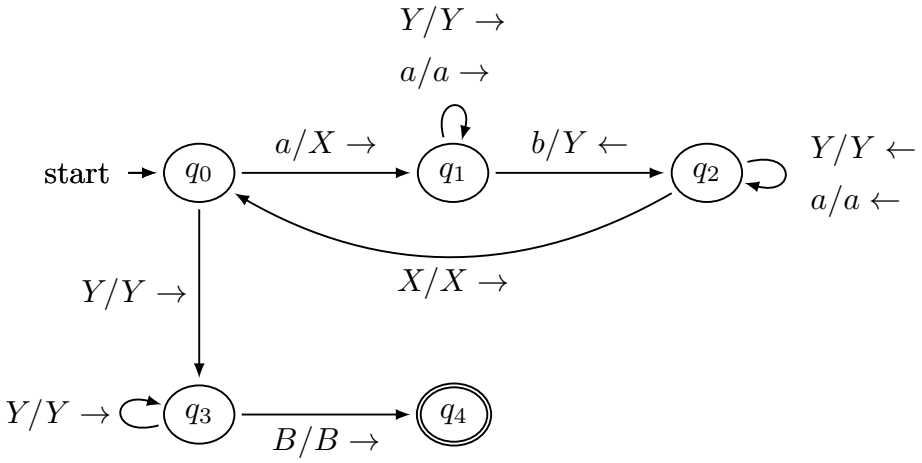


図 8.2　表 8.1 で定義される TM の $\{a^n b^n \mid n \geqq 1\}$ を受理する TM の状態遷移図

実際、入力を $aabb$ とする初期計算状況 q_0aabb からこの TM は次のように計算して受理状態に達して、入力を受理して停止する。

$$q_0aabb \vdash Xq_1abb \vdash Xaq_1bb \vdash Xq_2aYb \vdash q_2XaYb$$
$$\vdash Xq_0aYb \vdash XXq_1Yb \vdash XXYq_1b \vdash XXq_2YY \vdash Xq_2XYY$$
$$\vdash XXq_0YY \vdash XXYq_3Y \vdash XXYYq_3B \vdash XXYYBq_4B \blacktriangleright$$

一方、入力を $aaba$ に対しては、合法的計算状況への推移が存在せず TM は停止してしまう、結として入力は受理しない。

$$q_0aaba \vdash Xq_1aba \vdash Xaq_1ba \vdash Xq_2aYa \vdash q_2XaYa$$
$$\vdash Xq_0aYa \vdash XXq_1Ya \vdash XXYq_1a \vdash XXYaq_1B \triangleright$$

演習 8.1 表 8.1 または図 8.2 で与えられる TM が、言語 $\{a^nb^n \mid n \geqq 1\}$ を受理することを示しなさい。

（ヒント）TM のテープは、入力として $a^nb^{n+k}(k \geqq 0)$ を持つ初期計算状況 $q_0a^nb^{n+k}$ の計算として一般性を失わない。このとき、途中のテープ記号列は正規表現 $\boldsymbol{X^*a^*Y^*b}$ を持つことを示す。a から書き換えられた X の列に続いてまだ書き換えられていない a が続き、その後に b から書き換えられた Y の列とまだ書き換えられていない b の列が続くためである。

M は残っている a の最左の a の左隣に戻ると初期状態 q_0 となり、そこからヘッド下にある a を読んで X に書き換えて状態 q_1 になってヘッドを右移動する。

状態 q_1 ではテープ状の a と Y を読み飛ばしたまま状態を変えずにヘッドを右移動し（途中 X や B を読むと停止）、最左端にある b を読むと Y に書き直して状態 q_2 に遷移して、ヘッドを左移する。状態 q_2 のままで a や Y を読むと読み飛ばしながらヘッドを左に移動し続け、最右端にある X（その右隣りはまだ残っていれば a がある）を読むと状態 q_0 になってヘッドを右に移動する。こうして、1 つのサイクルが終了した結果、最左端の a が X に最左端の b が Y に書き換わる。

次に TM が取る 2 つの場合がある。

(1) まだ残っている a を読んで、いじょうのサイクルを繰り返す。
(2) 少なくとも $X^nY^nb^k(k \geqq 0)$ である。既に a がなく最左端の Y を読んで状態 q_3 になってヘッドは右に移動。$k = 0$ であれば（すべての b が X と同数 n だけの Y に置き換えられていれば）、入力 a^nb^n は受理すべきで、q_3 のままで Y を読み飛ばして、B を読んで q_4 に到達できれば受理状態で停止（入力 a^nb^n を受理）。その途中で（Y^n を読み飛ばした後に）b に出逢えば、次の合理的計算状況が存在せずに計算を停止（入力 $a^nb^{n+k}(k \geqq 1)$ を受理しない）。

8.3.2 $\{wcw \mid w \in \{a,b\}^*\}$ を受理する TM

あとで示すように、$L = \{wcw \mid w \in \{a,b\}^*\}$ は文脈依存文法 G で生成される文脈依存言語である。L を受理する TM M_2 の状態遷移図を図 8.3 に示した。

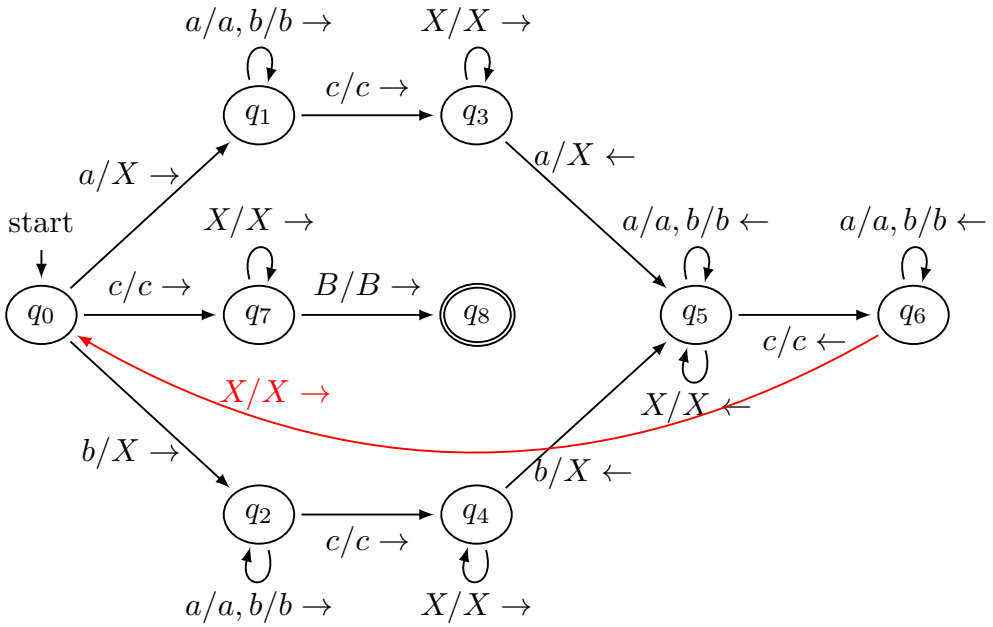


図 8.3　$\{wcw \mid w \in \{a,b\}^*\}$ を受理する TM M_2 の状態遷移図

演習 8.2 先の TM M_2 が言語 $L = \{wcw \mid w \in \{a,b\}^*\}$ を受理することを示しなさい。

（ヒント）$u_i = v_i$ のとき、状態 q_0 から遷移して q_6 に達した後に q_0 に復帰）：

$$
\begin{aligned}
&X^k q_0 u_i u_{i+1} \dots u_n c X^k v_i v_{i+1} \dots v_n \\
&\underset{M}{\vdash} X^k X q_1 u_{i+1} \dots u_n c X^k v_i v_{i+1} \dots v_n \\
&\quad \overset{*}{\underset{M}{\vdash}} X^{k+1} u_{i+1} \dots u_n q_1 c X^k v_i v_{i+1} \dots v_n \\
&\underset{M}{\vdash} X^{k+1} u_{i+1} \dots u_n c q_3 X^k v_i v_{i+1} \dots v_n \\
&\quad \overset{*}{\underset{M}{\vdash}} X^{k+1} u_{i+1} \dots u_n c X^k q_3 v_i v_{i+1} \dots v_n \\
&\underset{M}{\vdash} X^{k+1} u_{i+1} \dots u_n c X^{k-1} q_5 X X v_{i+1} \dots v_n \\
&\quad \overset{*}{\underset{M}{\vdash}} X^{k+1} u_{i+1} \dots u_n q_5 c X^{k+1} v_{i+1} \dots v_n \\
&\underset{M}{\vdash} X^{k+1} u_{i+1} \dots u_{n-1} q_6 u_n c X^{k+1} v_{i+1} \dots v_n \\
&\quad \overset{*}{\underset{M}{\vdash}} X^k q_6 X u_{i+1} \dots u_n c X^{k+1} v_{i+1} \dots v_n \\
&\underset{M}{\vdash} X^{k+1} q_0 u_{i+1} \dots u_n c X^{k+1} v_{i+1} \dots v_n
\end{aligned}
$$

先の $L = \{wcw \,|\, w \in \{a,b\}^*\}$ は文脈依存文法で生成される。文法 $G = (\Sigma_N, \Sigma_T = \{a,b,c\}, P, S)$ として、a と b に対応する非終端記号として $A, A', A_=$ および $B, B', B_=$ を導入して $\Sigma_N = \{S, T, A, B, A', B', A_@, B_@\}$ としたとき、次の生成規則 P を考える。

$x, y \in \{a,b\}$ と対応する $X, Y \in \{A,B\}$ について

$$
{}^{(1)}S \to c \,|\, xTX_@, \quad {}^{(2)}T \to c \,|\, xTX,
$$
$$
{}^{(3)}XY_@ \to Y'X_@, \quad {}^{(4)}XY' \to Y'X,
$$

$x \in \{a,b,c\}$ および $y \in \{a,b\}$ と対応する $\{A,B\}$ に対して

$$
{}^{(5)}xY' \to xy, \quad {}^{(6)}xY_@ \to xy.
$$

このとき $L = L(G)$ となり、L は文脈依存言語であることがわかる。実際、G は次のようにして $L = \{wcw \,|\, w \in \{a,b\}^*\}$ を生成する：

$$
\begin{aligned}
S &\overset{1}{\Rightarrow} a_1 T A_{1@} \overset{2}{\Rightarrow} a_1 a_2 T A_2 A_{1@} \\
&\overset{*2}{\Rightarrow} a_1 a_2 \ldots a_n T A_n A_{n-1} \ldots A_2 A_{1@} \\
&\overset{2}{\Rightarrow} a_1 a_2 \ldots a_n c A_n A_{n-1} \ldots A_2 A_{1@} \\
&\overset{3}{\Rightarrow} a_1 a_2 \ldots a_n c A_n A_{n-1} \ldots A_3 A_2 A_{1@} \\
&\overset{4}{\Rightarrow} a_1 a_2 \ldots a_n c A_n A_{n-1} \ldots A_4 A_3 A'_1 A_{2@} \\
&\overset{4}{\Rightarrow} a_1 a_2 \ldots a_n c A_n A_{n-1} \ldots A_5 A_4 A'_1 A_3 A_{2@} \\
&\overset{*4}{\Rightarrow} a_1 a_2 \ldots a_n c A_n A'_1 A_{n-1} \ldots A_4 A_3 A_{2@} \\
&\overset{4}{\Rightarrow} a_1 a_2 \ldots a_n c A'_1 A_n A_{n-1} \ldots A_4 A_3 A_{2@} \\
&\overset{5}{\Rightarrow} a_1 a_2 \ldots a_n c a_1 A_n A_{n-1} \ldots A_4 A_3 A_{2@} \\
&\overset{3}{\Rightarrow} a_1 a_2 \ldots a_n c a_1 A_n A_{n-1} \ldots A_5 A_4 A'_2 A_{3@} \\
&\overset{*4}{\Rightarrow} a_1 a_2 \ldots a_n c a_1 A'_2 A_n A_{n-1} \ldots A_4 A_{3@} \\
&\overset{5}{\Rightarrow} a_1 a_2 \ldots a_n c a_1 a_2 A_n A_{n-1} \ldots A_5 A_4 A_{3@} \\
&\overset{*}{\Rightarrow} a_1 a_2 \ldots a_n c a_1 a_2 \ldots a_{n-2} a_{n-1} A_{n@} \\
&\overset{6}{\Rightarrow} a_1 a_2 \ldots a_n c a_1 a_2 \ldots a_{n-1} a_n .
\end{aligned}
$$

8.4 TM の停止性

TM M への入力に対して、次の 3 つの可能性がある：

(1) 停止状態 q_H に到達して受理して停止する。
(2) ある計算状況より先の計算状況が存在せずに（M を定める 5 つ組のリストに次の計算状況を定めるものが存在しない）クラッシュして停止する（非受理）。
(3) 永久に状態遷移を繰り返し停止しない。

(1) と (2) のように永久に状態遷移を繰り返すことがない TM を停止性 **TM** という。

定義 8.7 停止性 TM M では入力 $w \in \Sigma^*$ に対して $w \in L(M)$ であれば停止して受理し、$w \notin L(M)$ であれば停止して非受理が判明するために、停止性 TM M は言語 $L(M)$ を決定するという。このような言語を決定可能な言語とも言う。

定義 8.8 言語 L が帰納的に列挙可能または帰納的に加算 (recursively enumerable) とは、ある TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ が存在して M によって受理（認識）される文字

列集合の全体である。

$$L(M) = \left\{ x \in \Sigma^* \mid q_0 x \vdash_M^* \alpha q_H \beta \quad q_H \in F, \alpha, \beta \in \Gamma^* \right\}.$$

言語 L が帰納的 (recursive) とは、停止性 TM M が存在すること、つまり任意の記号列に対して（受理されない時でも）常に停止するような TM M によって受理される文字列集合全体 $L(M)$ である。この定義より、次の包含関係が成立する：

帰納的列挙可能（帰納的加算）言語のクラス $\supset$ 帰納的言語のクラス

定義 8.9 自然数上の述語（真偽値を取る関数）$P(x_1, \ldots, x_n)$ が与えられている時、任意の $(x_1, \ldots, x_n)$ に対して $P(x_1, \ldots, x_n)$ が成立するか否かを決定するアルゴリズムを作る問題を述語 P に関する決定問題 (decision problem) という。可解 (solvable) または決定可能 (decidable) とは、決定のためのアルゴリズムが作れること（関連付けた言語が帰納的である）。

次の 2 つの定理は Turing(1936) によって得られた。

定理 8.1 与えられた TM M と入力 w に対して、M が w に関して停止するかどうかの決定問題は非可解（判定不能）である。

定理 8.2 TM を固定したとき、任意の自然数 x に対して（符号化 $\bar{x}$）、計算状況 $\bar{x} q_q B$ に始まり $w q_H B$ $(w \in \Gamma^*)$ で停止する M による計算が存在するか否かを決定する問題を非可解にする TM M が存在する。

定理 8.3 (Matiyasevich(1970)) 整数係数を持つ与えられた任意の多変数多項式がどのような場合に整数解を持つかを決定する一般解法 (Diophantine 方程式の可解性判定方法を見つける: Hilbert の第 10 問題）は非可解である。

第 9 章

オートマチック列

9.1 k-DFAO

出力付き決定有限オートマトン (**DFAO**:Deterministic Finite Automaton with Output) とは、節 2.4.2 の出力付き有限オートマトン（Moore 機械）$M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ である。有限状態集合 Q、入力アルファベット Σ、状態遷移関数 $\delta : Q \times \Sigma \to Q$ と初期状態 $q_0 \in Q$ に加えて、出力アルファベット Δ を値とする状態出力関数 $\tau : Q \times \Sigma \to \Delta$ を持つ機械である。初期状態 $q_0 \in Q$ について、入力記号列 $w \in \Sigma^*$ の（同じ記号を使った拡張された出力関数の) 出力 $\tau(w)$ は

$$\tau(\varepsilon) = \tau(\delta(q_0, \varepsilon)) = \tau(q_0)$$
$$\tau(w) = \tau(\delta(q_0, w))$$

で与えられる。このとき DFAO M は関数 $f_M : \Sigma^* \to \Delta$ を

$$f_M(w) = \tau(\delta(q_0, w)), \quad w \in \Sigma^*$$

を定める。関数値がこうして計算できる関数 $f : \Sigma^* \to \Delta$ を有限状態関数 (finite state function) という。

DFAO の状態遷移図（図 2.18）を再び図 9.1 に示す。

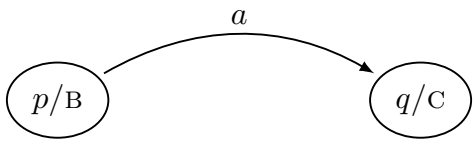


図 9.1　DFAO(出力付き決定有限オートマトンの状態遷移図。出力 B を持つ状態 p ($\tau(p) = \mathrm{B}$) が入力 $a \in \Sigma$ によって出力 C を持つ状態 q ($\tau(p) = \mathrm{C}$) へ遷移する ($p, q \in Q$, $\mathrm{B}, \mathrm{C} \in \Delta$)。

例 9.1 図 9.2 は出力アルファベット $\Delta = \{0, 1\}$ を持つ DFAO の状態遷移を表している。入力列 $w \in \Sigma^* = \{0, 1\}^*$ を読み込みながら状態に応じて値 0 または 1 を出力する。このとき、入力記号 1 に出会うたびに状態を変える。

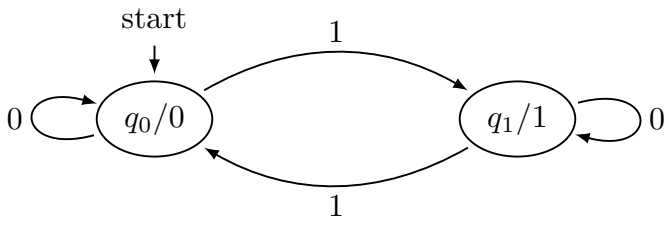


図 9.2　出力アルファベット $\Delta = \{0, 1\}$ を持つ DFAO. 入力列 $w \in \Sigma^* = \{0, 1\}^*$ を読み込みながら値 0 または 1 を出力する。このとき、入力記号 1 に出会うたびに状態を変える。

定理 9.1 $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ が DFAO であるとする。このとき、すべての $d \in \Delta$ に対して集合

$$L_d(M) = \left\{ w \in \Sigma^* \,|\, \tau(\delta(q_0, w)) = d \right\}$$

は正規言語である。

証明

■

定理 9.1 の逆、DFAO は異なる正規言語に対する DFA を糊付けすることで構成できること、も次のように成立する。

定理 9.2 Σ, Δ を有限アルファベットとする。$r \geqq 1$ を整数として、r 個の正規言語 $L_1, L_2, \ldots, L_r$ が集合 Σ^* を分割するとする。

$$\bigcup_{1 \leqq k \leqq r} L_k = \Sigma^*, \quad L_i \cap L_j = \phi (i \neq j).$$

いま、$\Delta = \{a_1, a_2, \ldots, a_r\}$ とする。

このとき、DFAO $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ が存在して、$w \in L_k$ であるときに限り $\tau(\delta(q_0)) = a_k$ となる。

証明

■

さて、DFAO において、ここでは特に入力列が数の基底 k-表示で表される場合を考えよう。すなわち、$\Sigma = \Sigma_k = \{0, 1, \ldots, k-1\}$ とした次のような入力列を考えた DFAO を k-DFAO と呼ぶ。

定義 9.1 整数 n の基底 k-表現を ***k*-DFAO** の入力列 $w = w_0 w_1 \ldots w_{j-1}$ とするとは、$[w]_k = n$、すなわち、k-DFAO に n の k-表現 $[w_0 w_1 \ldots w_{j-1}]_k$ の最左端記号から入力して、$\delta(q_0, w_1) = p_1$, $\delta(p_1, w_2) = p_2, \ldots, \delta(p_{j-2}, w_{j-1}) = p_{j-1}$ のように状態遷移が進行して、入力列 w に対する出力が $\tau(\delta(q_0, w)) = \tau(p_{j-1})$ となることである。

図 9.2 で表される状態遷移図を持つ DFAO は 2-DFAO でもある。

9.2 オートマチック列

定義 9.2 列 $(a_n)_{n \geqq 0}$ が有限アルファベット Δ 上の ***k*-オートマチック列**であるとは、k-DFAO $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ が存在して、

$$a_n = \tau(\delta(q_0, w)), \quad \text{すべての } n \geqq 0、[w]_k = n$$

となることである。ここではとりあえず、M は入力が 0 で始まる (leading zero) $[0 \cdots 0w]_k = n$ でも正しい値を返すとする。

与えられた列 $(a_n)_{n \geqq 0}$ がオートマチック列であるかは、例えば a_n を $[w]_k = n$ であるような入力列に対する出力であるような k-DFAO M を構成してみせることによって示すことができる。他にもさまざまな手法が探求されており、オートマチック列は現在盛んに研究されている分野である [19]。

9.2.1 Thue-Morse 列

定義 9.3 列 $\boldsymbol{t} = (t_n)_{n\geqq 0}$ が **Thue-Morse** 列であるとは

$$t_n = \begin{cases} 0 & n \text{ の 2-表示で 1 の数が偶数} \\ 1 & n \text{ の 2-表示で 1 の数が奇数} \end{cases}$$

である 0 と 1 からなる列

$$\boldsymbol{t} = t_0 t_1 t_2 \ldots = 01101001\ldots$$

である。

Thue-Morse 列 $\boldsymbol{t} = (t_n)_{n\geqq 0}$ が 2-オートマチック列であることは、$\boldsymbol{t}$ を出力する 2-DFAO を示してみればよい。実際、状態遷移図として図 9.2 で表される 2-DFAO が Thue-Morse 列を出力することは容易に確認できる。

演習 9.1 アルファベット A 上の記号列 $vw (v, w \in A^\dagger)$ に対する射 (morphism)σ は

$$\sigma(vw) = \sigma(v)\sigma(w)$$

で定義される。たとえば、$a \in A$ に対して $\sigma^n(a) = \sigma(\sigma^{n-1}(a)) = \underbrace{\sigma \circ \cdots \circ \sigma}_{n\text{-times}}(a)$ と記す。

$\{0,1\}$ 上の **Thue-Morse** 代入 (morphism または substitution) を

$$t(0) = 01, \quad t(1) = 10$$

とする。$t^2(0) = t(01) = 0110$, $t^3(0) = 01101001$ である。

$\lim_{n\to\infty} t^n(0)$ は Thue-Morse 列となることを確かめなさい。

9.2.2 Rudin-Shapiro 列

$\Sigma_k = \{0, 1, \ldots, k-1\}$ 上の文字列を $P \in \Sigma_k^*$ としたとき、

$$e_{k:P}(n) = n \text{ の基底 } k\text{-表示において、重複を含んで出現するパターン } P \text{ の個数}$$

とする。列 $(e_{k:P}(n))_{n\geqq 0}$ をパターン列ということがある。例えば、$e_{2:11}(7) = e_{2:11}(27) = 2$. ただし、パターン P は少なくとも 1 つ非ゼロを含むとき、n の k-表現が任意長の 0 列で始まるとする：$e_{2:010}(5) = 1$. また、P がある i について 0 パターン 0^i のときには、n の標準的な基底 k-表現で P の数を数えるとする：$e_{2:00}(36) = 2$.

定義 9.4 Rudin-Shapiro 列 $\boldsymbol{r} = (r_n)_{n\geqq 0}$ を次で定義する：

$$r_n = (-1)^{e_{2:11}(n)}.$$

ここで、$e_{2:11}(n)$ は n の 2-進数表示において、重複を含めた 11 の登場回数である。

表 9.1 にあるように

$$\boldsymbol{r} = r_0 r_1 r_2 \cdots = 1.1.1. - 1.1.1. - 1.1.1.1.1. - 1. - 1. - 1.1. - 1.1\ldots$$

である（わかりやすさのために数字間にドット (.) を挟んだ）。

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	…
$e_{2:11}(n)$	0	0	0	1	0	0	1	2	0	0	0	1	1	1	2	3	0	…
r_n	1	1	1	-1	1	1	-1	1	1	1	1	-1	-1	-1	1	-1	1	…
$r_n r_{n-1}$		1	1	-1	-1	1	-1	-1	1	1	1	-1	1	1	-1	-1	-1	…
$(-1)^n$	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	…
d_n		R	L	L	R	R	R	L	L	R	L	L	L	R	R	L	L	…

表 9.1　Rudin-Shapiro 列 $(r_n)_{n\geqq 0}$ とそれから定まる折りたたみ列 $(d_n)_{n\geqq 1}$.

Rudin-Shapiro 列を出力する 2-DFAO の状態遷移図を図 9.3 に示した。この 2-DFAO の状態遷移図において、ab/c とは、0 と 1 からなる入力列におけるパターン'11' が登場する回数の途中和（mod）を a、b を入力列の最後のディジット (0 または 1) を使って表される状態を ab とし、その状態出力が c としている。

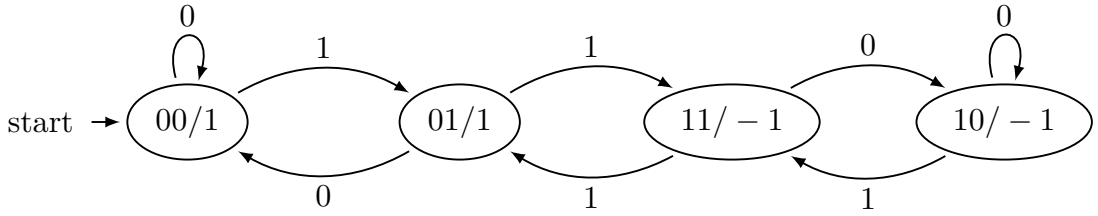


図 9.3　2-オートマチックな Rudin-Shapiro 列 $(r_n)_{n\geqq 0}$ を出力する 2-DFAO.

Rudin-Shapiro 列を平面 $\mathbb{R}^2$ 上の格子点訪問と関係づけると興味深い性質があることがわかる。原点 $(0,0)$ から出発する連続直線を考え、y-軸方向に 1 だけ進んで $(0,1)$ を訪れ

る（向きはまだ y-軸正方向を向いている）。以下、各 $n \geqq 1$ で曲線の進む向き d_n が次のように左 (L) または右 (R) に $\pi/2$ だけ転回するとし、距離 1 だけ連続曲線を延長していく：

$$d_n = \begin{cases} L & r_n r_{n-1} = (-1)^n \text{ であるとき,} \\ R & r_n r_{n-1} = -(-1)^n \text{ であるとき.} \end{cases}$$

こうして L と R からなる記号列 $(d_n)_{n\geqq 1}$ が得られる（表 9.1 参照）。

図 9.4 は、$n = 3500$ までの Rudon-Shapiro 列から定まる記号列 $(d_n)_{1\leqq n\leqq 3500}$ に沿って格子点を辿らせた連続曲線の様子である。詳しく観察してみると、原点から出発した曲線は格子点で接することはあっても、自身の曲線に交差することがない自己回避曲線 (self-avoiding curve) となっていることがわかる。

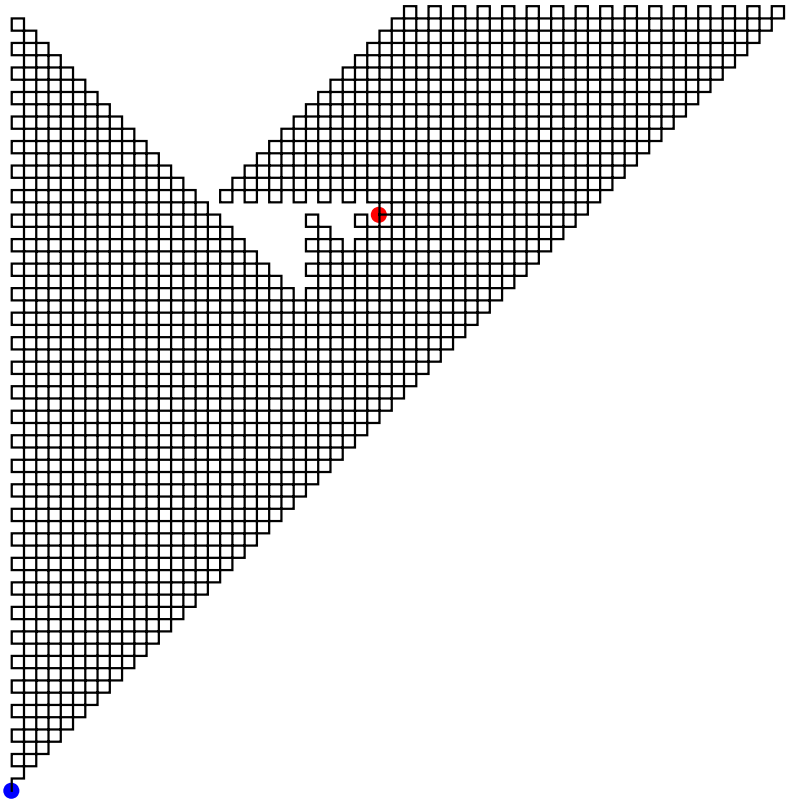


図 9.4　2-オートマチックな Rudin-Shapiro 列 $(r_n)_{n\geqq 0}$ から定めた転回記号列 $(d_n)_{1\leqq n\leqq 3500}(d_n \in \{L, R\})$ が描く連続曲線. 平面 $\mathbb{R}^2$ の 1/8 を埋め尽くしていく。

9.2.3　紙折り列

紙を 2 つ折りにする（紙の長さは $\frac{1}{2}$ になる）操作を n 回繰り返してて（紙の長さは $\frac{1}{2^n}$ になる)、展開してみると紙の左端から山（u）と谷（d）からなる折り目がならんでいることが観察できる。このとき得られる折り目 $\{u, d\}$ からなる紙折り列 (paper folding sequence)F_n を考える。

観察: 紙折り記号列 F_n は次のように決定される:

$$F_1 = u,$$
$$F_n = F_{n-1}u\overline{F_{n-1}^R}, \quad n \geqq 2.$$

ここで、F_n^R は記号列 F_n の反転（記号列を逆順)、記号列 $\overline{w}$ は $w \in \{u, d\}$ の記号 u と d を交換した記号列を表す。たとえば次のようになる。

$$F_2 = F_1u\overline{F_1} = uud,$$
$$F_3 = F_2u\overline{F_2} = uuduudd,$$
$$F_4 = F_3u\overline{F_3} = uuduuddduuddudd, \ldots$$

以下、$u \to 1$, $d \to -1$ と折れ目記号を読み替えることにする。

こうして定まる紙折り記号列 $\boldsymbol{f} = (f_n)_{n\geqq 1}$ は 2-オートマチック列である。実際、図 9.5 で表される 4 状態の状態遷移図で定義される 2-DFAO が紙折り記号列を出力する。

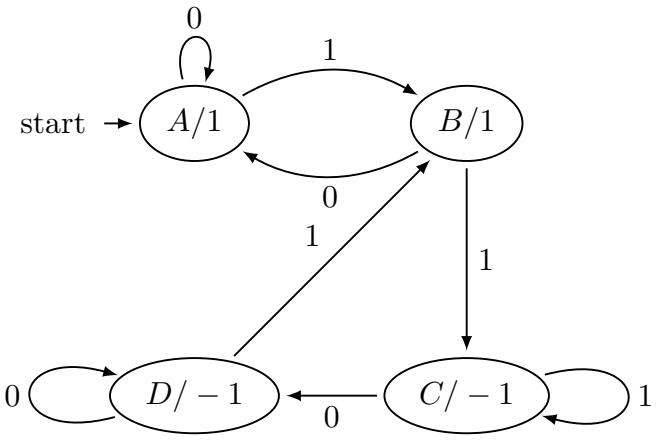


図 9.5　2-オートマチックな紙折り列 $(f_n)_{n\geqq 1}(f_n \in \{-1, 1\})$ を出力する 2-DFAO.

演習 9.2 2-オートマチック列である紙折り列 $L_F = \lim_{n\to\infty} F_n$ を言語 $L = \{wuw^R \,|\, w \in \{u,d\}^*\}$ と比較しなさい（節 7.2 参照）。

第 10 章

帰納関数と計算可能性

10.1　原始帰納関数

定義 10.1 自然数を $\mathbb{N} = \{0, 1, 2, \dots\}$ とする。n 個の自然数の組に対して（高々）1 この自然数を対応づける関数 $f : \mathbb{N}^n \to \mathbb{N}$ を数論的関数と呼ぶ。

(1) 零関数 $z(x) = 0, \qquad$ すべての $x \in \mathbb{N}$.
(2) 後者関数 $s(x) = x + 1$.
(3) 射影関数 $p_k(x_1, \dots, x_k, \dots, x_n) = x_k$.

を初期関数 (basic functions) という。

初期関数からより複雑な関数を組み立てる操作をとして次の 2 つを考える。

(I) 合成 (composition): r 変数の関数 h と r 個の n 変数関数 $g_i (1 \geqq i \geqq r)$ から n 変数の関数 f を次で定義：

$$f(x_1, \dots, x_n) \equiv h(g_1(x_1, \dots, x_n), \dots, g_r(x_1, \dots, x_n))$$

(II) 原始帰納 (primitive recursive): n 変数関数 g と $(n+2)$ 変数関数 h から $(n+1)$ 変数関数を次で定義:

$$f(x_1, \dots, x_n, 0) \equiv g(x_1, \dots, x_n)$$
$$f(x_1, \dots, x_n, y+1) \equiv h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, yA))$$

定義 10.2 初期関数 (1),(2),(3) に操作 (I),(II) を有限回適用して得られる関数を原始帰納関数 (promitive recursive function) という。

演習 **10.1** 計算機で使う多くの関数は原始帰納的である。

(1) 定数関数 $C_k(x_1, \ldots, x_n) = k$.
(2) 和 $\mathrm{Plus}(x, y) = x + y$.
(3) 積 $\mathrm{times}(x, y) = x \times y$.
(4) ベキ乗 x^y.
(5) 階乗 $x!$.
(6) 自然数上の減算 $x \dot{-} y$.
(7) 差の絶対値 $|x \dot{-} y|$.
(8) x の符号

$$\mathrm{sg}(x) = \begin{cases} 0, & x = 0 \\ 1, & x > 0. \end{cases}$$

命題 **10.1** 関数 $f(x_1, \ldots, x_n, y)$ が原始帰納的であれば、有限和

$$\sum_{y<z} f(x_1, \ldots, x_n, y)$$

や有限積

$$\prod_{y<z} f(x_1, \ldots, x_n, y)$$

も原始帰納的である。ただし、

$$\sum_{y<0} f(x_1, \ldots, x_n, y) = 0, \quad \prod_{y<0} f(x_1, \ldots, x_n, y) = 0.$$

定義 **10.3** 真偽値 $\{True, False\}$ を取る述語 (predicate)$P((x_1, \ldots, x_n)$ が原始帰納的であるとは、関数

$$C_P((x_1, \ldots, x_n) = \begin{cases} 0, & P((x_1, \ldots, x_n) \\ 1 & \neg P((x_1, \ldots, x_n) \end{cases}$$

が原始帰納であるときをいう。関数 C_P を述語 P の特徴関数という。

命題 **10.2** 述語 $P(x_1, \ldots, x_n, y)$ を原始帰納的と仮定する。このとき、次で定義される 2 つの述語は原始帰納的である：

$$(\exists y)_{<z} P(x_1, \ldots, x_n, y) \equiv P(x_1, \ldots, x_n, 0) \vee \cdots \vee P(x_1, \ldots, x_n, z-1)$$
$$(\forall y)_{<z} P(x_1, \ldots, x_n, y) \equiv P(x_1, \ldots, x_n, 0) \wedge \cdots \wedge P(x_1, \ldots, x_n, z-1).$$

定義 10.4 述語 $P(x_1,\ldots,x_n)$ に対して、有界 $\boldsymbol{\mu}$ 作用素 $(\mu y)_{<z}$ を次で定義する。

$$(\mu y)_{<z}P(x_1,\ldots,x_n,y)=\begin{cases}\min\{y\,|\,y<z\wedge P(x_1,\ldots,x_n,y)\}, & (\exists y)_{<z}P(x_1,\ldots,x_n,y)\\ z, & \neg(\exists y)_{<z}P(x_1,\ldots,x_n,y)\end{cases}$$

命題 10.3 述語 $P(x_1,\ldots,x_n,y)$ が原始帰納的であれば、関数 $(\mu y)_{<z}P(x_1,\ldots,x_n,y)$ は原始帰納的である。

10.2 帰納的関数

実際に計算できる関数の族は、原始帰納関数よりも拡大された帰納関数と一致する。

定義 10.5 述語 $P(x_1,\ldots,x_n,y)$ が正則であるとは、任意の $(x_1,\ldots,x_n)$ に対して $P(x_1,\ldots,x_n,y)$ を True とする y が存在することである。関数 $g(x_1,\ldots,x_n,y)$ が正則とは、任意の $(x_1,\ldots,x_n)$ に対して $g(x_1,\ldots,x_n,y)=0$ となる y が存在することである。

定義 10.6 述語 $P(x_1,\ldots,x_n,y)$ に対して、$\boldsymbol{\mu}$ 作用素 (μy) を次で定義する。

$$(\mu y)P(x_1,\ldots,x_n,y)=\begin{cases}\min\{y\,|\,y<z\wedge P(x_1,\ldots,x_n,y)\}, & (\exists y)P(x_1,\ldots,x_n,y)\\ \text{無定義}, & \neg(\exists y)P(x_1,\ldots,x_n,y).\end{cases}$$

μy は $n+1$ 変数の述語から n 変数の関数を作る。

定義より、$(x_1,\ldots,x_n)$ に対して $(\mu y)P(x_1,\ldots,x_n,y)$ が無定義となることがあり得るために、μ 作用素でで定義される関数は全域的でない場合がある。

新しい関数を定義する第 3 の操作を次で導入する。

(III) 関数 $g(x_1,\ldots,x_n,y)$ から関数 $f(x_1,\ldots,x_n)$ を

$$f(x_1,\ldots,x_n)=\mu y g(x_1,\ldots,x_n,y)$$

によって組み立てる。

g が全域的でも f が全域的であるとは限らないため、f の全域性を保証する操作として

(III′) 正則関数 $g(x_1,\ldots,x_n,y)$ から関数 $f(x_1,\ldots,x_n)$ を

$$f(x_1,\ldots,x_n)=\mu y\big(g(x_1,\ldots,x_n,y)=0\big)$$

によって組み立てる。

定義 **10.7** 初期関数 (1),(2),(3) に操作 (Ⅰ), (Ⅱ), (Ⅲ) を有限回適用して定義される関数を部分帰納的 (partial recursive) という。初期関数 (1),(2),(3) に操作 (Ⅰ), (Ⅱ), (Ⅲ′) を有限回適用して定義される関数を帰納的 (recursive) という。

10.3 原始帰納関数でない関数の例

すべての原始帰納的関数は、それがどう定義されるかを示す有限文字列で記述することができる。そのような記述は符号化して順序付けられ、それゆえすべての原始帰納関数の集合は加算である。

次の定理は、$\mathbb{N}$ 上の関数 $F : \mathbb{N} \to \mathbb{N}$ の集合には原始帰納的でない関数の存在をいう。

定理 **10.1** 任意の 1 変数の原始帰納関数 $f(x)$ に対して、

$$f(x) = F(n, x)$$

となる自然数 n が存在するような関数 $F(n, x)$ は原始帰納的でない。

証明　$\mathbb{N}$ 上のすべての関数が原始帰納的であると仮定する。それらを $F(1, x), F(2, x), \ldots, F(n, x), \ldots$ と順序付け並べることができる。関数 $F(n, x)$ は原始帰納的である。

したがって、これに 1 を加算する関数を $f(x)$

$$f(x) = F(x, x) + 1$$

と定義した時、$f(x)$ は原始帰納的となる。定理は

$$f(x) = F(^{\exists}n, x)$$

となる自然数 n が存在することを主張する。そこで $n = x$ のときを考えてみると

$$f(n) = F(n, n)$$

であるが、定義より $f(x)$ は

$$f(n) = F(n, n) + 1$$

であった。これより、$F(n, n) = F(n, n) + 1$ となり矛盾。故に、$F(n, x)$ は原始帰納的ではない。 ■

注意 **10.1** ただし、$F(n,x)$ は実際に計算できる（そのための手続きが与えられた）関数、すなわち以下を満たす帰納関数 $F(x,y)$ として選ぶことができる。

(1) 任意に x を固定すると $F(x_{\mathrm{fix}},y)$ は原始帰納的。
(2) 任意に 1 変数原始帰納関数 $f(x)$ に対して

$$f(x)=F(n,x)$$

となるような自然数 n が存在。

10.3.1 Ackermann 関数

原始帰納関数でない計算可能な関数（帰納関数）として **Ackermann** 関数が知られている。

定義 **10.8** Ackermann 関数 $A:\mathbb{N}\times\mathbb{N}\to\mathbb{N}$ を次の手続きで定義する：

$$\begin{aligned}&A(0,y)=y+1\\&A(x+1,0)=A(x,1)\\&A(x+1,y+1)=A(x,A(x+1,y))\end{aligned}$$

Ackermann 関数 $A(x,y)$ の第一変数 x を定数 j に固定した 1 変数関数を $A_j(y)$

$$A_j(y)\equiv A(j,y)$$

とする。

補題 **10.1** $A_j:\mathbb{N}\to\mathbb{N}$ は原始帰納関数である。

証明　帰納法に依る。$j=0$ のとき、$A_0(y)=y+1$ より明らか。$A_{j+1}(y)$ は、定数 $A_j(1)$ と関数 A_j を使う原始帰納法

$$\begin{cases}A_{j+1}(0)=A_j(1)\\A_{j+1}(y+1)=A_j(A_{j+1}(y))\end{cases}$$

より A_{j+1} が定義されていることから、A_j が帰納関数であれば A_{j+1} もまた帰納的である。また Ackermann 関数 $A(x,y)$ は $\mathbb{N}\times\mathbb{N}$ で定義される全域関数であることもわかる（$A_j(y)$ も全域関数で常に値を持つ）。 ■

このとき、次の補題が成立する（[12, 補題 4.13]、[13, 補題 5.6.2] ）。

補題 10.2 Ackermann 関数 $A(x,y)$ について、x を固定した関数 $A_j(y) = A_j(y)$ は次の性質を持つ。

(1) $y+2 \leqq A_{j+1}(y)$.
(2) $y+j < A_j(y)$.
(3) $A_j(y) < A_j(y+1) < A_{j+1}(y)$.
(4) 任意の k, ℓ に対して

$$A_k(A_\ell(y)) \leqq A_i(y)$$

となる i が存在する。

定理 10.2 任意の原始帰納関数 $f(x_1, \ldots, x_n)$ に対して

$$f(x_1, \ldots, x_n) \leqq A_c(\max(x_1, \ldots, x_n))$$

となる c が存在する。

定理 10.3 Ackermann 関数 $A(x,y)$ は原始帰納的でない。

証明　$A(x,y)$ を原始帰納的と仮定する。このとき、1 変数関数 $A_x(x)$ は補題 10.1 より原始帰納的である。定理 10.2 から、原始帰納的関数 $A_x(x)$ に対して

$$A_x(x) \leqq A(c, \max(x)) = A(c,x)$$

となる定数 c が存在する。このとき、$x = c+1$ とおくと

$$A_{c+1}(c+1) \leqq A_c(c+1)$$

となる。しかし、補題 10.2(3)$A_j(y) < A_{j+1}(y)$ より $A_c(c+1) \leqq A_{c+1}(c+1)$ でなければならず、矛盾する。 ■

10.3.2　Ackermann 関数の計算

Ackermann 関数値 $A_1(y) = A(1,y), A_2(y) = A(2,y), A_3(y) = A(3,y)$ を計算してみよう。

$$
\begin{aligned}
A(1,y) &= A(0,A(1,y+1)) = A(1,y-1)+1\\
&= A(0,A(1,y-2))+1 = A(1,y-2)+2\\
&\cdots\\
&= A(1,0)+y = y+2.\\
A(2,y) &= A(1,A(2,y-1)) = A(2,y-1)+2\\
&= A(1,A((2,y-2))+2 = A(2,y-2)+4\\
&\cdots\\
&= A(2,0)+2y = A(1,1)+2y\\
&= 2y+3.\\
A(3,y) &= A(2,A(3,y-1)) = 2A(3,y-1)+3\\
&= 2A((2,A(3,y-2))+3 = 2\big(2A((3,y-2)+3)\big)+3\\
&= 2^2\big(2A(3,y-3)+3\big)+3(2+1) = 2^3A(3,y-3)+3(2^2+2+1)\\
&\cdots\\
&= 2^yA(3,0)+3(2^{y-1}+\cdots+2^2+2^1+1) = 2^yA(3,0)+3(2^y-1)\\
&= 8\cdot 2^y-3 = 2^{y+3}-3.
\end{aligned}
$$

また、$A(3,y)$ を認めると次のことが確かめられる。

$$
\begin{aligned}
A(4,y) &= A(3,A(4,y-1)) = 2^{3+A(4,y-1)}-3\\
&= 2^{3+\left(2^{3+A(4,y-2)}\right)-3}-3 = 2^{2^{3+A(4,y-2)}}-3\\
&= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y}{}^{3+A(4,0)} \qquad -3\\
&= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y}{}^{2^{2^4}} -3 = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y+3}{}^{2^2} -3.
\end{aligned}
$$

ここで、$A(4,0)=A(3,1)=2^4-3$ を使った。

10.3.3 Knuth の冪塔記法

Knuth(1976) は乗算のベキ乗を拡張した冪塔演算 (べきタワー: power tower) に関する自然な記法↑を考えた。これを更に拡張した Conway(1995) の鎖矢記法 (chained arrow) がある。

↑演算を

$$a \uparrow b \equiv \underbrace{a \times a \times \cdots \times a}_{a \text{ に関する } b \text{ 回の乗算}}$$

と定義する。このとき、通常のベキ乗の式は

$$a^b = a \uparrow b$$

で表される。

また、2 重タワー ↑↑ 演算 (tetration) を

$$a \uparrow\uparrow b \equiv \underbrace{a \uparrow a \uparrow \ldots \uparrow a}_{a \text{ に関する } b \text{ 回の } \uparrow \text{ 演算}}$$

と定義する。このとき、ベキ指数が連なる式は

$$\underbrace{a^{a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}}}_{b \text{ 回}} = a \uparrow\uparrow b$$

と表される。

3 重タワー演算 ↑↑↑ 演算 (pentation) を

$$a \uparrow\uparrow\uparrow b \equiv \underbrace{a \uparrow\uparrow a \uparrow\uparrow \ldots \uparrow\uparrow a}_{a \text{ に関する } b \text{ 回の } \uparrow\uparrow \text{ 演算}}$$

で定義する。具体的には、次のようになる

$$a \uparrow^3 b = \underbrace{a \uparrow^2 \left(a \uparrow^2 \left(\ldots \uparrow^2 (a \uparrow^2 a)\right)\right)}_{a \text{ が b 回登場}}$$

一般的に記法

$$\uparrow^k = \underbrace{\uparrow \ldots \uparrow}_{k \text{ 回}}$$

を導入して、n-重 $\uparrow^n$ 演算を

$$a \uparrow^n b = \underbrace{a \uparrow^{n-1} a \uparrow^{n-1} \ldots \uparrow^{n-1} a}_{a \text{ に関する } b \text{ 回の } \uparrow^{n-1} \text{ 演算}}$$

と定義する。あるいは次のようにも定義できる。

$$a \uparrow^n b = \begin{cases} 1, & n = 0, \\ a^b, & n = 1, \\ a \uparrow^{n-1} (a \uparrow^n (b-1)), & n \geqq 2. \end{cases}$$

冪塔演算は結合則を満たさい。計算順序を右から計算すると決める。計算の順序を変えると値が変わってしまう。

$$2 \uparrow^2 (2 \uparrow^2 2) = 2 \uparrow^4 = 65536$$

$$(2 \uparrow^2 2) \uparrow^2 2 = 4 \uparrow^2 2 = 256$$

先に、具体的に計算した $A(4, y)$ は Tower 記法を使うと

$$A(4, y) = 2 \uparrow\uparrow (y + 3) - 3$$

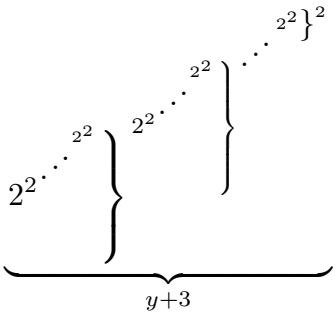
と表される。このように Ackermann 関数は指数関数以上の急増加関数である。

演習 **10.2** 次の数を計算しなさい。

- $2 \uparrow\uparrow 2$
- $2 \uparrow\uparrow 3$
- $2 \uparrow\uparrow 4$
- $2 \uparrow\uparrow 5$
- $2 \uparrow\uparrow\uparrow 1$
- $2 \uparrow\uparrow\uparrow 2$
- $2 \uparrow\uparrow\uparrow 3$

補題 **10.3** Ackermann 関数 $A(x, y)$ において x を固定した $A_x(y)$ は $A_0(y) = y + 1$ に続いて、次のような原始帰納関数である。

(1) $A_1(y) = y + 2.$
(2) $A_2(y) = 2y + 3.$
(3) $A_3(y) = 2^{y+3} - 3 = 2 \uparrow (y + 3) - 3$
(4) $A_4(y) == 2 \uparrow\uparrow (y + 3) - 3 = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^2}}}}}}_{y+3} - 3$

(5) $A_5(y) = 2 \uparrow\uparrow\uparrow (y+3) - 3 =$ 

(6) 一般に

$$A_n(y) = 2 \uparrow^{n-2} (y+3) - 3, \qquad n \geqq 3.$$

10.4 Post の対応問題

定義 10.9 リスト α および β に対する Post の対応問題 (post Corresponding Problem) とは、1 文字以上からなるアルファベット Σ 上の文字列を $k \geqq 1$ 個並べた 2 つのリスト α, β

$$\alpha = [x_1, x_2, \ldots, x_k], \qquad x_i \in \Sigma^\dagger$$
$$\beta = [y_1, y_2, \ldots, y_k] \qquad y_i \in \Sigma^\dagger$$

ならなる（リストの要素は 1 から k まで順序付けられ、i 番目の要素は x_i または y_i である)。リストの組 (α, β) を PCP の実例（インスタンス）と呼ぶ。

PCP の実例 $P(\alpha, \beta)$ が解を持つとは、それぞれのリストから重複を許して $i_1, i_2, \ldots, i_n$ 番目の文字列を取り出して並べて得られる文字列が一致

$$A = x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n} = B$$

すること、つまり、リストから取り出す要素場所の数列 $i_1, i_2, \ldots, i_n (n \geqq 1)$ が存在することである。

数列 $i_1, i_2, \ldots, i_n$ に対して、番号 i とそれに対応付けられた文字列 x_i と y_i をカードの上と下に $\left[\dfrac{x_i}{y_i}\right]$ と書いて次のように同一視する。

$$i_1 i_2 \ldots i_n \Leftrightarrow \left[\frac{x_{i_1}}{y_{i_1}}\right]_{i_1}, \left[\frac{x_{i_2}}{y_{i_2}}\right]_{i_2}, \cdots, \left[\frac{x_{i_n}}{y_{i_n}}\right]_{i_n} \Leftrightarrow \frac{x_{i_1} x_{i_2} \ldots x_{i_n}}{y_{i_1} y_{i_2} \ldots y_{i_n}}$$

選びだした数列 $i_1, i_2, \ldots, i_n$ が実例 $P(\alpha, \beta)$ の解であるとき、上の右端にある上下の文字列 $x_{i_1} x_{i_2} \ldots x_{i_n}$ と $y_{i_1} y_{i_2} \ldots y_{i_n}$ は等しい。

例 **10.1** 実例 $P(\alpha, \beta)$ は必ずしも解を持つとは限らない。$\Sigma = \{a, b\}$ 上の Post の対応問題について、次の実例を検討しよう。

$$\left(\left[\frac{ba}{bab} \right]_1, \left[\frac{abb}{bb} \right]_2, \left[\frac{bab}{abb} \right]_3 \right).$$

インスタンスが解 $i_1, i_2, \ldots, i_n$ を持つと仮定するとき、まず i_1 が決まるかを考える。$i_1 = 2$ とすると文字列 A は abb で B は bb で始まり、$i_1 = 3$ とすると文字列 A は bab で B は abb で始まために、A と B の一致はありえず $i_1 \neq 2, 3$。一方、$i_1 = 1$ とすれば文字列 x は ba で y は bab で始まり、i_2 以降を上手く続けて x と y を一致させる可能性が残るため、$i_1 = 1$ である。

$$\left[\frac{A_1}{B_1} \right] = \left[\frac{ba}{ba\,\overline{b}} \right]_1$$

次に、$i_2 = 1$ と選ぶと $A = ba \cdot ba$、$B = bab \cdot bab$、$i_2 = 2$ と選ぶと $A = ba \cdot abb$、$B = bab \cdot bb$ となり A, B は一致し得ず $i_2 \neq 1, 2$ である。一方、$i_2 = 3$ と選ぶと $A = ba \cdot bab$、$B = bab \cdot abb$ となり

$$\left[\frac{A_{1,3}}{B_{1,3}} \right] = \left[\frac{babab}{babab\,\overline{b}} \right]_{1,3}$$

を得る。ここで、$B_1 = A_1 b$、$B_{1,3} = A_{1,3} b$ に注意する。つまり、この過程は解を持とうとしたとき、これ以降も $i_3 = i_4 \ldots i_m = 3$ と選ぶ必要があることを意味する。したがって、同じ状況

$$B\underbrace{_{1,3,\ldots,3}}_{m\text{ 個}} = A\underbrace{_{1,3,\ldots,3}}_{m\text{ 個}} b$$

が再現され A は B の真の接頭語であり続け、A の文字列の長さは B の長さに追いつくことはない。これより、結果 $A = B$ と文字列が一致することはできない。

この PCP の実例は解を持たない。

定理 10.4 (Post(1946)) Post の対応問題は非可解である。

系 10.1 任意に与えられた CFG G_1 と G_2 に対して

$$L(G_1) \cap L(G_2) = \phi$$

であるか否かを決定する問題は非可解である。

参考文献

[1] ホップクロフト・モトワニ・ウルマン: 『オートマトン 言語理論 計算論 I/II 第 2 版』サイエンス社 (2003). ホップクロフト・ウルマンによる同名の初版（1984 年）は今なお評判が高い。

[2] サローマ: 『計算論とオートマトン理論』サイエンス社 (1988).

[3] M.Sipser: 『計算論の基礎 1, 2, 3 第 2 版』共立出版（2008）.

[4] Jeffrey Shallit:“*A Second Course in Formal Languages and Automata Theory*”, Cambridge University Press (2009). 執筆動機が水準を保っていた [1] の初版の入手が難しくなったと前書きにある。

[5] Arto Salomaa, *Formal Languages*, Academic Press(1973).

[6] Willem J.M. Levelt, *An Introduction to the Theory of Formal Languages and Automata*,John Benjamins Publishing Company(2008).

[7] Peter Linz, *An Itroduction to formal language and automata*, Jones & Bartlett Learning(2011)

[8] P. Prusinkiewcz & A. Lindenmayer: “*The Algorithmic Beauty of Plants*”, Springer (1990).

[9] 丸岡章：『やさしい計算理論』サイエンス社 (2017)

[10] 富田悦次・横堀貫: 『オートマトン・言語理論（第二版)』森北出版 (2013).

[11] 細井勉: 『計算の基礎理論』教育出版 (1975).

[12] 有川節夫・宮野悟: 『オートマトンと計算可能性』培風館 (1986).

[13] 高橋正子：『コンピュータと数学』朝倉書店 (2016).

[14] C. ペゾルド: 『チューリングを読む』日経 BP (2012).

[15] 小倉久和: 『形式言語と有限オートマトン入門』コロナ社 (1996).

[16] D. N. Arden, *Delayed-logic and finite-state machines*, Switching Circuit Theory and Logical Design, pp.133 - 151(1961).

[17] A. Salomaa, *Two complete axiom systems for the algebra of regular events*, J.of ACM.13: No.1, pp.158–169(1966).

[18] D.E. Knuth, J.H. Morris and V.R. Pratt, *Fast pattern matching in strings*, SIAM Journal on Computing 6(1):323-350(1977).

[19] Jean-Paul Allouche & Jeffrey Shallit: "*Automatic Sequences – Theory, Applications Generalizations*,", Cambridge University Press (2003).

[20] M.R.Gray & D.S. Johnson: "*Computers and Intractability – A Guide to the Theory of NP-Completness*", Freeman (1979).

[21] : "*On Computable Numbers, with an Application to the Entscheidungsproblem*", Proceedings of the London Mathematical Society. 2-42: 230 – 265(1937).