情報数学・離散数学

オートマトンと言語理論

中田 昌輝

目次

図目次

表目次

1　集合・写像・証明

1.1　集合

集合 (set) とは, ある確定された対象物の集まりである. つまり「もの」の集まりである. 集合は任意の「もの」が属するかどうか明確にする必要がある. 集合に属する個々の対象物は, その集合の**元・要素** (element) とよぶ. 元に関しては以下のように記入していく.

$$a \in A \quad : \ a \text{ は集合 } A \text{ の元である.} \tag{1.1}$$

一方で b が集合 A の元でないときは

$$b \notin A \tag{1.2}$$

と表す.
有限個の要素より成る集合を**有限集合** (finite set), 無限個の要素から成る集合を**無限集合** (infinite set) という.
自然数全体の集合は $\mathbb{N}$ と表現する. また自然数に関しては $\mathbb{Z}$ となる. 集合に関する記法としては要素を全て列挙する**外延的表現**とある条件を満たすもの全ての集合を表す**内包的表現**の 2 つの表現の仕方がある.
たとえば, 要素 0,1 から成る集合は外延的表現で $\{0,1\}$ と表す. 外延的表現で集合の元であるかどうかを表現すると $0 \in \{0,1\},\ 2 \notin \{0,1\}$ となる. 一方で, これを表現する内包的表現は例えば

$$D = \{x \,|\, x \text{ は実数の集合 } \mathbb{R} \text{ に属し, かつ } x^4 - 1 = 0\}$$

と表現することも可能である. 内包的表現では “$|$” の左の元で, “$|$” の右の条件を満たすものの全ての集合を表すという表記方法である.
つまりは x に関する性質を $P(x)$ としたとき, 性質 $P(x)$ を満足するような x をすべて集めた集合を

$$\{x \,|\, P(x)\} \tag{1.3}$$

と表す. 性質 $P(y)$ と性質 $Q(y)$ とを共に満足するような y をすべて集めた集合を

$$\{y \,|\, P(y), \text{かつ } Q(y)\} \text{ , または } \{y \,|\, P(y), Q(y)\} \tag{1.4}$$

などと表す. すでに明らかな集合 S に対し, $\{x \,|\, x \in S, \text{かつ } Q(x)\}$ と定義される集合は

$$\{x \in S \,|\, Q(x)\} \tag{1.5}$$

のように表されることもある. たとえば, 実数の集合を $\mathbb{R}$ としたとき, 先の (1.3) は

$$D = \{x \in \mathbb{R} \,|\, x^4 - 1 = 0\}$$

とも表すことができる.

1.2　集合演算と基本的性質

集合 A, B において, 集合 A の元は必ず集合 B の元 (A の元はすべて B の元である) ともなっているとき, A は B の**部分集合** (subset) であるという. これを以下のように表記する.

$$A \subseteq B \quad \text{あるいは} \quad B \supseteq A \tag{1.6}$$

A 自体も A の部分集合である.
たとえば, $\{1,4\} \subseteq \{1,3,4\},\ \{1,3,4\} \subseteq \{1,3,4\}$ となる.
集合 A と集合 B が等しい ($A = B$) とは,

$$A = B \iff A \subseteq B \text{ かつ } A \supseteq B \tag{1.7}$$

となるときである. つまり A は B の部分集合であり, B は A の部分集合のとき等しくなる. 一方で等しくない場合は $A \neq B$ と表記する.
$A \subset B$ かつ $A \neq B$ のとき, A は B の**真部分集合**という. 部分集合について元を 1 つも持たない集合である**空集合**も部分集合となる. 部分集合は ϕ と表し,

$$\text{任意の集合 } A \text{ について } \ \phi \subseteq A \tag{1.8}$$

1.2.1 共通部分・和集合・差集合

共通部分 (intersection), 和集合 (union), 差集合 (difference) の定義は以下のようになる.

- 共通部分 $A \cap B = \{x \,|\, x \in A \text{ かつ } x \in B\}$
- 和集合 $A \cup B = \{x \,|\, x \in A \text{ または } x \in B\}$
- 差集合 $A \setminus B = \{x \,|\, x \in A \text{ かつ } x \notin B\}$

例を以下に挙げる.

$$\{1,2,3,4\} \cap \{3,4,5,6\} = \{3,4\}$$
$$\{1,2,3,4\} \cup \{3,4,5,6\} = \{1,2,3,4,5,6\}$$
$$\{1,2,3,4\} \setminus \{3,4,5,6\} = \{1,2\}$$

詳説

一般の和集合は記号 $\bigcup$ を用いて $\bigcup A$ と表す. 集合 $S_1, S_2, ..., S_n (n \geq 2)$ に対し,

$$S = \{x \,|\, x \in S_1, \text{あるいは } x \in S_2, ..., \text{あるいは } x \in S_n\}$$

を $S_1, S_2, ..., S_n$ の和集合といい,

$$\begin{aligned} S &= S_1 \cup S_2 \cup ... \cup S_n \\ &= \bigcup_{i=1}^{n} S_i \end{aligned}$$

この一般の和集合 $\bigcup$ の定義は

$$x \in \bigcup W \iff \exists F(F \in W \wedge x \in F) \tag{1.9}$$

つまり $A \cup B$ は $\bigcup\{A, B\}$ の中置記法である.
ある集合族 X があるとし, X の含む全ての集合の和をとる. $\bigcup$ の意味はある集合族 X に対して, ある集合 Y が存在し, この集合 Y はちょうど集合族 X の含む集合の元を全部含む.

$$\forall X \exists Y \left(Y = \bigcup X = \{a \,|\, \exists (B \in X \wedge a \in B)\} \right) \tag{1.10}$$

例として, $A = \{a\}, B = \{b\}$ とすると

$$\bigcup\{A, B\} = \bigcup\{\{a\}, \{b\}\} = \{a, b\}$$

また

$$\bigcup(\phi) = \phi$$

これは ϕ は何も含まない集合として定義されているので, 何も含まないものを合併しても何もないので結果は ϕ となる.

$$\bigcup(\{A\}) = A$$

集合 A の集合から集合 A を合併しても A となる. また,

$$\bigcup(\{\phi\}) = \phi$$

空集合からなる集合から空集合を合併しても空集合となる.

和集合と同様に, 集合族 W の共通集合を考えてみる. 共通集合を $\bigcap W$ と記述する. 以下のように共通集合に関しても定義してみる.

$$x \in \bigcap W \Longleftrightarrow \forall F(F \in W \Rightarrow x \in F)$$

これを変形すると以下のようになる.

$$x \in \bigcap W \Longleftrightarrow \forall F(\neg(F \in W) \wedge x \in F)$$

W が ϕ であるとき, $F \in W$ は偽であるので, $\neg(F \in W)$ は正となる. よって

$$x \in \bigcap W \Longleftrightarrow \forall F(\mathrm{true})$$

したがって, 任意の $x(\in \bigcap W)$ に対しても true となることを意味しているので, $\bigcap W$ は全ての集合となる (**全体集合**, **普遍集合**). このような集合は存在しないので, この定義は成立しない. ゆえに共通集合を定義することはできない.

.....

1.2.2 全体集合・補集合

数学の個々の理論において, 集合が 1 つ固定されていて, その元と部分集合のみが考察されるとき, その固定された集合を**全体集合**という. 全体集合が U であるとき, 集合 A の**補集合** (complement) は ${}^cA(\bar{A}, A^c)$ と表し,

$${}^cA = U \setminus A = \{x \in U \,|\, x \notin A\} \tag{1.11}$$

である.
例を以下に挙げる.
$U = \{a, b, c, d\},\ A = \{a, b\},\ B = \{c, d\}$ であるときそれぞれの補集合は

$${}^cA = \{c, d\} \qquad {}^cB = \{a, b\}$$

1.3 写像

1.3.1 写像の種類

A, B を集合とする. A から B への**写像** $f : A \to B$ とは各 $a \in A$ に $f(a) \in B$ を与える規則のことである.
例を以下に挙げる.
$A = \{a, b\},\ B = \{0, 1\}$ のとき, $F(a) = 1,\ f(b) = 0$
$f : \mathbb{R} \to \mathbb{N},\ f(x) = \lfloor e^x \rfloor$ (e^xを超えない最大の整数)

A を $f : A \to B$ の**定義域**という. $C \subset A$ に対して

$$f(C) = \{f(a) \,|\, a \in C\} \subset B$$

を f による C の像という. 特に $f(A)$ を f の**値域**という.
写像の性質として**単射**, **全射**, **全単射**というものがある.

- $f : A \to B$ が**単射** $\Leftrightarrow$ 違う元は違う元に行く.
  - $x \neq y \Rightarrow f(x) \neq f(y)$
  - $f(x) = f(y) \Rightarrow x = y$
  - 「f は 1 対 1 である」ともいう

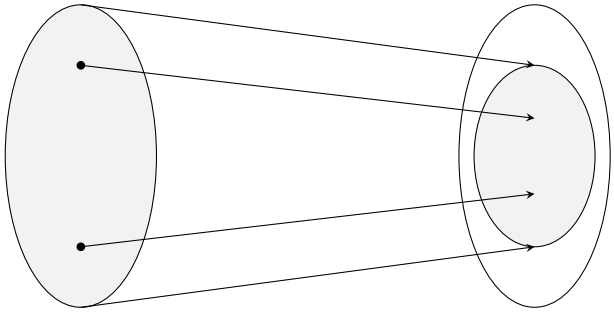


図 1.1 単射の図

- $f : A \to B$ が**全射** $\Leftrightarrow$ 全ての元に行く.
  - 全ての $y \in B$ について, ある $x \in A$ が存在して $f(x) = y$ を満たす
  - 「f は上への写像である」ともいう

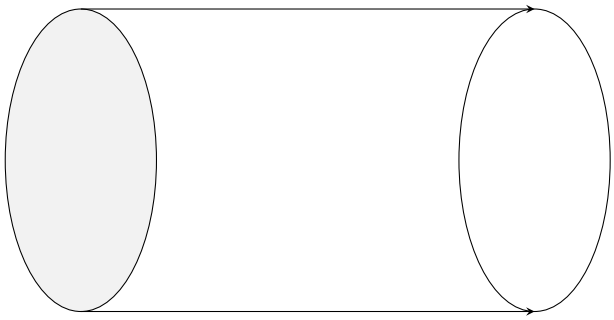
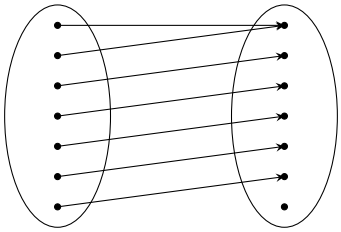
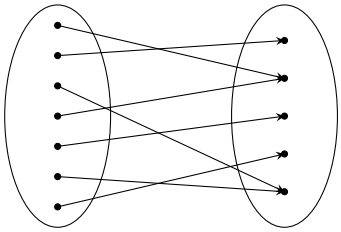


図 1.2 全射の図

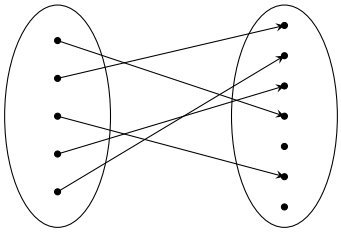
- $f : A \to B$ が**全単射** $\Leftrightarrow$ f が全射かつ単射



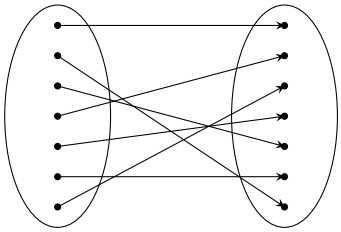
全射でも単射でもない



全射であるが単射ではない



全射ではないが単射である



全単射

図 1.3 写像の例

例を以下に挙げる.

(i) $f_1 : \mathbb{R} \to \mathbb{R}; x \mapsto x^2$
$x^2 \geq 0$ であることにより, $\mathbb{R}$ 内全域を写像しないので全射ではないことがいえる. また $x = \pm 1$ においては $x^2 = 1$ となるので, 写像先が一致し単射ではない. ゆえに全射でも単射でもない.

(ii) $f_2 : \mathbb{R} \to \mathbb{R}; x \mapsto x^3 + 3x$
$\mathbb{R}$ 上では $x^3 + 3x = x(x^2 + 3)$ と因数分解できる. 写像先が一致することはないので全単射である.

(iii) $f_3 : \mathbb{R} \to \mathbb{R}; x \mapsto x^3 - 3x$
$\mathbb{R}$ 上では $x^3 - 3x = x(x - \sqrt{3})(x + \sqrt{3})$ と因数分解できる. ゆえに, $x = 0, \pm\sqrt{3}$ においては $x^3 - 3x = 0$ となるので, 写像先が一致し単射ではない. 一方で写像先が $\mathbb{R}$ 上すべての点を通るので全射である. ゆえに全射であるが単射ではない.

(iv) $f_4 : \mathbb{R} \to \mathbb{R}; x \mapsto \dfrac{x}{1 + |x|}$

$$f = \frac{x}{1 + |x|}$$
$$f' = \frac{1}{(1 + x)^2} \qquad (x \geq 0)$$

となるので写像は単調増加関数である. また

$$\lim_{x \to \infty} \frac{x}{1 + |x|} = 1$$

関数の対称性より

$$\lim_{x \to -\infty} \frac{x}{1 + |x|} = -1$$

ゆえに図 1.4 からもわかるように単射であるが全射ではない.

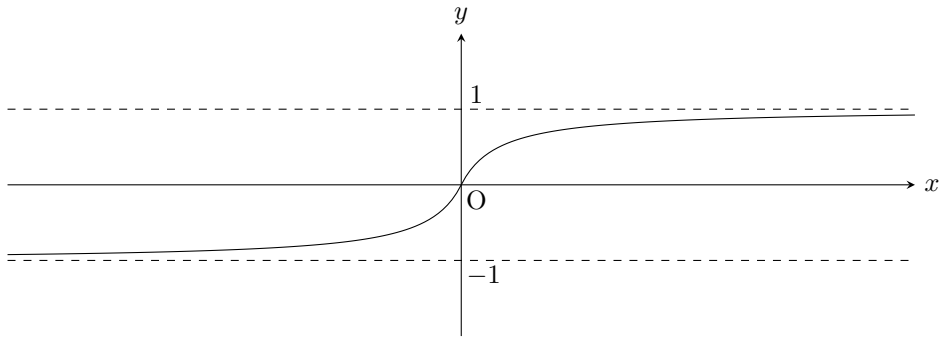


図 1.4 $y = \dfrac{x}{1+|x|}$ の概形図

1.3.2 写像についての定義

合成写像

写像について $f : A \to B, g : B \to C$ について f と g の**合成写像**は $g \circ f : A \to C$ と表し, $g \circ f(x) = g(f(x))$ と定義される.

逆写像

$f : A \to B$ が全単射であるとき, すべての $y \in B$ に $x \in A$ がただ一つ存在して $f(x) = y$ となる. 各 $y \in B$ に, この $x \in A$ を与える写像を f の**逆写像**といい, $f^{-1} : B \to A$ と書く.

逆像

任意の写像 $f : A \to B$ と $C \subset B$ について

$$f^{-1}(C) = \{a \in A \,|\, f(a) \in C\} \subset A \tag{1.12}$$

を f による C の**逆像**という.

以下に例を挙げる.

(i) $f_1 : \mathbb{R} \to \mathbb{R}; f(x) = e^x (x \mapsto e^x)$
単射であるが全射ではない. 像は

$$f_1(\mathbb{R}) = \mathbb{R}^+ = \{x \in \mathbb{R} \,|\, x > 0\}$$

(ii) $f_2 : \mathbb{Z} \to \mathbb{N}^*; f(x) = |x|$. ここで $\mathbb{N}^*$ とは 0 と自然数全体の集合.
全射であるが単射ではない. 逆像は

$$f_3^{-1}(\{1, 2\}) = \{-2, -1, 1, 2\}$$

(iii) $f_3 : \mathbb{R} \to \mathbb{R}; f(x) = e^{|x|}$
全射でも単射でもない. これは $x \mapsto |x|$ と $x \mapsto e^x$ の合成写像である.

(iv) $f_4 : \mathbb{R} \to \mathbb{R}; f(x) = -x$
全単射である. 逆写像は

$$f_4^{-1} = f_4$$

1.4 証明

1.4.1 証明とは

証明とは数学的命題が正しいことを, 正しいとした他の命題[*1](公理・定理) から導くことまたは導いた過程を書いた文書である.
証明の基本

- T の証明:最後に T が来る命題の列
- 列において良い命題:
  - 公理・定理
  - 仮定
    「A ならば B」という形の命題を証明する時, A を仮定としておいて B を導いていく.
  - 前に置かれた命題から演繹できる命題

演繹の例

- 「A かつ B」が真なら「A」も真 (「B」も)
- 「A」が真で「B」なら「A かつ B」も真
- 「A」が真なら「A または B」も真
- 「A ならば B」が真で「A」も真なら「B」は真
- 「A」から「B」が導かれれば「A ならば B」は真

証明に関しては演繹に重ねて証明するがわからなかったら定義に戻ることが重要である. つまり証明したい命題を定義に従って書いてみることで証明しやすくなる.
例えば
$f(C \cup D) = f(C) \cup f(D)$ とは $f(C \cup D) \supset f(C) \cup f(D)$ かつ $f(C \cup D) \subset f(C) \cup f(D)$ ということである.
証明手法として
背理法
A を証明するため A でないことを仮定して矛盾を導く.
対偶の証明
A ならば B を証明する代わりに B でないなら A でないことを証明する.

1.4.2 証明の例

$$f : A \to B, C \subset A, D \subset A \text{ ならば } f(C \cup D) = f(C) \cup f(D)$$

定義から考えると

$$f(C \cup D) = f(C) \cup f(D) \overset{\text{def}}{\iff} \begin{cases} f(C \cup D) \subset f(C) \cup f(D) \\ f(C \cup D) \supset f(C) \cup f(D) \end{cases}$$

となる. さらに元に着目して

$$f(C \cup D) \subset f(C) \cup f(D) \iff (b \in f(C \cup D) \Rightarrow b \in f(C) \cup f(D))$$

である. これを掘り下げて,

$$b \in f(C \cup D) \overset{\text{def}}{\iff} \text{ある } a \in C \cup D \text{ において } b = f(a)$$

[*1] 正しいか, 正しくないか確実に定まる文

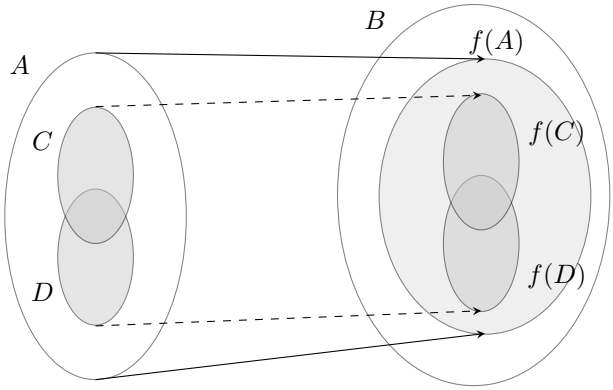


図 1.5 $f : A \to B, C \subset A, D \subset A$ ならば $f(C \cup D) = f(C) \cup f(D)$

a について

$$a \in C \cup D \overset{\text{def}}{\iff} a \in C \text{ または } a \in D$$

したがって,

$$b \in f(C) \cup f(D) \overset{\text{def}}{\iff} b \in f(C) \text{ または } b \in f(D)$$

となる.
$f(C \cup D) \supset f(C) \cup f(D)$ についても同様に考えてみる.

$$f(C \cup D) \supset f(C) \cup f(D) \iff \begin{cases} f(C \cup D) \supset f(C) \\ f(C \cup D) \supset f(D) \end{cases}$$

したがって,

$$f(C \cup D) \supset f(C) \iff (b \in f(C) \Rightarrow b \in f(C \cup D))$$

先と同様に

$$b \in f(C) \overset{\text{def}}{\iff} C \text{ のある元 } a \text{ が存在して } b = f(a)$$

また,

$$b \in f(C \cup D) \overset{\text{def}}{\iff} C \cup D \text{ のある元 } a \text{ が存在して } b = f(a)$$

これを証明の形に直すと以下のようになる.

$b \in f(C)$ とすると, C の元 a が存在して $b = f(a)$.
$a \in C$ より $a \in C \cup D$ だから $b \in f(C \cup D)$
「$f(C)$ の元はすべて $f(C \cup D)$ の元」だから $f(C \cup D) \supset f(C)$. 同様に $f(C \cup D) \supset f(D)$. したがって,

$$f(C \cup D) \supset f(C) \cup f(D)$$

また, $b \in f(C \cup D)$ とすると, $C \cup D$ の元 a が存在して $b = f(a)$
$a \in C$ または $a \in D$ で $a \in C$ なら $b \in f(C)$, $a \in D$ なら $b \in f(D)$ となる. だから $b \in f(C)$ または $b \in f(D)$. したがって, $b \in f(C) \cup f(D)$.

$$f(C \cup D) \subset f(C) \cup f(D)$$

以上より

$$f(C \cup D) = f(C) \cup f(D)$$

同じような問題として

$$f : A \to B, C \subset A, D \subset A \text{ ならば } f(C \cap D) = f(C) \cap f(D)$$

について考えてみる.
これを図を用いて表現してみると以下のようになる.

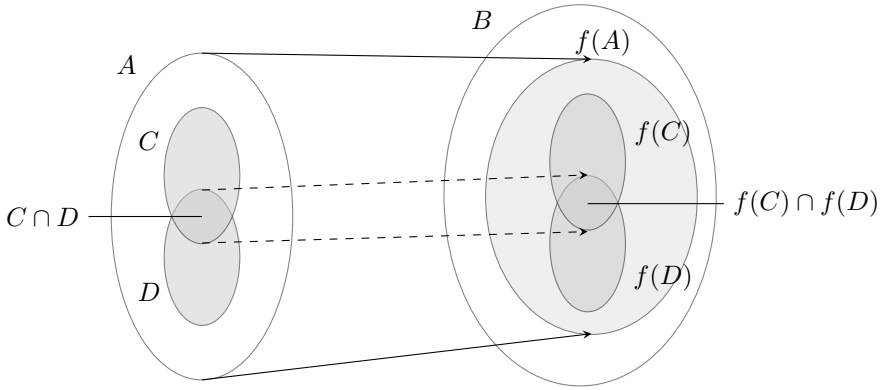


図 1.6 $f : A \to B, C \subset A, D \subset A$ ならば $f(C \cap D) = f(C) \cap f(D)$

この場合についても C に属する元の写像先は $f(C)$ に属し, D に属する元の写像先は $f(D)$ に属するから成り立っているように思える (実際, 図 1.6 では正しい).
しかし $C \cap D = \phi$ であるときは例外である. というのは $f(\phi) = \phi$ であるが, $f(C \cap D) = \phi$ とは限らないからである. ゆえにこの命題は偽である.

反例
$A = \{0, 1\}, B = \{a\}, C = \{0\}, D = \{1\}, f(0) = f(1) = a$ とすると,

$$f(C \cap D) = f(\phi) = \phi$$
$$f(C) \cap f(D) = \{a\} \cap \{a\} = \{a\}$$

類題として

$$f : A \to B, C \subset B, D \subset B \text{ ならば } f^{-1}(C \cap D) = f^{-1}(C) \cap f^{-1}(D)$$

正しい.
証明
$x \in f^{-1}(C \cap D)$ とすると, $f(x) \in C \cap D$. $f(x) \in C \cap D$.
$f(x) \in C$ だから $x \in f^{-1}(C)$. $f(x) \in D$ だから $x \in f^{-1}(D)$. つまり $x \in f^{-1}(C) \cap f^{-1}(D)$
したがって, $f^{-1}(C \cap D) \supset f^{-1}(C) \cap f^{-1}(D)$.
以上より

$$f^{-1}(C \cap D) = f^{-1}(C) \cap f^{-1}(D)$$

1.5 集合族

集合の元がそれ自体集合であることもある.
例を以下に挙げる.

$$\{\phi, \{0\}, \{0,1\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,2,3,4\}, ...\}$$
$$\{\phi, \{\phi\}, \{\phi, \{\phi\}\}, \{\phi, \{\phi\}, \{\phi, \{\phi\}\}\}, ...\}$$

集合 A の部分集合全体の集合を A の**冪集合** (powerset) といい, 2^A と書き表す.
例を以下に挙げる.

$$A = \{a,b,c\} \text{ のとき}, 2^A = \{\phi, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$$
$$2^\phi = \{\phi\}$$
$$2^{\{\phi\}} = \{\phi, \{\phi\}\}$$

ここで “自分自身を元として含まない集合” 全体の集合 X を考える. つまり

$$X = \{Y : \text{集合} \,|\, Y \notin Y\}$$

において, X が X に属すると仮定した場合, 集合 X の定義の $Y \in Y$ が成り立たなくなる. つまり

$$X \in X \Rightarrow X \notin X$$

また, X が X に属しないと仮定した場合は, 集合 X の定義の $Y \notin Y$ が成立し, $X \in X$ となってしまい矛盾が起きる. つまり

$$X \notin X \Rightarrow X \in X$$

このように定義した集合 X は矛盾が生じる. このような矛盾をラッセル (Russel) のパラドックスという. 現代数学において $\{X \,|\, X \notin X\}$ は集合ではないと考えている. このような矛盾は自分自身を否定することで生じる. たとえば「僕は嘘つきである.」に関しては嘘という自分自身の発言の否定をすることでこの発言に矛盾が生じることとなる. 集合論に関してはこのようなことに関しては厳密に定義が必要であるとする. またこのような矛盾など歴史的経緯から, 集合を要素とする集合を**集合族**と呼ぶことがある.

1.5.1 有限集合・無限集合

集合 A, B について全単射 $f : A \to B$ が存在する時, A と B は**対等**であるといい $A \sim B$ とかく.
対等であるとは, 元の間に 1 対 1 対応 (one-to-one mapping) があるということである.
つまり, 集合 A, B と写像 $f : A \to B, g : B \to A$ が存在して

$$\forall x \in A(g(f(x)) = x) \wedge \forall y \in B(f(g(y)) = y) \tag{1.13}$$

とできるときであり, f や g を 1 対 1 対応という. 対等な集合は同一視可能である.
以下に例を挙げる.
$\mathbb{N}$ と $\mathbb{Z}$ は対等である. つまり $\mathbb{N} \sim \mathbb{Z}$. これの対応づけは

$$f : \mathbb{N} \to \mathbb{Z},\ f(n) = (-1)^n \left\lfloor \frac{n+1}{2} \right\rfloor$$

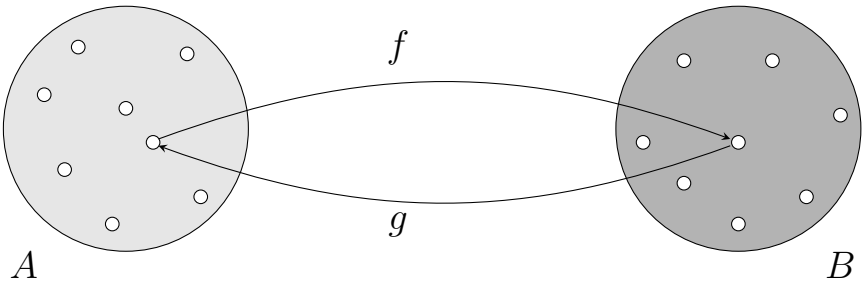


図 1.7 対等な集合

集合 A が対等な真部分集合 $B \subseteq A$, $B \neq A$ を持つとき, A は**無限集合**という.
以下に例を挙げる.

(i) $f : \mathbb{N} \to \mathbb{N}^+, f(n) = n + 1$ により自然数全体 $\mathbb{N}$ と正の自然数全体 $\mathbb{N}^+$ は対等である. ゆえに $\mathbb{N}$ は無限集合である.

(ii) $f : \mathbb{R} \to \mathbb{R}^+, f(x) = e^x$ により実数全体 $\mathbb{R}$ と正の実数全体 $\mathbb{R}^+$ は対等である. ゆえに $\mathbb{R}$ は無限集合である.

一方で, 無限集合でない集合を**有限集合**という.

1.5.2 濃度・可算集合

集合 A, B が対等のとき, A, B の**濃度**が等しいともいう. これを $|A| = |B|$ と書く.A が有限集合のときは, $|A|$ で $|A|$ の元の数を表す.
$\mathbb{N}$ と対等な集合を**可算な集合 (可算集合)** という. 可算集合は番号をつけられる集合である. 有限か, 可算であるとき, **たかだか可算**という. $\mathbb{Z}$ や, 有理数全体 $\mathbb{Q}$ は可算である.
有理数全体の集合 $\mathbb{Q}$ が可算であることについて, 正の有理数について番号を振ることができれば有理数全体に関しても同様に番号を振ることが可能である ($+, =$ と交互に番号を振ればいいため). 正の有理数に関しては図のように番号を振ればよい (ここで, すでに現れているものは飛ばす).

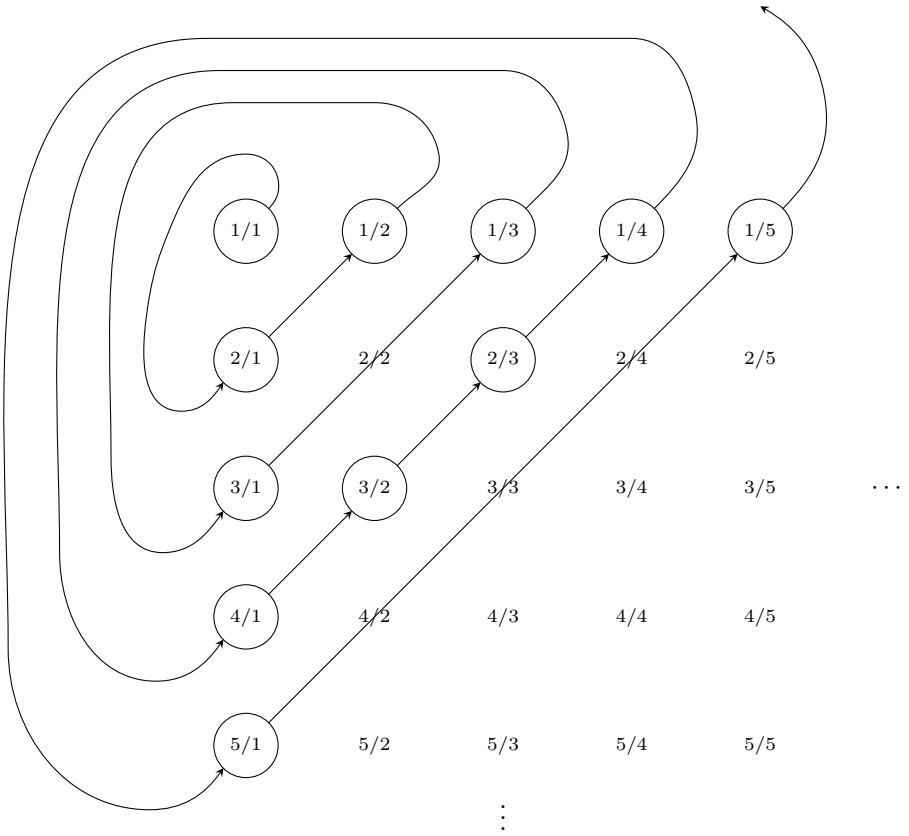


図 1.8 有理数全体の集合 $\mathbb{Q}$ が可算であるという証明の図

実数全体の集合 $\mathbb{R}$ に関しては非可算である. 以下, 証明をする. この証明は背理法であり**対角線論法**と呼ばれる.

$\mathbb{N}$ と $\mathbb{R}$ が対等でないことを示せばよい.
$f : \mathbb{N} \to \mathbb{R}$ が全単射であると仮定する. f を使って, 実数全体の一覧表が作れる. この表にない実数 x を以下のように作る.
x の小数点以下 n 桁目は $f(n)$ の n 桁目と異なる数とする. x は表の n 番目の実数とは小数点以下 n 桁目が異なる. よって x はすべての実数と異なる. これは最初に実数全体の一覧表を作ったことと矛盾する.

n	$f(n)$
0	3.14159...
1	1.41421...
2	2.74232...
3	0.98476...
⋮	⋮

このようなケースの場合, 新たに $x = 2.165...$ などと作ればよい.

1.6 関係

1.6.1 順序対

集合 A, B の元を一つずつとった, $a \in A$, $b \in B$ の組 $\langle a, b\rangle$ または (a, b) を**順序対**という. 順序をつける (順序を無視しない) ので,

$$\langle a, b\rangle \neq \langle b, a\rangle \tag{1.14}$$

集合 $\{a, b\}$(非順序対) に関しては

- $\{a, b\} = \{b, a\}$
- $\{a, a\} = \{a\}$

.....

詳説

順序対は非順序対があれば定義できる. 順序対の定義は以下の二つである.

$$\langle a, b\rangle = \{\{a\}, \{a, b\}\} \tag{1.15}$$
$$\langle a, b\rangle = \{\{a, 0\}, \{b, 1\}\} \tag{1.16}$$

順序対は以下の性質が成立する.

$$\langle a, b\rangle = \langle c, d\rangle \Longleftrightarrow a = c \land b = d \tag{1.17}$$

これを式 (1.15),(1.16) で成立するか確かめる.
式 (1.15) について,$\{a\} = \{c\} \land \{a, b\} = \{c, d\}$ と $\{a\} = \{c, d\} \land \{a, b\} = \{c\}$ が考えられる. まず以下のケースについて考えてみる.

$$a = c \land ((a = c \land b = d) \lor (a = d \land b = c))$$

この場合は, 以下の二つのパターンが考えられる.

(i)

$$a = c \land b = d$$

(ii)

$$a = c \land (a = d \land b = c) \Longleftrightarrow a = b = c = d$$

したがって, (i),(ii) より

$$\{a\} = \{c\} \land \{a, b\} = \{c, d\} \Longrightarrow a = c \land b = d$$

となる. 次に, $\Longleftarrow$ について示す.

$$\begin{aligned} a = c \land b = d &\Longrightarrow a = c \land (a = c \land b = d) \\ &\Longrightarrow \{a\} = \{c\} \land \{a, b\} = \{c, d\} \end{aligned}$$

となる. ゆえに式 (1.15) において

$$\{a\} = \{c\} \land \{a, b\} = \{c, d\} \Longleftrightarrow a = c \land b = d$$

つぎに,

$$\{a\} = \{c, d\} \land \{a, b\} = \{c\}$$

の場合について考える. 元が一つと元が二つという関係性に着目すれば,

$$a = c = d \wedge c = a = b \Longleftrightarrow a = b = c = d$$

したがって, 式 (1.15) において, 性質 (1.17) が成り立つことが分かる.
次に式 (1.16) について考えてみる.
$\{\{a,0\},\{b,1\}\} = \{\{c,0\},\{d,1\}\}$ であるとき, $\{a,0\} = \{c,0\} \wedge \{b,1\} = \{d,1\}$ と $\{a,0\} = \{d,1\} \wedge \{b,1\} = \{c,0\}$ が考えられる. まず, $\{a,0\} = \{c,0\} \wedge \{b,1\} = \{d,1\}$ のケースについて

$$(a = c \wedge b = d) \vee (a = 0 \wedge c = 0 \wedge b = 1 \wedge d = 1) \Longrightarrow a = c \wedge b = d$$

つぎに, $a = c \wedge b = d \Longrightarrow \{a,0\} = \{c,0\} \wedge \{b,1\} = \{d,1\}$ を示す.

$$\begin{aligned} a = c \wedge b = d &\Longleftrightarrow a = c \wedge 0 = 0 \wedge b = d \wedge 1 = 1 \\ &\Longrightarrow \{a,0\} = \{c,0\} \wedge \{b,1\} = \{d,1\} \end{aligned}$$

$\{a,0\} = \{d,1\} \wedge \{b,1\} = \{c,0\}$ のケースについて

$$a = 1 \wedge d = 0 \wedge b = 0 \wedge c = 1 \Longleftrightarrow a = c = 1 \wedge b = d = 0$$

となる. したがって, 式 (1.16) の場合も性質 (1.17) が成り立つことが分かる.

1.6.2 直積集合

A, B の元の順序対全体の集合

$$A \times B = \{\langle a,b\rangle \,|\, a \in A, b \in B\} \tag{1.18}$$

を A と B の**直積集合**, あるいは単に**直積**という.
たとえば, $A = \{a,b\}, B = \{0,1\}$ とすると

$$A \times B = \{\langle a,0\rangle, \langle a,1\rangle\langle b,0\rangle\langle b,1\rangle\}$$

となる. 三つの集合の直積は以下のようになる.

$$A \times B \times C = \{\langle a,b,c\rangle \,|\, a \in A, b \in B, c \in C\} \tag{1.19}$$

同様にして n 個の集合の直積は以下のように定義できる.

$$A_1 \times A_2 \times \cdots A_n = \{\langle a_1, a_2, \cdots, a_n\rangle \,|\, a_i \in A_i,\ i = 1,2,...,n\} \tag{1.20}$$

$x - y - z$ 空間は実数の集合 $\mathbb{R}3$ つの直積である $\mathbb{R}^3 = \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ となる.

詳説

n 個の直積における名前を以下に挙げる.

- $A \times B$ ペア (対, pair) の集合
- $A \times B \times C$ 三つの組 (triple/triplet) の集合
- $A \times B \times C \times D$ 四つの組 (quadruple/quadruplet) の集合
- 一般に, n 個のものの組を n-tuple という
  - 個数を明示しない場合は単に tuple という

また次の性質がある.

$$A \times B \neq B \times A (A \text{ または } B \text{ が} \phi \text{のとき, または } A = B \text{ のとき不成立}) \tag{1.21}$$
$$(A \times B) \times C \neq A \times (B \times C) \tag{1.22}$$

空集合と直積の関係は以下である.

$$A \times \phi = \phi \quad , \quad \phi \times A = \phi \tag{1.23}$$

N 要素ベクトル (配列) の集合に関して

- $A \times A = A^2$
- $A \times A \times A = A^3$
- $\overbrace{A \times A \times \cdots \times A}^{n \text{ 個}} = A^n$
- A^0 は一つの元からなる集合 (単集合). 空集合ではない.
- $A^1 = A$

.....

1.6.3 関係

集合 A の 2 元の間の「関係」は, その関係にある順序対からなる直積 $A \times A$ の部分集合を定める.
たとえば, 等しいという関係を表す $=$ は対角線集合を定める. $x-y$ 平面のグラフ $y=x$ は「等しい」という関係にある順序対 $\langle x, y \rangle$ からなる直積 $\mathbb{R} \times \mathbb{R}$ の部分集合

$$\{\langle x, y \rangle \in \mathbb{R} \times \mathbb{R} \,|\, x = y\}$$

である.
このことから, 直積 $A \times A$ の任意の部分集合を A 上の **2 項関係**と呼ぶ. 異なる集合の場合: $A \times B$ の任意の部分集合も関係と呼ぶ. 3 項以上においても同様に, 3 項関係, 4 項関係, ... と呼ぶ.
ここで, X, Y を集合とする. 順序対 $\langle x, y \rangle \in X \times Y$ に対して満たすか満たさないかが判別できるある規則 R(X から Y への関係) について, 順序対 $\langle x, y \rangle$ が関係 R を満たせば xRy とかく. 以下に例を挙げる.
ある関係 R: ○ ○ ○ さんが飼っているペットは △ △ △ である.

- $P = \{a, b, c, d\}$
- $Q = \{$ 犬, 猫, 鳩 $\}$
- $R = \{\langle a, \text{犬} \rangle, \langle b, \text{犬} \rangle, \langle d, \text{鳩} \rangle\} = P \times Q$ の部分集合

ここで a さんは犬を飼っているので, aR 犬 とかける.

1.6.4 関係の性質

関係 $R \subset A \times A$ の性質として次のものがある.

- R が**反射的**: $\forall x \in A(xRx)$
  任意の $x \in A$ について $\langle x, x \rangle \in R$
- R が**対称的**: $\forall x \in A \forall y \in A(xRy \Longrightarrow yRx)$
  任意の $x, y \in A$ について「$\langle x, y \rangle \in R$ ならば $\langle y, x \rangle \in R$」
- R が**反対称的**: $\forall x \in A \forall y \in A(xRy \wedge yRx \Longrightarrow x = y)$
  任意の $x, y \in A$ について「$\langle x, y \rangle \in R$ かつ $\langle y, x \rangle \in R$ ならば $x = y$」
- R が**推移的**: $\forall x \in A \forall y \in A \forall z \in A(xRy \wedge yRz \Longrightarrow xRz)$
  任意の $x, y, z \in A$ について「$\langle x, y \rangle \in R$ かつ $\langle y, z \rangle \in R$ ならば $\langle x, z \rangle \in R$」

表 1.1 関係の性質

反射的

	a	b	c	d
a	$*$			
b		$*$		$*$
c	$*$		$*$	
d	$*$			$*$

対称的

	a	b	c	d
a		$*$		$*$
b	$*$	$*$		
c				$*$
d	$*$		$*$	$*$

反対称的

	a	b	c	d
a		$*$		$*$
b		$*$		
c	$*$			
d			$*$	$*$

推移的

	a	b	c	d
a		$*$	$*$	$*$
b			$*$	$*$
c			$*$	$*$
d			$*$	$*$

関係の性質は以下に挙げる.

表 1.2 関係の性質の有無の例

	○	×
反射的	$=, \leq$	$<$
対称的	$=$	$\leq, <$
反対称的	$=, \leq$	$<$
推移的	$=, \leq, <$	絶対値の差が 2 である関係

反射的に関して, $x = x$ や $x \leq x$ は成り立つものの, $x < x$ は成立しない. 対称的に関して $x = y \Rightarrow y = x$ であるが $x \leq y \not\Rightarrow y \leq x$ である. 反対称的に関して, $x = y \Rightarrow y = x$ となるのは $x = y$ のときである. $x \leq y \Rightarrow y \leq x$ となるのは $x = y$ のときである. $x < y \Rightarrow y < x$ となるときはないので不適. 推移的に関して $x < y$ かつ $y < z$ ならば, $x < z$ である. しかし $|x - y| = 2$ かつ $|y - z| = 2$ ならば $|x - z| \neq 2$ であるので不適.

1.6.5 同値関係

同値関係とは反射的かつ対称的かつ推移的な関係のことである. 同値関係 $R \subset A \times A$ について $\langle x, y \rangle \in R$ を $x \sim {}_R y$ とかく.

反射的を $x \sim {}_R x$, 対称的を $x \sim {}_R y \Rightarrow y \sim {}_R x$, 推移的を $x \sim {}_R y, y \sim {}_R z \Rightarrow x \sim {}_R z$ と表せる.

同値関係にあるもの同士のグループに分けることをができる. このグループを**同値類**と呼ぶ. 元 $a \in A$ に含まれる同値類を $[a]_R$ とかき, 同値類の集合を A/R とかく. 以下に例を挙げる.

$$R = \{\langle x, y \rangle \,|\, \text{ある } z \in \mathbb{Z} \text{ で } x - y = 2z\} \subset \mathbb{Z} \times \mathbb{Z}$$

とすると, 同値類は 2 つであり, 偶数と奇数の集合である. ゆえに以下のようにかける.

$$\mathbb{Z}/R = \{[0]_R, [1]_R\}$$

また, 偶数の同値類に関しても

$$[0]_R = \{n \in \mathbb{Z} \,|\, n \text{ は偶数 }\} = [2]_R = [4]_R = \cdots$$

詳説

集合 A 上の関係を合成することを考える. つまり以下の式について考える.

$$R_1 \circ R_2 = \{\langle x, z \rangle \,|\, x \in A \wedge z \in A \wedge \exists y \in A(\langle x, y \rangle \in R_1 \wedge \langle y, z \rangle \in R_2)\} \tag{1.24}$$

合成の繰り返しを帰納的に定義すると以下のようになる.

$$R^0 = \{\langle x, x \rangle \,|\, x \in A\} \tag{1.25}$$
$$R^{n+1} = R \circ R^n \quad (n \in \mathbb{N}) \tag{1.26}$$

ここで, $R \circ R = R^2$ とかけるが, これは直積の $A \times A = A^2$ ではないので注意.
推移的である関係をもつ推移閉包 (transitive closure) については以下で定義される.

$$R^+ = \bigcup_{n \in \mathbb{N}} R^{n+1} \tag{1.27}$$

また反射推移閉包 (reflexive transitive closure) については以下で定義される.

$$R^* = \bigcup_{n \in \mathbb{N}} R^n \ (= R^0 \cup R^+) \tag{1.28}$$

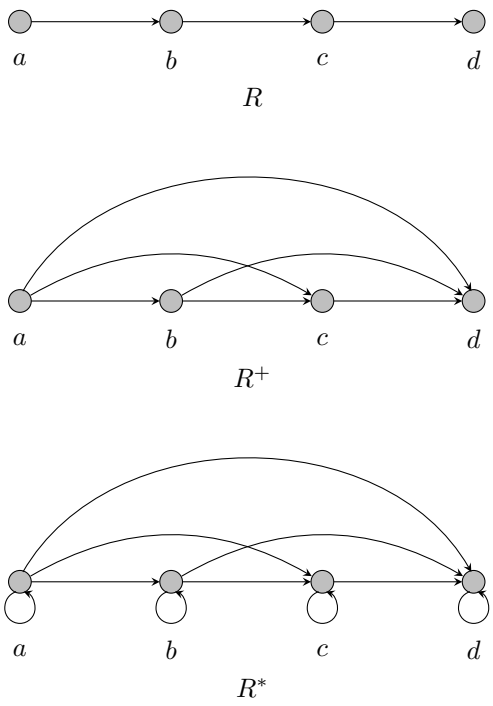


図 1.9 閉包の関係

1.6.6 **有向グラフ・無向グラフ**

有限個の頂点を矢印 (辺) でつないだものを**有向グラフ**と呼ぶ.
頂点を**接点**, 辺を**枝**とも呼ぶ.

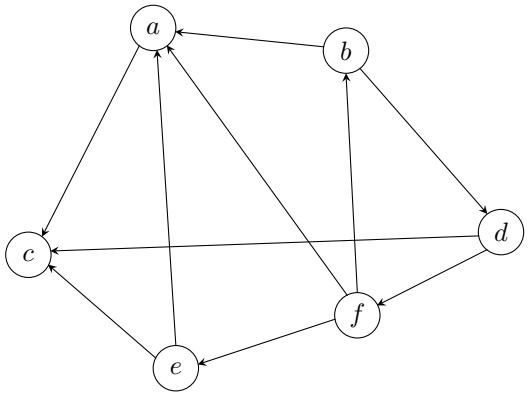


図 1.10 有向グラフ

有向グラフを数学的に定義することを考える. すると頂点の (有限) 集合 V と, V の二項関係 E の組

$$G = \langle V, E \rangle \qquad (E \subset V \times V) \tag{1.29}$$

と考えればよい. ここで, $\langle u, v \rangle \in E$ が頂点 u から頂点 v への辺を表す.
先の図 1.10 で, V, E は以下のように書き表すことができる.

$$V = \{a, b, c, d, e, f\}$$
$$E = \{\langle a, c\rangle, \langle b, a\rangle, \langle b, d\rangle, \langle d, c\rangle, \langle d, f\rangle, \langle e, a\rangle, \langle e, c\rangle, \langle f, a\rangle, \langle f, b\rangle, \langle f, e\rangle\}$$

これに対し, E が対称的なとき, 2 頂点の間には辺がないか, あれば常に両向きにある. つまり向きがないと考えられるのでこのようなグラフを**無向グラフ**という.

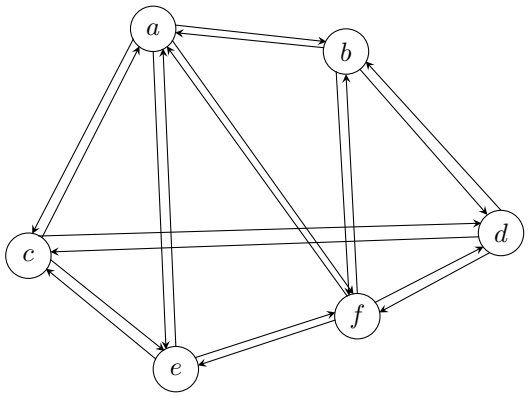


図 1.11 矢印で表現した無向グラフ

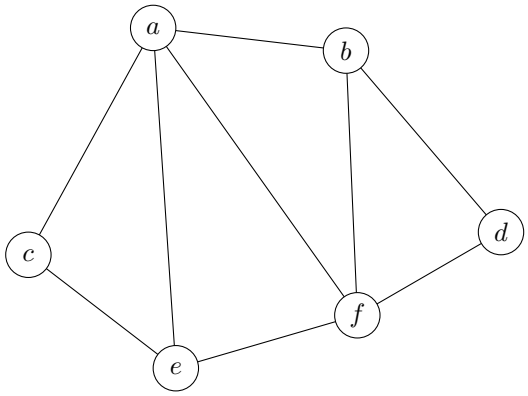


図 1.12　矢印無しの無向グラフ

2　命題論理, 述語論理

2.1　命題と真理値

命題とは真か偽か定まっている言明のことである. 例を以下に挙げる.

- $1+1=2$　　(真命題)
- 素数は無限に存在する　　(真命題)
- 世界は像の背中に乗っている　　(偽命題)
- x, y, z が 1 以上の自然数, n が 2 以上の自然数で $x^n+y^n=z^n$ なら $n=2$　　(真命題)
- 風が吹けば桶屋が儲かる　　(必ずしも真でも偽でもないので命題でない)
- $x=1$　　(真偽が x によるので命題ではない)

命題に対して定まっている真か偽の値を**真理値**という. 真を t または 1 で, 偽を f または 0 で表す. t,f を**命題定数**, 真理値を表す変数を**命題変数**という.

2.2　論理式

2.2.1　論理記号

論理記号とは 1 つ以上の命題から新しい命題を作る記号のことである. 以下の記号がある.

- $A \vee B$: A または B
  - **論理和**という
  - A OR B , $A+B$ とも表記する
- $A \wedge B$: A かつ B
  - **論理積**という
  - A AND B , $A \cdot B$ とも表記する
- $\neg A$: A でない
  - **否定**という
  - NOT A, $\bar{A}$ とも表記する
- $A \Rightarrow B$: A ならば B
  - **含意**という
- $A \Leftrightarrow B$: (A ならば B) かつ (B ならば A)
  - **同値**という

命題の真理値の対応表を**真理値表**という. 各論理記号における真理値表は以下のとおりである.

表 2.1 各論理記号における真理値表

A	B	$A \vee B$
t	t	t
t	f	t
f	t	t
f	f	f

A	B	$A \wedge B$
t	t	t
t	f	f
f	t	f
f	f	f

A	$\neg A$
t	f
f	t

A	B	$A \Rightarrow B$
t	t	t
t	f	f
f	t	t
f	f	t

A	B	$A \Leftrightarrow B$
t	t	t
t	f	f
f	t	f
f	f	t

論理記号 AND, OR, NOT に関しては論理回路が存在し, 以下の回路で表現する.

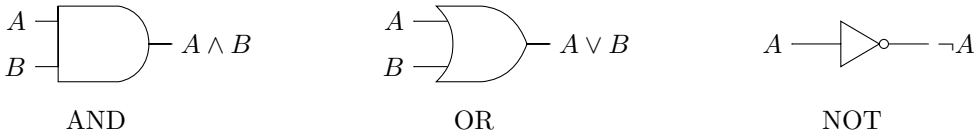


図 2.1 論理回路

含意の $A \Rightarrow B$ を「ならば」と考えられるが, A が偽ならば B に関わらず $A \Rightarrow B$ は真となる. したがって, たとえば $0 = 1$ならば私はローマ法王は真である. ゆえに含意が偽となるのは A が真で B が偽のときのみである.

2.2.2 恒真論理式 (トートロジー)

命題定数 t と f はそれぞれ**論理式**である. また命題変数も論理式である. A, B が論理式であるとき

$$(A \vee B), (A \wedge B), (\neg A), (A \Rightarrow B)$$

はそれぞれ論理式である. ここで, 一番外側の括弧は省略することができ, $(\neg A)$ の括弧も省略することが可能である.

論理式の含むすべての命題変数の真理値が決まれば, 論理式の真理値が決まる.. 論理式の含む命題変数の真理値の組合せ全てで論理式の真理値が真となるとき, その論理式を**恒真論理式**または**トートロジー**という. 以下に例を挙げる.

$$(A \vee A) \Leftrightarrow A \text{ はトートロジー}$$

$(A \vee A) \Rightarrow A$ と $A \Rightarrow (A \vee A)$ が常に (A が真でも偽でも) 共に真である.

- A が真のとき
  $A \vee A$ も真
- A が偽のとき
  $A \vee A$ も偽
  X が偽ならば $X \Rightarrow Y$ は真になるので, A が偽のとき $(A \vee A) \Leftrightarrow$ は成立する.

ゆえに A の真偽に関わらず, $(A \vee A) \Leftrightarrow A$ であるため, この命題はトートロジーである.

論理式がトートロジーかどうかを判定するには全ての組合せを試せばよい $\Longrightarrow$ 常に有限の手続で判定することが可能である. 以下に例を挙げる.

表 2.2 トートロジーの判定法

A	B	$A \vee B$	$A \wedge (A \vee B)$	$(A \wedge (A \vee B)) \Leftrightarrow A$
f	f	f	f	t
t	f	t	t	t
f	t	t	f	t
t	t	t	t	t

A	B	$A \Rightarrow B$	$\neg A$	$\neg A \vee B$	$(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$
f	f	t	t	t	t
t	f	f	f	f	t
f	t	t	t	t	t
t	t	t	f	t	t

論理式 A, B について, $A \Leftrightarrow B$ がトートロジーであることを A と B は**同等**であるといい, $A \equiv B$ とかく. 基本的なトートロジーを以下に挙げる.

- $(A \vee A) \Leftrightarrow A,\ (A \wedge A) \Leftrightarrow A$ (巾等律)
- $(A \vee (B \vee C)) \Leftrightarrow ((A \vee B) \vee C)$,
  $(A \wedge (B \wedge C)) \Leftrightarrow ((A \wedge B) \wedge C)$ (結合律)
- $(A \vee B) \Leftrightarrow (B \vee A),\ (A \wedge B) \Leftrightarrow (B \wedge A)$ (交換律)
- $(A \wedge (A \vee B)) \Leftrightarrow A,\ (A \vee (A \wedge B)) \Leftrightarrow A$ (吸収律)
- $(A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C))$,
  $(A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C))$ (分配律)

その他にも基本的なトートジーとして以下のものがある. 以下のトートロジーは論理式の変形でよく用いられる.

- $(\neg(\neg A)) \Leftrightarrow A$ (二重否定の除去)
- $(\neg(A \vee B)) \Leftrightarrow (\neg A \wedge \neg B),\ \neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$ (**ド・モルガンの法則**)
- $(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$ (含意記号の置換)

命題定数を含むトートロジーに関しては以下のようなものがある.

- $\neg \mathrm{t} \Leftrightarrow \mathrm{f},\ \neg \mathrm{f} \Leftrightarrow \mathrm{t}$
- $(A \wedge \mathrm{t}) \Leftrightarrow A,\ (A \vee \mathrm{f}) \Leftrightarrow A,\ (A \wedge \mathrm{f}) \Leftrightarrow \mathrm{f},\ (A \vee \mathrm{t}) \Leftrightarrow \mathrm{t}$
- $(A \wedge \neg A) \Leftrightarrow \mathrm{f}$ (矛盾律)
- $(A \vee \neg A) \Leftrightarrow \mathrm{t}$ (排中律)
- $\neg A \Leftrightarrow (A \Rightarrow \mathrm{f}),\ A \Leftrightarrow (\mathrm{t} \Rightarrow A)$

2.2.3 充足可能な論理式

論理式の含む命題変数の真理値の組合せ全てについて真となるときトートロジーと呼んだ. 一方で, 論理式が真となる命題変数の真理値の組合せが一つでもある論理式は**充足可能**であるという. 対して, 充足可能でない論理式を**充足不可能**という.
論理式 A が充足不可能である必要十分条件は論理式 $\neg A$ がトートロジーであることである.

これを示す.

論理式 A が充足不可能である $\iff$ 論理式 A が全ての命題変数の真理値の組合せについて偽である
$\iff$ 論理式$\neg A$ が全ての命題変数の真理値の組合せについて真である
$\iff$ $\neg A$ がトートロジーである

2.2.4 論理式の標準形

命題変数, または命題変数の直前に否定記号がひとつだけ付いた形の論理式を**リテラル**という. 命題変数 A を正リテラル, $\neg B$ を負リテラルという. このリテラルをいくつかの論理積で表現したものを**論理積項**という. 以下に例を挙げる.

$$A \wedge \neg B,\ \neg B \wedge C,\ \neg A \wedge B \wedge C,\ ...$$

また, 論理積項の論理和の形を**論理和標準形 (Disjunctive Normal Form: DNF)** という. 以下に例を挙げる.

$$(A \wedge \neg B) \vee (\neg B \wedge C) \vee (\neg A \wedge B \wedge C)$$

積ではなく, リテラルをいくつかの論理和で表現したものを**論理和項**という. 以下に例を挙げる.

$$\neg A \vee \neg B,\ B \vee C,\ \neg A \vee B \vee \neg C,\ ...$$

また, 論理和項の論理積の形を**論理積標準形 (Conjunctive Normal Form: CNF)** という. 以下に例を挙げる.

$$(\neg A \vee \neg B) \wedge (B \vee C) \wedge (\neg A \vee B \vee \neg C)$$

論理和の拡張として $\bigvee$ を用いて

$$\bigvee_{i=1}^{n} A_i = A_1 \vee A_2 \vee \cdots \vee A_n \tag{2.1}$$

論理積の拡張として $\bigwedge$ を用いて

$$\bigwedge_{i=1}^{n} A_i = A_1 \wedge A_2 \wedge \cdots \wedge A_n \tag{2.2}$$

論理式は必ず論理和標準形でも論理積標準形でもかける. つまり, 任意の論理式 A について, 次の論理式がトートロジーになるようなリテラル L_{ij} および M_{ij} が存在する.

$$A \iff \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n_i} L_{ij} \tag{2.3}$$

$$A \iff \bigvee_{i=1}^{k} \bigwedge_{j=1}^{l_i} M_{ij} \tag{2.4}$$

項毎にリテラルの数は異なる (n_i, l_i). 例えば, $(\neg A \vee \neg B) \wedge (C \vee D \vee \neg E)$ であるとリテラルが 2 個と 3 個の項の論理積で表される.
次に任意の論理式を論理和標準形にすることを考える. 数学的に機械的に行う方法は以下である.

1. $A \Leftrightarrow B$ を $(A \Rightarrow B) \wedge (B \Rightarrow A)$ で置換
2. $A \Rightarrow B$ を $\neg A \vee B$ で置換
3. $\neg(A \vee B)$ を $\neg A \wedge \neg B$ で, $\neg(A \wedge B)$ を $\neg A \vee \neg B$ で置換して, $\neg$ が $\wedge$ や $\vee$ の内側に現れるようにする (ド・モルガンの法則)

4. 偶数個の $\neg$ を除去 (二重否定の除去)
5. $A \wedge (B \vee C)$ を $(A \wedge B) \vee (A \wedge C)$ で置換して, $\wedge$ の内側の $\vee$ を外側に出す (分配律)
6. さらにトートロジーを用いて同値な論理式に変形してもよい

以下に例を挙げる.

$$\begin{aligned}\neg(A \Rightarrow (B \wedge C)) &\overset{2}{\iff} \neg(\neg A \vee (B \wedge C)) \\ &\overset{3}{\iff} \neg\neg A \wedge \neg(B \wedge C) \\ &\overset{3,4}{\iff} A \wedge (\neg B \vee \neg C) \\ &\overset{5}{\iff} (A \wedge \neg B) \vee (A \wedge \neg C) \qquad \text{(論理和標準形)}\end{aligned}$$

また, さらに次のような変形をすることができる.

$$\begin{aligned}A \wedge \neg B &\iff A \wedge \neg B \wedge \mathrm{t} \\ &\iff A \wedge \neg B \wedge (C \vee \neg C) \\ &\iff (A \wedge \neg B \wedge C) \vee (A \wedge \neg B \wedge \neg C)\end{aligned}$$

より,

$$\neg(A \Rightarrow (B \wedge C)) \iff (A \wedge \neg B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg C)$$

とすることができるので, 標準形は一意に定まらない.

次の方法は, 論理回路の分野で多く用いられる. 命題変数の真理値の特定の組合せのときのみに t となる論理積項が簡単に作れる. たとえば, $A \wedge \neg B \wedge \neg C$ は $(A, B, C) = (\mathrm{t}, \mathrm{f}, \mathrm{f})$ のときのみ t である. これを利用して複数の組合せについて t とする各組合せについて論理積項を作って $\vee$ をとればいい.

- 論理式 A の命題変数を $B_1, ..., B_m$ とする
- A が t となる組合せ $(B_1, ..., B_m)$ が n 通りある場合, n この論理積項を作る.
- i 番目の組合せの B_j が t の場合は B_j を, f の場合は $\neg B_j$ をリテラルとして i 番目の論理積項を作る.

以下に例を挙げる.
$\neg(A \Rightarrow (B \wedge C))$ の論理和標準形を求める. 真理値表は以下のようになる.

表 2.3 論理和標準形における$\neg(A \Rightarrow (B \wedge C))$

A	B	C	$B \wedge C$	$A \Rightarrow (B \wedge C)$	$\neg(A \Rightarrow (B \wedge C))$	
f	f	f	f	t	f	
f	f	t	f	t	f	
f	t	f	f	t	f	
f	t	t	f	t	f	
t	f	f	f	f	t	$A \wedge \neg B \wedge \neg C$
t	f	t	f	f	t	$A \wedge \neg B \wedge C$
t	t	f	f	f	t	$A \wedge B \wedge \neg C$
t	t	t	t	t	f	

ゆえに,

$$\neg(A \Rightarrow (B \wedge C)) \iff ((A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge \neg C))$$

論理回路の分野では, 項が最小となるように変形する. ゆえに $\neg C \vee \neg C = \mathrm{t}$ であることを用いて,

$$\neg(A \Rightarrow (B \wedge C)) \iff ((A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B))$$

とする.

同様にして論理積標準形についても考えてみる. 数学的に行う方法は 5. で $A \wedge (B \vee C)$ を $(A \vee B) \wedge (A \vee C)$ で置換する以外は論理和標準形のときと同じである. つまり,

1. $A \Leftrightarrow B$ を $(A \Rightarrow B) \wedge (B \Rightarrow A)$ で置換
2. $A \Rightarrow B$ を $\neg A \vee B$ で置換
3. $\neg(A \vee B)$ を $\neg A \wedge \neg B$ で, $\neg(A \wedge B)$ を $\neg A \vee \neg B$ で置換して, $\neg$ が $\wedge$ や $\vee$ の内側に現れるようにする (ド・モルガンの法則)
4. 偶数個の $\neg$ を除去 (二重否定の除去)
5. $A \wedge (B \vee C)$ を $(A \vee B) \vee (A \vee C)$ で置換して, $\wedge$ の内側の $\vee$ を外側に出す (分配律)
6. さらにトートロジーを用いて同値な論理式に変形してもよい

以下に例を挙げる.

$$\begin{aligned} \neg(A \Rightarrow (B \wedge C)) &\iff \neg(\neg A \vee (B \wedge C)) \\ &\iff \neg\neg A \wedge \neg(B \wedge C) \\ &\iff A \wedge (\neg B \vee \neg C) \qquad \text{(論理積標準形)} \end{aligned}$$

論理回路的な考え方としては, 以下のようである.

- A が f となる組合せの分だけ論理和項を作る
- 組合せの B_j の真理値が t の場合は $\neg B_j$ を, f の場合は B_j をリテラルとして論理和項を作る

f となる組合せの論理和標準形に否定を取ってあげると t となる組合せの論理式となる. ここで, ド・モルガンの法則から論理和標準形の否定は論理積の標準形となる. 以下に例を挙げる.
$\neg(A \Rightarrow (B \wedge C))$ の論理積標準形を求める. 真理値表は以下のようになる.

表 2.4 論理積標準形における$\neg(A \Rightarrow (B \wedge C))$ の真理値表

A	B	C	$B \wedge C$	$A \Rightarrow (B \wedge C)$	$\neg(A \Rightarrow (B \wedge C))$	
f	f	f	f	t	f	$A \vee B \vee C$
f	f	t	f	t	f	$A \vee B \vee \neg C$
f	t	f	f	t	f	$A \vee \neg B \vee C$
f	t	t	f	t	f	$A \vee \neg B \vee \neg C$
t	f	f	f	f	t	
t	f	t	f	f	t	
t	t	f	f	f	t	
t	t	t	t	t	f	$\neg A \vee \neg B \vee \neg C$

ゆえに,

$$\begin{aligned} &\neg(A \Rightarrow (B \wedge C)) \\ \iff &((A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg C)) \end{aligned}$$

2.2.5 論理記号の縮約

標準形では論理記号を 3 つだけ使う $(\wedge, \vee, \neg)$. ここでド・モルガンの法則を用いれば, 論理記号は 2 つで済む ($\wedge$ と $\neg$, または $\vee$ と $\neg$). 同様に, $\Rightarrow$ と $\neg$ の 2 つの論理記号で済ませることも出来る. さらに 1 つの記号で全ての論理式を表現することも可能である. それが NAND と NOR である. 論理回路の観点から見れ

ば複数の素子を集めるより, 一つの素子だけで回路を構成するほうがコストがかからずに済む (その分, 回路が冗長になるので遅延が発生する可能性がある).

- Sheffer stroke function(**NAND**)
  $A|B = \neg(A \wedge B)$
- Peirce function(**NOR**)
  $A \downarrow B = \neg(A \vee B)$

NAND 回路だけで NOT,AND,OR を構成する.
NOT について

$$\begin{aligned} \neg A &\iff \neg(A \wedge A) \\ &\iff (A|A) \end{aligned}$$

となるので,

$$\neg A \iff (A|A) \tag{2.5}$$

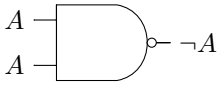


図 2.2 NAND 回路で構成した NOT 回路

AND について

$$\begin{aligned} A \wedge B &\iff \neg(\neg(A \wedge B)) \\ &\iff \neg(\neg(A \wedge B) \wedge \neg(A \wedge B)) \\ &\iff ((A|B)|(A|B)) \end{aligned}$$

となるので,

$$A \wedge B \iff ((A|B)|(A|B)) \tag{2.6}$$

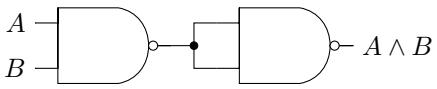


図 2.3 NAND 回路で構成した AND 回路

OR について

$$\begin{aligned} A \vee B &\iff \neg(\neg(A \vee B)) \\ &\iff \neg(\neg A \wedge \neg B) \\ &\iff \neg(\neg(A \wedge A) \wedge \neg(B \wedge B)) \\ &\iff ((A|A)|(B|B)) \end{aligned}$$

となるので,

$$A \vee B \iff ((A|A)|(B|B)) \tag{2.7}$$

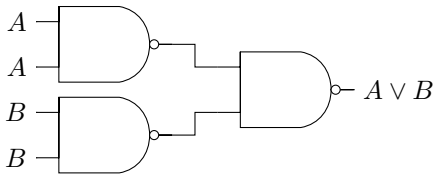


図 2.4 NAND 回路で構成した OR 回路

NOR 回路だけで NOT,AND,OR を構成する.
NOT について

$$\begin{aligned}\neg A &\iff \neg A \land \neg A \\ &\iff \neg(A \lor A) \\ &\iff A \downarrow A\end{aligned}$$

となるので,

$$\neg A \iff A \downarrow A \tag{2.8}$$

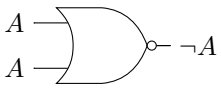


図 2.5 NOR 回路で構成した NOT 回路

AND について

$$\begin{aligned}A \land B &\iff \neg(\neg(A \land B)) \\ &\iff \neg(\neg A \lor \neg B) \\ &\iff \neg((\neg A \land \neg A) \lor (\neg B \land \neg B)) \\ &\iff \neg(\neg(A \lor A) \land \neg(B \lor B)) \\ &\iff (A \downarrow A) \downarrow (B \downarrow B)\end{aligned}$$

となるので,

$$A \land B \iff (A \downarrow A) \downarrow (B \downarrow B) \tag{2.9}$$

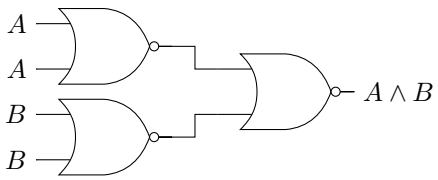


図 2.6 NOR 回路で構成した AND 回路

OR について

$$\begin{aligned}A \lor B &\iff (A \lor B) \land (A \lor B) \\ &\iff \neg(\neg((A \lor B) \land (A \lor B))) \\ &\iff \neg(\neg(A \lor B) \lor \neg(A \lor B)) \\ &\iff (A \downarrow B) \downarrow (A \downarrow B)\end{aligned}$$

となるので,

$$A \vee B \Longleftrightarrow (A \downarrow B) \downarrow (A \downarrow B) \tag{2.10}$$

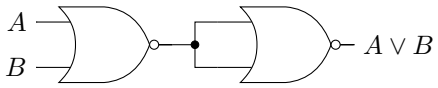


図 2.7 NOR 回路で構成した OR 回路

2.3 述語論理

2.3.1 命題関数

命題とは真か偽か定まっている言明であり, 命題変数とは真理値を表す変数で t か f の値をとるものであった. ここでは命題関数について扱う. **命題関数**とは変数 (**対象変数**) を含む “命題” のことである. 例を以下に挙げる.

- $x \leq 0$
- n は素数である

命題関数 $P : D \to T$ の定義域 D を**対象領域** (domain) という. 例を以下に挙げる.

- $T = \{\mathrm{t}, \mathrm{f}\}$
- $P : D \times D \to T$ 2 変数 (引数) の命題関数

D の要素を**対象**, D の定数を**対象定数**という.

2.3.2 述語論理式

命題関数の具体的な表現のことを**述語**という. 例を以下に挙げる.

- $Le(x, 0) : x \leq 0$, 2 変数 (引数)
- $Prime(n) : n$ は素数である

ここで, Le や $Prime$ の部分を**述語記号**という.
すべての, 任意のなどの意味を持つ記号を**全称記号**といい, $\forall$ で表す. 例えば, $\forall x P(x)$ と表記すれば, 全ての $x \in D$ に対して $P(x)$ が真という意味になる. これに対してある $\sim$ が存在するという意味を持つ記号を**存在記号**といい, $\exists$ で表す. 例えば $\exists y\, Q(y)$ と表記すれば, ある $y \in D$ が存在して $Q(y)$ が真という意味になる. この $\forall$ と $\exists$ を合わせて**量化記号**という.

述語, 論理記号, 量化記号の組合せで表される論理式を**述語論理式**という. 述語は述語論理式であり, この形を**原子論理式**という. また A, B が述語論理式なら $(A \vee B), (A \wedge B), (\neg A), (A \Rightarrow B)$ も述語論理式である.
x が対象変数で, A が述語論理式なら $(\forall x\, A), (\exists x\, A)$ も述語論理式となる. 例を以下に挙げる.
対象領域 D を自然数全体 $\mathbb{N}$ とする.

- $\forall z\, (x = y \times z)$
- $\neg(\exists n \exists x \exists y \exists z\, (n \geq 3 \wedge x \geq 1 \wedge y \geq 1 \wedge z \geq 1 \wedge x^n + y^n = z^n))$
- $\forall n \forall x \forall y \forall z\, (n \leq 2 \vee x \leq 0 \vee y \leq 0 \vee z \leq 0 \vee x^n + y^n \neq z^n)$

2.3.3 自由変数と束縛変数

変数について, $\forall x \exists y\,(x < y)$ の変数 x, y は具体的な値を代入する対象にはならない. 全ての x について $\sim\sim$ となる y が存在するというのは具体的に値を一つ定めることは不可能であるためである. またこのような変数は $\forall z \exists y\,(z < y)$ としても意味が変わらない. このような “変数” のことを**束縛変数**という. 一方で代入できる変数を**自由変数**という. 自由変数には $\forall, \exists$ などの記号を併用してはならない. 以下に例を挙げる.

対象領域 D を自然数全体 $\mathbb{N}$ とする.

- $\exists z\,(x = y \times z)$: x は y の倍数である
  このとき x, y が自由変数であり, z が束縛変数となる.

- $\exists z\,(x = z \times z)$: x は平方数である
  このとき x が自由変数であり, z が束縛変数となる.

- $\exists x \exists y \exists z\,(n \geq 3, \wedge x \geq 1 \wedge y \geq 1 \wedge z \geq 1 \wedge x^n + y^n = z^n)$

自由変数を含む述語論理式は真偽が決まらない. 例えば x は平方数である $(\exists z(x = z \times z))$ という述語論理式に関しては x の値が 4 の場合は満たされるのに対し, x の値が 3 であると不適となる. 一方で, 自由変数を含まない論理式は真偽が定まり, このような論理式を**閉論理式**という. 論理式 A に含まれる自由変数 x に t を代入して得られる論理式を $A[t/x]$ とかく.

自由変数と束縛変数について扱ってきたが, 自由変数と束縛変数の両方の意味をもつ変数が存在する場合はある. たとえば

$$\exists y\,(x = y + y) \wedge \exists z\,(y = z + z + 1)$$

とすると $\wedge$ の前では y は束縛変数である. 一方で, $\wedge$ の後では y は自由変数となっている. 出現単位で自由・束縛を区別する必要性がでてくる. また自由変数を代入するときに問題となる. ここで, 先に扱ったように束縛変数は変数の文字を変えても意味が変わらない. 束縛変数をうまく名前を書き換えることで解決させる. 以下に例を挙げる.

$$\exists z\,(x = y \times z)\ :\ x \text{ は } y \text{ の倍数である}$$

この自由変数 y に対して z を代入して x は z の倍数であるとしたい. しかしこの状態で z を代入すると

$$\exists z\,(x = z \times z)\ :\ x \text{ は平方数である}$$

となるので, 意味が変わってしまう. ゆえに代入する前に束縛変数の名前を書き換えることで解消する. 束縛変数の z を w に書き換える. すると

$$\exists w\,(x = y \times w)$$

となる. この後に y に z を代入すると

$$\exists w\,(x = z \times w)\ :\ x \text{ は } z \text{ の倍数である}$$

と代入した結果となる.

2.3.4 解釈 $\models$ 閉論理式

述語論理式の論理式の意味を考えるには, 対象領域を定める必要がある. 以下のように対応づけを行う. これを**解釈** (interpretation) という.

- 空でない集合 D を定める

- 対象定数に D の元を対応づける
- 対象変数は D を動く
- 関数記号に D 上の関数を対応づける
- 述語記号に D 上の述語を対応づける
  述語記号 “=” の解釈は通常の等号とする.

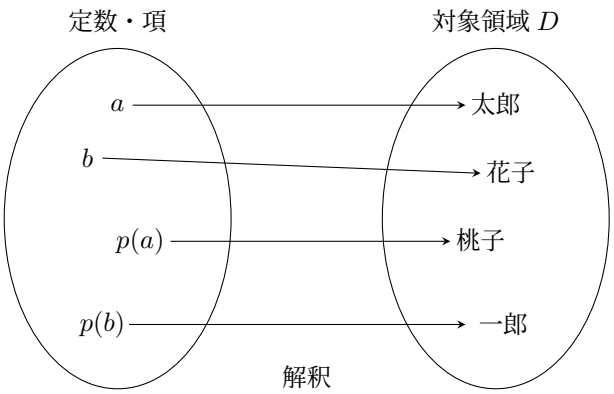


図 2.8 解釈の例

記号に具体的な要素・関数・述語を対応づける. 簡単にいえば, D を自然数全体 $\mathbb{N}$ とすれば, 変数や定数や関数, 記号はすべて自然数全体 $\mathbb{N}$ で考えるということである.
解釈について, もう少し数式化する. その上でまず**構造** (structure) というものを扱う. 構造とは対象領域 D と解釈 σ の対である. $\langle D, \sigma \rangle$ と表記する. 言語 L に対する構造 $\mu = \langle D, \sigma \rangle$ とは,

- D は空でない集合 (μ の対象領域という)
- σ は L の対象定数, 関数記号, 述語記号を D 上の要素, 関数, 述語に対応させる写像
  - c が対象定数のとき, $c^\sigma \in D$
  - f が n 変数の関数記号のとき, f^σ は D 上の n 変数の関数: $f^\sigma : D^n \to D$
  - P が n 変数の述語 (等号以外) のとき, P^σ は D 上の n 変数の述語: $P^\sigma \subseteq D^n$

である. これらを満たす σ のことを解釈という.
言語 L に対して $\mu = \langle D, \sigma \rangle$ の対象領域 D の要素を対象定数として追加したものを $L[\mu]$ という. $u \in D$ に対して, u は $L[u]$ の対象定数である. また, $u^\sigma = u$ となる. 以下, 構造の例を挙げる.
言語 L を以下とする.

- 対象定数 : a, b, c
- 関数記号 : f
- 述語記号 : $S(x), P(x), L(x, y)$

構造 $\mu = \langle D, \sigma \rangle$ を以下とする.

- $U =$ { 太郎, 一郎, 花子, 桃子 }
- 対象定数
  - $a^\sigma =$ 太郎
  - $b^\sigma =$ 花子
  - $c^\sigma =$ 桃子
- 関数記号
  - f^σ(太郎) = 一郎

- f^σ(一郎) = 太郎
- f^σ(花子) = 桃子
- f^σ(桃子) = 花子

- 述語記号
  - S^σ = { 太郎, 花子 }
  - P^σ = { 花子 }
  - L^σ = {(太郎, 花子), (桃子, 一郎), (花子, 太郎)}

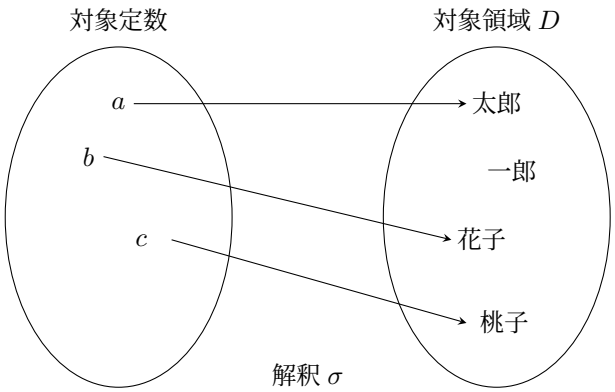


図 2.9 構造の例

変数を含まない $L[\mu]$ の項 t の構造 $\mu = \langle D, \sigma \rangle$ における意味づけ μ を次のように定義する.

1. t が対象定数 c のとき, $t^\mu = c^\sigma$
2. t が $f(t_1, ..., t_n)$ の形のとき, $t^\mu = f^\sigma(t_1^\mu, ..., t_n^\mu)$

例を以下に挙げる.
先の構造の例から $f(a)^\mu$ を求める.

$$f(a)^\mu \overset{2}{=} f^\sigma(a^\mu) \overset{1}{=} f^\sigma(a^\sigma) = f^\sigma(\text{太郎}) = \text{一郎}$$

解釈 $\models$ 閉論理式について扱う. このとき定義は以下である.

- $\mathcal{M} \models A$: 解釈 $\mathcal{M}$ において論理式 A が真
- $\mathcal{M} \not\models A$: 解釈 $\mathcal{M}$ において論理式 A が偽

また解釈 $\models$ 閉論理式において, 以下が満たされるべき性質である.

1. A が原子論理式 $P(t_1, t_2, ..., t_n)$ のとき, $\mathcal{M} \models P(t_1, t_2, ..., t_n)$ となるのは, P の解釈である命題関数に $t_1, t_2, ...t_n$ の解釈である元を入れたときに真になる場合である
2. $\mathcal{M} \models A \wedge B$ となるのは, $\mathcal{M} \models A$かつ$\mathcal{M} \models B$ のとき
3. $\mathcal{M} \models A \vee B$ となるのは, $\mathcal{M} \models A$または$\mathcal{M} \models B$ のとき
4. $\mathcal{M} \models A \Rightarrow B$ となるのは $\mathcal{M} \models A$ならば$\mathcal{M} \models B$ のとき
5. $\mathcal{M} \models \neg A$ となるのは $\mathcal{M} \not\models A$ のとき
6. $\mathcal{M} \models \forall x A$ となるのはすべてのD の元 u に対して $\mathcal{M} \models A[u/x]$ となるとき
7. $\mathcal{M} \models \exists x A$ となるのはあるD の元 u が存在して $\mathcal{M} \models A[u/x]$ となるとき
8. 自由変数を含む論理式 B について $\mathcal{M} \models B$ となるのは $\mathcal{M} \models B^C$ となるとき

ただし, B の自由変数を $x_1, x_2, ..., x_m$ として, $\forall x_1, \forall x_2, ..., \forall x_m B$ を B の**閉包**と呼び B^C とかく.

2.3.5 恒真な述語論理式

いかなる解釈に対しても真である述語論理式を**恒真**な述語論理式という. 例を以下に挙げる.

A は x を自由変数として含まないこととする (含んでしまう場合は閉論理式ではないため).

1. $\forall x\, A \Leftrightarrow A,\ \ \exists x\, A \Leftrightarrow A$
2. $\forall x\, B \Leftrightarrow \forall y(B[y/x]),\ \ \exists x\, B \Leftrightarrow \exists y(B[y/x])$
   ただし, y は B に含まれない新しい変数
3. $A \wedge \forall x\, B \Leftrightarrow \forall x(A \wedge B),\ \ A \wedge \exists x\, B \Leftrightarrow \exists x(A \wedge B)$
4. $A \vee \forall x\, B \Leftrightarrow \forall x(A \vee B),\ \ A \vee \exists x\, B \Leftrightarrow \exists x(A \vee B)$
5. $\forall x\, B \wedge \forall x\, C \Leftrightarrow \forall x(B \wedge C),\ \ \exists x\, B \vee \exists x\, C \Leftrightarrow \exists (B \vee C)$
6. $\forall x\, B \vee \forall x, C \Rightarrow \forall x(B \vee C),\ \ \exists x(B \wedge C) \Rightarrow \exists x\, B \wedge \exists x\, C$
   $\Leftrightarrow$ ではないことに注意.
7. $\forall x \forall y\, B \Leftrightarrow \forall y \forall x\, B,\ \ \exists x \exists y\, B \Leftrightarrow \exists y \exists x\, B$
8. $\exists x \forall y\, B \Rightarrow \forall y \exists x\, B$
   $\Leftarrow$ の反例 $(D = \mathbb{N})$: $\exists x \forall y(x > y)$ は偽, $\forall y \exists x(x > y)$ は真
9. $\forall x\, B \Rightarrow \exists x\, B$
10. $\neg \forall x\, B \Leftrightarrow \exists x \neg B,\ \ \neg \exists x\, B \Leftrightarrow \forall x \neg B$ (ド・モルガンの法則)
11. $(A \Rightarrow \forall x\, B) \Leftrightarrow \forall x(A \Rightarrow B),\ \ (A \Rightarrow \exists x\, B) \Leftrightarrow \exists x(A \Rightarrow B)$
12. $(\forall x\, B \Rightarrow A) \Leftrightarrow \exists x(B \Rightarrow A),\ \ (\exists x\, B \Rightarrow A) \Leftrightarrow \forall x(B \Rightarrow A)$
13. $\exists x(B \Rightarrow C) \Leftrightarrow (\forall x\, B \Rightarrow \exists x\, C)$
14. $\forall x(B \Rightarrow C) \Rightarrow (\forall x\, B \Rightarrow \forall x\, C)$
15. $\forall x(B \Rightarrow C) \Rightarrow (\exists x\, B \Rightarrow \exists x\, C)$

これらの例を数式ではなく日本語で理解するとイメージが楽である. 例を以下に挙げる.
$S(x) =$「x は学生である」, $T(x) =$「x は教員である」という二つの論理式を考える. $\Leftrightarrow$ が成り立たないケースを考える.

$$\exists x(S(x) \wedge T(x)) \Longrightarrow \exists x\, S(x) \wedge \exists x\, T(x)$$

これは, 日本語に直すと左が「学生であり教員である人が存在する」, 右が「学生である人が存在しかつ, 教員である人が存在する」となる. 学生であり教員である人が存在すれば右が成り立つことはわかる. また,

$$\exists x\, S(x) \wedge \exists x\, T(x) \Longrightarrow \exists x(S(x) \wedge T(x))$$

日本語に直すと「学生である人が存在して教員である人が存在すれば学生であり教員である人が存在する」となる. ここで, 左は学生である人と教員である人は異なっても構わないが, 右の場合は同一人物でなければいけないのでこれは必ずしも成立するとは限らないので $\Leftrightarrow$ が成り立たない.

つぎに, $M(x) =$「x は男である」, $F(x) =$「x は女である」という二つの論理式を考える.

$$\forall x\, M(x) \vee \forall x\, F(x) \Longrightarrow \forall x(M(x) \vee F(x))$$

これは, 日本語に直すと左が「全ての人は男であるまたは全ての人は女である」, 右が「全ての人は女であるかまたは男である」となる. 左は集団として男だけかまたは女だけの集団であることを言っている. 右は全ての人が個々で見て男か女であることを言っている (女と男以外はいないということを意味している). また,

$$\forall x(M(x) \vee F(x)) \Longrightarrow \forall x\, M(x) \vee \forall x\, F(x)$$

日本語に直すと「全ての人は女か男ならば全ての人は男または全ての人は女である」となる. 左の集団は男と女が共存しても構わないが, 右では許容されない. ゆえにこの論理式は成り立たないこととなるので, $\Leftrightarrow$

は成立しない.

最後に, $L(x,y)$ =「x は y が好き」という論理式を考える.

$$\exists x \forall y\, L(x,y) \Longrightarrow \forall y \exists x\, L(x,y)$$

これは日本語に直すと左は「ある人は全ての人が好きである」, 右は「全ての人が好きな人が存在する」となる. これは言い換えになっているだけで同じ意味であることがわかる. 一方で

$$\forall x \exists y\, L(x,y) \Longrightarrow \exists y \forall x\, L(x,y)$$

日本語に直すと「全ての人はある人が好きであるならば, ある人を全ての人が好きである」となる. 左は全ての人が好きである人が異なる可能性があるけど, 右は全ての人が同一人物を好きになっているので意味として異なるので $\Leftrightarrow$ が成り立たない.

恒真な述語論理式の証明を行っていく.

$$(\exists x\, B \Longrightarrow A) \Longleftrightarrow \forall x (B \Longrightarrow A)$$

この論理式に含まれる自由変数を $y_1, y_2, ..., y_m$ として, 任意の解釈 $\mathcal{M}$ について閉包

$$\forall y_1 \forall y_2 ... \forall y_m \{(\exists x\, B \Longrightarrow A) \Longleftrightarrow \forall x (B \Longrightarrow A)\}$$

を示せばよい.
解釈により次を示す.
すべての $u_1, u_2, ...u_m \in D$ について,

$$\begin{aligned} &\mathcal{M} \models (\exists x\, B \Longrightarrow A)[u_1/y_1, u_2/y_2, ..., u_m/y_m] \\ \Longleftrightarrow\ &\mathcal{M} \models \forall x\, (B \Longrightarrow A)[u_1/y_1, u_2/y_2, ..., u_m/y_m] \end{aligned}$$

以下では既に項の代入が済んでいるとする. $\mathcal{M} \models (\exists x\, B \Rightarrow A)$ と仮定すると
$\Rightarrow \mathcal{M} \models \exists x\, B$ ならば $\mathcal{M} \models A$
$\Rightarrow$ “ある $v \in D$ が存在して $\mathcal{M} \models B[v/x]$” ならば $\mathcal{M} \models A$
$\Rightarrow$“$\mathcal{M} \models B[v/x]$ を満たす $v \in D$ は不存在” または $\mathcal{M} \models A$
$\Rightarrow$“すべての $v \in D$ について $\mathcal{M} \not\models B[v/x]$” または $\mathcal{M} \models A$
A は x を自由変数として含まないから
$\Rightarrow$ すべての $v \in D$ につき “$\mathcal{M} \not\models B[v/x]$ または $\mathcal{M} \models A[v/x]$”
$\Rightarrow$ すべての $v \in D$ につき “$\mathcal{M} \models\models \neg B[v/x]$ または $\mathcal{M} \models A[v/x]$”
$\Rightarrow$ すべての $v \in D$ につき $\mathcal{M} \models (\neg B[v/x] \vee A[v/x])$
$\Rightarrow$ すべての $v \in D$ につき $\mathcal{M} \models (B[v/x] \Rightarrow A[v/x])$
$\Rightarrow \mathcal{M} \models \forall x (B \Rightarrow A)$

逆に $\mathcal{M} \models \forall x (B \Rightarrow A)$ と仮定すると
$\Rightarrow$ すべての $v \in D$ につき “$\mathcal{M} \models B[v/x]$ ならば $\mathcal{M} \models A[v/x]$”
A を x を自由変数として含まないから
$\Rightarrow$ すべての $v \in D$ につき “$\mathcal{M} \not\models B[v/x]$ または $\mathcal{M} \models A$”
$\Rightarrow$“すべての $v \in D$ につき $\mathcal{M} \not\models B[v/x]$” または $\mathcal{M} \models A$
$\Rightarrow$“$\mathcal{M} \models B[v/x]$ を満たす $v \in D$ は不存在” または $\mathcal{M} \models A$
$\Rightarrow$“ある $v \in D$ が存在して $\mathcal{M} \models B[v/x]$” ならば $\mathcal{M} \models A$
$\Rightarrow \mathcal{M} \models (\exists x\, B \Rightarrow A)$

つぎに以下の式が恒真であることを示す.

$$(A \Longrightarrow \forall x\, B) \Leftrightarrow \forall x (A \Longrightarrow B)$$

この論理式に含まれる自由変数を $y_1, y_2, ..., y_m$ として, 任意の解釈 $\mathcal{M}$ について閉包

$$\forall y_1 \forall y_2 ... \forall y_m \{(A \Longrightarrow \forall x\, B) \Longleftrightarrow \forall x (A \Longrightarrow B)\}$$

を示せばよい.
解釈により次を示す.
すべての $u_1, u_2, ...u_m \in D$ について,

$$\mathcal{M} \models (A \Longrightarrow \forall x\, B)[u_1/y_1, u_2/y_2, ..., u_m/y_m] \Longleftrightarrow \mathcal{M} \models \forall x (A \Longrightarrow B)[u_1/y_1, u_2/y_2, ..., u_m/y_m]$$

以下では既に項の代入が済んでいるとする.
$\mathcal{M} \models (A \Rightarrow \forall x\, B)$ と仮定すると
$\Rightarrow \mathcal{M} \models A$ ならば $\mathcal{M} \models \forall x\, B$
$\Rightarrow \mathcal{M} \models A$ ならばすべての $v \in D$ について $\mathcal{M} \models B[v/x]$
A は x を自由変数として含まないから
$\Rightarrow$ すべての $v \in D$ について $\mathcal{M} \models A[v/x]$ ならば $\mathcal{M} \models B[v/x]$
$\Rightarrow \mathcal{M} \models \forall x (A \Rightarrow B)$
逆はこれをたどればよい.

2.3.6　冠頭標準形

ある解釈に対して真である述語論理式を**充足可能**な述語論理式という.(先にも扱った内容であるが, 解釈の概念が含まれるので自由変数や束縛変数の扱いが可能となる). 充足可能でない論理式を**充足不可能**という.
全称記号 $\forall$ と存在記号 $\exists$ が, 他の論理記号の外側にあるとき, **冠頭論理式**という. 任意の論理式と同値な冠頭論理式が存在する. これを**冠頭標準形**という.

冠頭標準形の変形を考える. 恒真な論理式の例の 3,4,10,11,12 を使って同値変形する. ただし A が自由変数を x を含むときは, まず 2 により x を新しい変数で置き換えておく必要がある. 以下に例を挙げる.

$$\begin{aligned}
&\neg\exists y\, P(x,y) \wedge \forall y \{\exists z\, Q(x,y,z) \Rightarrow R(x,y)\} \\
\Longrightarrow\ &\neg\exists y\, P(x,y) \wedge \forall w \{\exists\, Q(x,w,z) \Rightarrow R(x,w)\} \\
\Longrightarrow\ &\forall y \neg P(x,y) \wedge \forall w \{\exists z\, Q(x,w,z) \Rightarrow R(x,w)\} \\
\Longrightarrow\ &\forall y \forall w \{\neg P(x,y) \wedge (\exists z\, Q(x,w,z) \Rightarrow R(x,w))\} \\
\Longrightarrow\ &\forall y \forall w \{\neg P(x,y) \wedge \forall z (Q(x,w,z) \Rightarrow R(x,w))\} \\
\Longrightarrow\ &\forall y \forall w \forall z \{\neg P(x,y) \wedge (Q(x,w,z) \Rightarrow R(x,w))\} \\
&\text{さらに論理和標準形にすると} \\
\Longrightarrow\ &\forall y \forall w \forall z \{\neg P(x,y) \wedge (\neg Q(x,w,z) \vee R(x,w))\} \\
\Longrightarrow\ &\forall y \forall w \forall z \{(\neg P(x,y) \wedge \neg Q(x,w,z) \vee (\neg P(x,y) \wedge R(x,w))\}
\end{aligned}$$

3　順序機械

3.1　オートマトン

3.1.1　オートマトンとは

オートマトンは状態を持つ機械のことである. コンピュータに限らずさまざまな計算システムの計算モデルとして用いられる. 最も重要な計算モデルの 1 つである. 応用範囲としては計算機システムの解析と設計, 文字列検索, 音声認識 (確率オートマトン), 離散事象システム (データベースシステムなど) の解析, 遺伝子の配列解析 (確率オートマトン), 生物の発生モデル (セルオートマトン), 数値シミュレーション (セルオートマトン) 等がある.

自動販売機で 150 円の品物を販売している場合を考える．この場合 100 円を入れた後 50 円を入れるとおつりなしで品物が出てくる．一方で 100 円入れた後更に 100 円をいれると合計が 200 円となるので品物が出た後におつりが 50 円返ってくる．一方で 50 円,50 円,50 円という払い方もできる．これをオートマトンで表現すると以下のようになる．

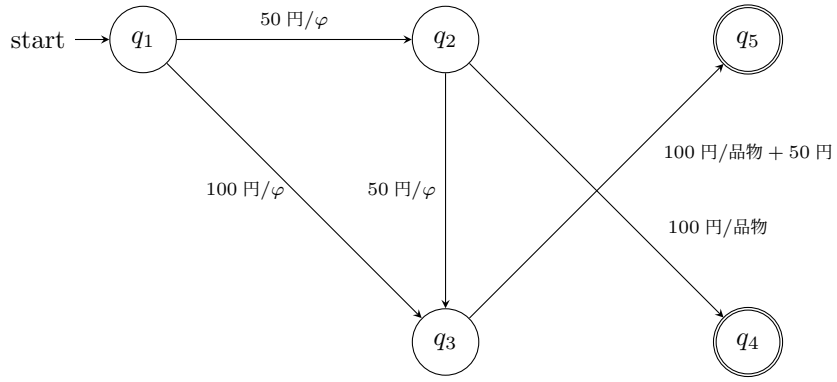


図 3.1 自動販売機のオートマトン

オートマトンは文字列検索へも応用されている．たとえば aba を検索するオートマトンは以下のように表現することができる．

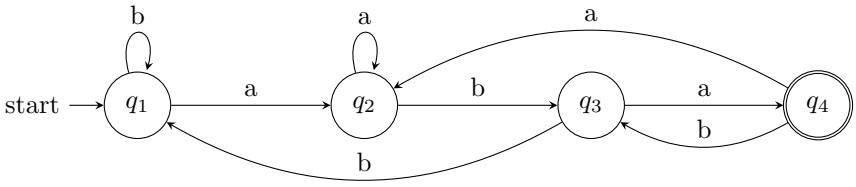


図 3.2 aba を検索するオートマトン

図 3.2 を C++ で実装すると以下のようになる．

```
#include<bits/stdc++.h>
using namespace std;

struct Node {
  bool accept=false;
  Node *a_next, *b_next;
};

Node *p_1,*p_2,*p_3,*p_4;

int main() {
  int count = 0;
  string str;
  Node *point;

  //memory allocation
  p_1 = new Node;
  p_2 = new Node;
  p_3 = new Node;
  p_4 = new Node;

  //p_4 is accepting
  p_4->accept = true;

  //node assignment
  p_1->a_next = p_2;
  p_1->b_next = p_1;
  p_2->a_next = p_2;
  p_2->b_next = p_3;
  p_3->a_next = p_4;
  p_3->b_next = p_1;
  p_4->a_next = p_2;
  p_4->b_next = p_3;
```

```
34
35    point = p_1;
36
37    cin >> str;
38
39    for(int i = 0; i < str.length(); i++){
40      switch(str[i]){
41      case 'a':
42        point = point->a_next;
43        break;
44      case 'b':
45        point = point->b_next;
46        break;
47      default:
48        point = p_1;
49        break;
50      }
51      if(point==p_4) count++;
52    }
53
54    cout << "The␣total␣number␣of␣aba␣is␣" << count << endl;
55
56    return 0;
57  }
```

図 3.3 aba を検索するオートマトンの実装例

図 3.3 を実行すると様々なパターンで実行すると以下のようになる.

```
case 1:
aabbababaaba
The total number of aba is 3

case 2:
abababababa
The total number of aba is 5
```

このようになり,aba がカウントされていることが分かる.

3.1.2 形式的記述

オートマトンを扱う上で形式的記述について扱う. オートマトンの分野で扱われる言語は**形式言語**と呼ばれるものである. 日本語, 英語, フランス語などの**自然言語** (matual language), および, FORTRAN, Pascal, C, Java などの**プログラミング言語** (programming language) に対し, それらを抽象化して定められた言語が**形式言語** (formal language), あるいは**数理言語** (mathmatical language) とよばれるものである. 形式言語を考える場合にはまずその分を構成するための最小単位要素の領域を定め, それを抽象的に記号の集合 Σ で表す. 簡単にいえば Σ はアルファベットである. この Σ は有限集合である. これは, たとえば英語において, その文を構成する最小単位を単語として考えたとき, 各単語を 1 記号とみなし, 使用可能とする全単語をあらかじめ指定しておくことに相当する. そう考えると**文** (sentence) はそのような記号から構成される記号列とみなすことができる. さらに, 言語はそのような記号列のうちで特定の条件を満足するようなものだけの集まりである. これは英語においては, 英文法的に正しい文だけの集まり全体を “英語” とみなすことに相当する.
形式的に Σ はアルファベットであるので, 以下のように表す.

$$\Sigma = \{a_1, a_2, ..., a_m\} \quad (a_1, a_2, ..., a_m \text{の } m \text{ 個の記号からなる集合}) \tag{3.1}$$

また, Σ 中の記号を重複を許して有限個並べて得られる

$$w = a_{i_1} a_{i_2} \cdots a_{i_n} \tag{3.2}$$

を, Σ 上の**記号列** (string over Σ) あるいは**系列** (sequence over Σ) という. この記号列 w の**長さ** (length) は, w を構成している記号の個数 n であると定義し, $|w|$ で表す. つまり, $|w| = n$ である. 特に, 長さが

$0(n=0)$ の記号列を**空記号列** (empty string) あるいは**空系列** (empty sequence) とよび, ε(epsilon) で表す. たとえば, $w=01011$ は $\Sigma=\{0,1\}$ 上の記号列であり, $|w|=5$ である. 記号列 x と y に対して, 記法 xy は x と y の**連接** (concatenation) とよばれる演算結果を表し,

$$x=a_{i_1}\cdots a_{i_s},\quad y=b_{i_1}\cdots b_{i_t}\ \text{のとき},\quad xy=a_{i_1}\cdots a_{i_s}b_{i_1}\cdots b_{i_t} \tag{3.3}$$

で定義される. ゆえに, xy は記号列 x の後に記号列 y をつなぎ合わせてできる記号列を表す.
記号列 $w=xyz$ において, x を w の**接頭辞** (prefix), z を**接尾辞** (suffix) という. また, x,y,z はおのおの w の**部分記号列** (substring) であるという. ここで, x,y,z は空記号列 ε であってもよく, $\varepsilon yz=yz, x\varepsilon z=xz, xy\varepsilon=xy$ である. 特に, $yz\neq\varepsilon$ であるとき, x は $w=xyz$ の**真の** (proper) 接頭辞である, などという.
$w=a_{i_1}a_{i_2}\cdots a_{i_n}$ において $a_{i_1}=a_{i_2}=\cdots=a_{i_n}=a$, つまり, w は同じ記号 a が n 個並べられたものであるとき, $w=a^n$ と表すこともある. 空記号列 ε を含めて, Σ 上で作り得るあらゆる記号列の全体から成る無限集合 Σ^* と表し, これを Σ の**スター閉包** (star closure) とよぶ. 以下に例を挙げる.
$\Sigma=\{0,1\}$ のとき

$$\Sigma^*=\{\varepsilon,0,1,00,01,10,11,000,001,010,011,100,101,110,111,0000,0001,...\}$$

また, $\Sigma=\{a,b,c\}$ (a,b,c の 3 記号からなる集合のとき

$$\Sigma^*=\{\varepsilon,a,b,c,aa,ab,ac,ba,bb,bc,ca,cb,cc,aaa,aab,aac,aba,abb,abc,aca,acb,acc,baa,bab,...\}$$

以上の記法を用いると, Σ^* の中からある特定の条件を満足する記号列だけを集めた集合 L, すなわち Σ^* のある特定の部分集合 L が形式言語理論における言語であり, これを Σ 上の言語 (language over Σ) という.

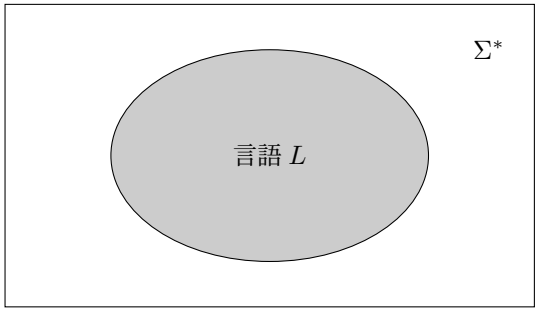


図 3.4 Σ 上の言語 L

Σ 上の言語 L は一般には無限集合であり, これを有限記述で Σ^* 中から規定するための 1 つの方法として, 形式文法とよばれる文法が用いられる. すなわち, S の文法規則に従うと, ちょうど L 内の文だけが正しい文として生成されるような文法を指定することにより, 言語 L が規定される. また, 言語 L を規定するためのもう 1 つの方法として, オートマトンによる方法がある. そのためには, L 中の記号列に対してだけ, それを入力しつくした後に出力 “1(yes. 受理)” の応答を出すオートマトンを指定すればよい.
このようにして, オートマトンと文法とは言語を介して密接な関連をもつ. なお, Σ 上の言語をオートマトン側で考える場合には, Σ は終端記号の有限集合と呼ばれるものとなる. このように, 入力記号の有限集合, 出力記号の有限集合, 終端記号の有限集合などのように, 一般にある指定された 1 個以上の記号から成る有限集合は**アルファベット** (alphabet) とよばれる. また, アルファベット上の記号列は**語** (word) と呼ばれることもある. 形式言語は, その構造に従って, 正規 (正則) 言語, 文脈自由言語, 文脈依存言語, 句構造言語と階層的に分類される. ここで, 前者の言語 (集合) は後者の言語の特別な場合となっていることが知られている.

3.1.3 順序機械

過去の一定の範囲内の入力に応じた状況を記憶することができ, その記憶内容と新たな入力との組み合わせによって, 次の動作が決まるような機械を, **順序機械** (sequential machine) という. ここで, 記憶できる内

容はあらかじめ定められた有限種類の範囲内のものであるが, 順序機械の能力として本質的であり, 機械がそれらの各記憶内容を保持しているとき, それに対応した**状態** (state) あるいは**内部状態** (internal state) にあるという. 特に, 機械の動作開始時点に設定される状態を**初期状態** (initial state) という. また入力により引き起こされる状態 (記憶内容) の変化を**状態遷移** (state transition) とよぶ.
形式的には, 機械のとり得る状態は保持できる記憶の種類だけの記号の有限集合 Q として与え, その具体的な意味づけは対象とするシステムに応じて定められる. **入力** (input) および**出力** (output) についても同様であり, おのおの考慮すべき入力および出力の種類だけの記号の有限集合 Σ(sigma), Δ(delta) として与える. 順序機械は, これらの中での状態推移および出力の仕方を指定することにより定められる.

3.1.4 Mealy 型順序機械

順序機械には 2 つの種類が存在する. それは Mealy(ミーリー) 型順序機械と Moore(ムーア) 型順序機械である. 自動販売機などの順序機械は Mealy 型である. また aba の単語を判断するオートマトンなど単語を認識する機械を Moore 型機械という. コンパイラにおいて, ユーザの書いたソースプログラムにおける字句 (トークン, token) を認識するための簡単な**字句解析器** (lexical analyzer) を実現する順序機械は Moore 型機械となる. Mealy 型順序機械は機械の出力が状態と入力との組み合わせによって決まる方式の順序機械である. 3 つの有限集合 Q, Σ, Δ と, これらの関係を定める 2 つの関係 δ(delta), λ(lambda), および特定の状態 q_0 を指定することにより定められるシステムである.

Q	状態の有限集合
Σ	入力記号の有限集合
Δ	出力記号の有限集合
$\delta : Q \times \Sigma \to Q$	状態遷移関数
$\lambda : Q \times \Sigma \to \Delta$	出力関数
$p_0 \in Q$	初期状態

状態遷移関数 (state transition function)δ とは機械の現在の状態 $p(\in Q)$ とそれへの入力 $a(\in \Sigma)$ のすべての組み合わせに対して, 次の時点の状態 (遷移先状態)$q(\in Q)$ を $\delta(p, a) = q$ により一意に定まる規則 (関数) のことである. 出力関数 (output function)λ とは機械の現在の状態 $p(\in Q)$ とそれへの入力 $a(\in \Sigma)$ のすべての組み合わせに対して, 次の状態へ遷移する間に出よくする出力記号 $b(\in \Delta)$ を $\lambda(p, a) = b$ により一意的に定める規則 (関数) のことである. このようにして定められた順序機械を M_e としたとき, これらをこの順に並べて形式的に,

$$M_e = (Q, \Sigma, \Delta, \delta, \lambda, q_0) \tag{3.4}$$

と表す. 以下に例を挙げる.
$M_e = (Q, \Sigma, \Delta, \delta, \lambda, p_0)$
$Q = \{p, q\},\ p_0 = p$
$\Sigma = \Delta = \{0, 1\}$
$\delta(p, 0) = p,\ \delta(p, 1) = q,\ \delta(q, 0) = p,\ \delta(q, 1) = q$
$\lambda(p, 0) = 0,\ \lambda(p, 1) = 0,\ \lambda(q, 0) = 0,\ \lambda(q, 1) = 1$
Mealy 型順序機械の挙動をより見やすく表現するための表のことを状態遷移表 (state transition table) という. これを先の例に従うと,

表 3.1　M_e の状態遷移表

現在の状態	次の状態		出力	
	入力		入力	
	0	1	0	1
→ p	p	q	0	0
q	p	q	0	1

状態遷移図をかくと

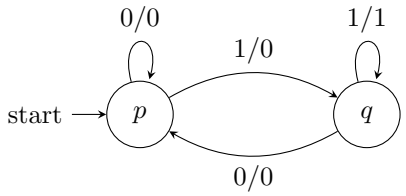


図 3.5　M_e の状態遷移図

$\delta(p,a)=q,\ \ \lambda(p,a)=b$ であるとき, 状態から p から状態 q へ向かう遷移を表す矢印を引き, その矢印には入出力を表すラベル a/b をつける.
この Mealy 型順序機械 M_e の動作例は以下のようである.

入力列	0	1	0	1	1	1	0	$\cdots$
状態列	p	p	q	p	q	q	q	p $\cdots$
出力列	0	0	0	0	1	1	0	$\cdots$

図 3.6　Mealy 型順序機械 M_e の動作例

3.1.5 Moore 型順序機械

Moore(ムーア) 型順序機械は, Mealy 型の場合のように出力が遷移途中で出されるのではなく, 遷移完了後にその遷移先状態だけによって決まる出力記号を出す方式の順序機械であり, 出力関数の他は, すべて Mealy 型順序機械の場合と同様のシステム

$$M_o=(Q,\Sigma,\Delta,\delta,\lambda,q_0) \tag{3.5}$$

で表す. この組は出力関数以外は基本的に Mealy 型機械と同じ意味を持つ.

Q	状態の有限集合
Σ	入力記号の有限集合
Δ	出力記号の有限集合
$\delta: Q\times\Sigma\to Q$	状態遷移関数
$\lambda: Q\times\Sigma\to\Delta$	出力関数 (状態に入るときに出力. 初期状態では出力しない)
$p_0\in Q$	初期状態

出力関数 (output function)λ は, 機械の遷移先の各状態 $p(\in Q)$ に対し, それのみに依存して出される出力記号 $b(\in\Delta)$ を $\lambda(p)=b$ により一意的に定める規則 (関数) である. したがって, 各状態に対し, それに入力される入力記号とは独立に出力は定められる. 特別な場合として, 初期状態 q_0 に対しては, 入力が加えられる以前に (言い換えれば, 空記号列 ε の入力が与えられたとき), あらかじめ定められた出力記号 $\lambda(q_0)$ が出される. (これに対し, Mealy 型順序機械では空記号列 (ε) の入力に対する出力はない.)

以下に例を挙げる.
$M_o = (Q, \Sigma, \Delta, \delta, \lambda, p_0)$
$Q = \{r, s, t\}, p_0 = r$
$\Sigma = \Delta = \{0, 1\}$
$\delta(r, 0) = r,\ \delta(r, 1) = s,\ \delta(s, 0) = r,\ \delta(s, 1) = t,\ \delta(t, 0) = r,\ \delta(t, 1) = t$
$\lambda(r) = 0,\ \lambda(s) = 0,\ \lambda(t) = 1$
状態遷移表について, Mealy 型と異なる点は出力が入力に依存していないところである.

表 3.2 M_o の状態遷移表

現在の状態	次の状態 入 0	次の状態 入 1	出力
→ r	r	s	0
s	r	t	0
t	r	t	1

また状態遷移図は次のようになる. Moore 型順序機械の状態遷移図においては, 状態 p に対する

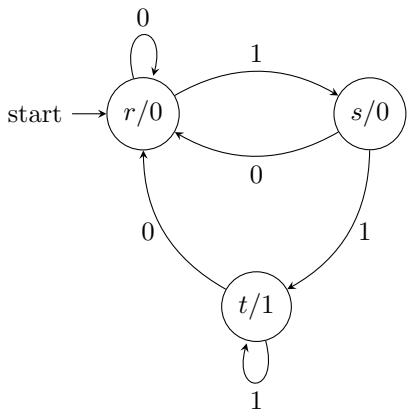


図 3.7 M_o の状態遷移図

出力が b であるとき, つまり $\lambda(p) = b$ であるとき, そのことを p/b を丸で囲ったもので表す. また, $\delta(p, a) = q,\ \lambda(p) = b,\ \lambda(q) = c$ であるとき, p/b かあ q/c に向かう矢印を引き, その矢印にはその遷移を引き起こす入力記号 a をラベルとしてつける.
この Moore 型順序機械 M_o の動作例は以下のようである.

入力列	(ε)	0	1	0	1	1	1	0	$\cdots$
状態列	r	r	s	r	s	t	t	r	$\cdots$
出力列		0	0	0	0	1	1	0	$\cdots$

図 3.8 Moore 型順序機械 M_o の動作例

3.1.6 Mealy 型順序機械と Moore 型順序機械の同等性

図 3.5 の順序機械 M_e と図 3.7 の順序機械 M_o は等価である. 出力のタイミングは半クロックずれる結果となる. たとえば, M_e と $_o$ に共通の入力 01101110$\cdots$ を加えたときの動作例を以下に示す.

入力列	(ε)	0	1	0	1	1	1	0	$\cdots$
M_e の状態列	p	p	q	p	q	q	q	p	$\cdots$
M_o の状態列	r	r	s	r	s	t	t	r	$\cdots$
M_e の出力列		0	0	0	0	1	1	0	$\cdots$
M_o の出力列		0	0	0	0	1	1	0	$\cdots$

図 3.9 M_e と M_o の動作例と同等性

(a) 与えられた Moore 型に対して, 変換機として等価な Mealy 型を構成することができる.
変換方法として, 与えられた Moore 型順序機械の各状態 p につき, それに対応する出力を出力記号を a としたとき, a が状態 p に至るすべての遷移の途中に出されるように出力を前にずらすことにより, ただちに対応する Mealy 型順序機械が得られる.

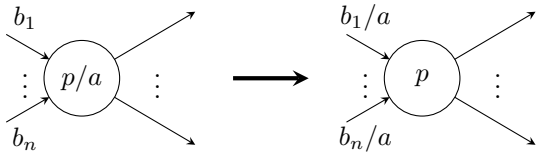


図 3.10 Moore 型から Mealy 型の変換

(b) 与えられた Mealy 型に対して, 変換機として等価な Moore 型を構成することができる.
簡単のために, 与えられた Mealy 型順序機械 M'_e において, その各状態へ遷移する途中に出される出力記号が, すべての遷移先状態ごとに同じとなっている場合を考える. このような場合は, 各状態 p に対し, そこへの推移途中に出される出力記号を b としたとき, b が遷移完了後の状態 p に対応して出されるように出力を後ろへずらすことにより, ただちに目的の Moore 型順序機械 M'_o が得られる.

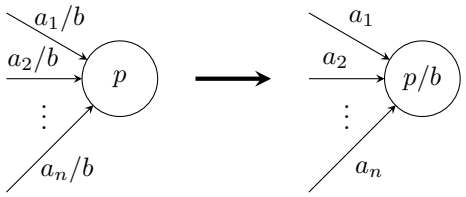


図 3.11 Mealy 型から Moore 型の基本的変換

以下のケースの場合は図 3.11 のような変換を行うことができる.

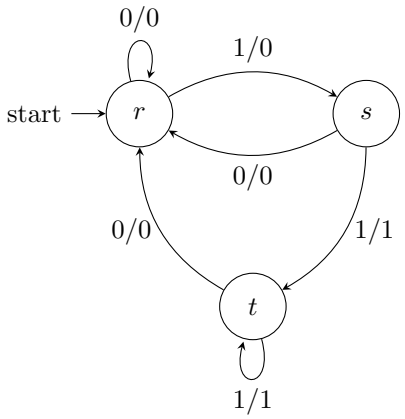


図 3.12 基本的変換で変換可能な Mealy 型順序機械

これの場合は図 3.7 のような Moore 型順序機械と等価となる.

一般的には, 任意の Mealy 型順序機械において, 遷移途中に出される出力記号が遷移先状態ごとに同じであるとの条件が満たされているわけではない. しかし, 任意の Mealy 型順序機械 M_e に対して, 適当な変形を行うことにより, M_e と全く同じ入出力応答を示す (等価な)Mealy 型順序機械 M'_e で, そのような条件を満たすものを得ることが可能である. 与えられた Mealy 型順序機械の M_e 中のある状態 p への遷移に対して, そこへ遷移する途中に出される出力記号が $b_1, b_2, ..., b_t$ の t 種類あったとする. このときには, 状態 p を $p_1, p_2, ..., p_t$ の t 個に分離し, 元の状態 p への遷移は, その途中に出されていた出力記号に応じ, それが b_i であったならば同じく b_i を出力して新しい状態 p_i へ至るようにと, 振り分けをおこなう. さらに, 元の状態 p から出ていた遷移は, 新しい状態 $p_1, p_2, ..., p_t$ のおのおのすべてからもまったく同様に出るようにする. このようにして得られる新しい Mealy 型順序機械の入出力応答は, もとの順序機械 M_e のものと同じになる. 以上に述べた変形操作を, 与えられた Mealy 型順序機械 M_e のすべての状態に対して適用することにより, 遷移途中に出力記号は遷移先状態ごとに同じで, かつ入出力応答はもとの M_e と全く同じ M'_e が得られる.

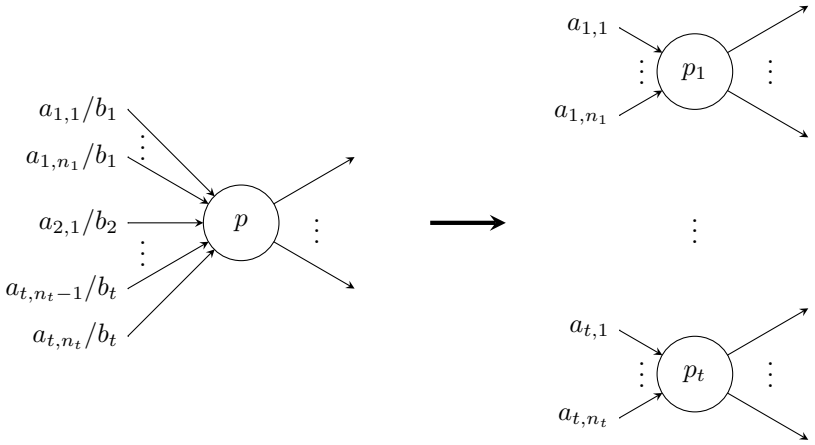


図 3.13 Mealy 型順序機械の変形

図 3.5 の Mealy 型順序機械 M_e を Moore 型順序機械に変換することを考える. 図 3.13 を元に変形することを考える. 状態 q へ遷移する途中に出される出力記号は 0 と 1 の 2 種類である. そこで, 状態 q を 2 個に分割し, 0 を出力して到達するほうの新しい状態を q_0, 1 を出力して到達するほうの新しい状態を q_0 とする. これを変換をすることで以下のようになることがわかる. ここで状態名を p, q_0, q_1 から r, s, t とすれば図 3.7 と同じ Moore 型順序機械が構成される.

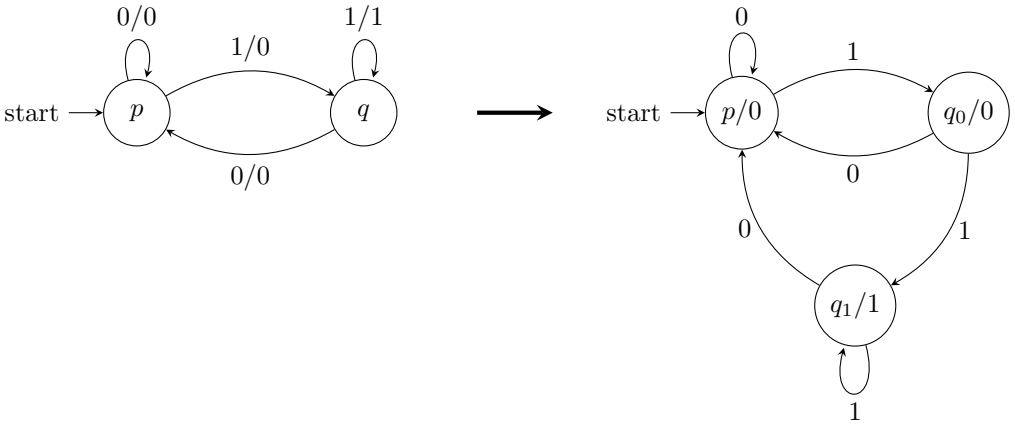


図 3.14 Mealy 型から Moore 型への変換例

3.1.7 順序機械の簡単化

順序機械について状態数を減らすことが可能である. 以下の 2 つの Mealy 型順序機械は等価であるが, 状態数が異なる.

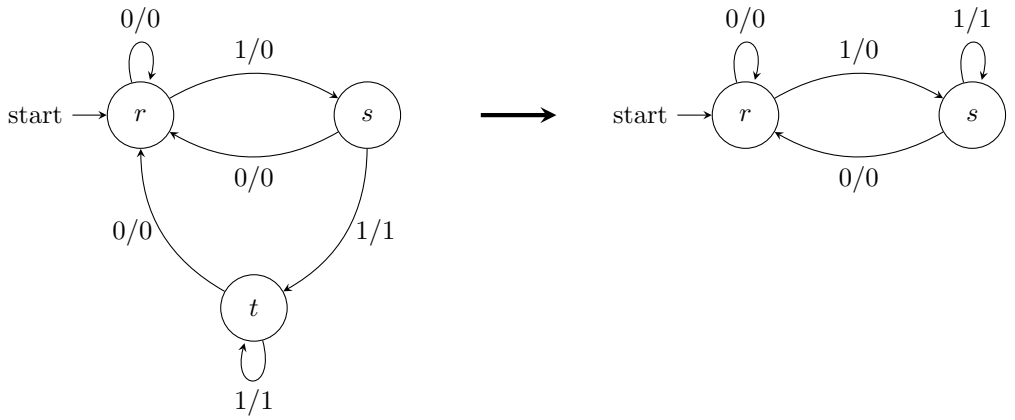


図 3.15 Mealy 型順序機械の簡単化

図 3.15 の左側を M'_e とし, 右側を M''_e とする. ここで, ある時点において M'_e は状態 s にあるとする. このとき入力 0 が加えられると, M'_e は出力 0 を出して状態 r へと遷移する. 一方, 状態 M'_e が状態 t にあるとして入力 0 が加えられた場合を考えると, M'_e は出力 0 を出して先の場合と同じ状態 r へと遷移する. したがって, M'_e が状態 s あるいは t にあり, そこで 0 で始まる任意の入力記号列が加えられたときの出力記号列は, いずれの場合に対しても全く同じとなる.

次に入力 1 が加えられたときを考察する. この場合も状態 s または t どちらにいても出力 1 が出されて同じ状態 t へと遷移する. したがって, M'_e が状態 s あるいは t にあるときに 1 で始まる任意の入力記号列が加えられたときの出力記号列は, いずれの場合に対してもまったく同じとなる. 対象としている入力記号は 0

か 1 である ($\Sigma = \{0,1\}$) ことから, 状態 s にいようが, 状態 t にいようが以降で加えられるいかなる入力記号列に対しても出力記号列は同じとなる. 一般に, 順序機械の 2 つの状態 s と t に対して, どのような入力記号列を加えても両者からはまったく同じ出力記号列が出されるとき, 2 つの状態は**等価** (equivalent) という. 等価性の判定は以下のようにしてできる.

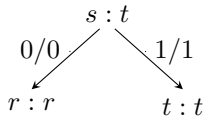


図 3.16 等価性の判定

このような等価な状態 s,t は機能的には全く区別する必要が無いので, 2 つの状態は統合することができる. すなわち, 一方の状態名 t を等価な状態名 s に付け替え, 両者をまったく同一化 (統合) することにより, もとの状態 t を省略することができる. こうして得られる順序機械は, 元のものと全く同じ入出力特性をもつ. したがって, 統合することによって等価な M_e'' が得られる. このように与えられた順序機械の変換する操作を, 順序機械の**簡単化** (simplification) という. 以上のような概念は, Moore 型順序機械の場合に対しても, 全く同様に適用することができる.

3.2 有限オートマトン

3.2.1 決定性有限オートマトン

順序機械は, 入力記号列を出力記号列へと変換するオートマトンであり, オートマトンのうちでも**変換器** (transducer) とよばれる部類に属するものである. ここで, 順序機械が出力する出力記号を特に 1(yes), 0(no) の 2 種類だけと限定し, この系として受け入れてよい入力記号列が入力され終わった時点では出力記号 1(yes) を出し, それ以外の入力記号列が入力された時点では出力記号 0(no) を出すようにすると, この順序機械は, 受け入れてもよい入力記号列だけを認識する (すなわち, 出力として 1(yes) を出す) **認識機械** (recognizer) とみなすことができる. これはある状態になったときに出力が 0 または 1 であるかを決定するので, Moore 型順序機械を用いる. ここで, 入力し終わった時点で出力が 1(yes) を出す状態へ到達させるような入力記号列はこの機械に**受理** (accept) されるという. また, 出力 1 を出す状態を**受理状態** (accepting state) あるいは**最終状態** (final state) とよぶ. なお, 順序機械への入力を開始する時点には, 機械は**初期状態** (initial state) とよぶ特定の状態に設定する. このようにして受理される入力記号列の集合は, この系に対応した 1 つの言語を定義することになる. このように認識機械としてみた場合の Moore 型順序機械は特に, **有限オートマトン** (finite automaton : **FA**), あるいは **有限状態オートマトン** (finite state automaton : **FSA**) とよばれる. ここで, この機械が記憶することのできる記憶の種類, すなわち状態の数は有限であることが, 有限(状態) オートマトンの名前の由来である. 有限オートマトンにおいては, 受理状態とそうでない状態 (非受理状態) とを区別することにより, 出力関数は明示しない. ゆえに形式的には, 有限オートマトンは, 2 つの有限集合 Q,Σ と, これらの関係を定める関数 δ, 特定の状態 q_0, および特定の状態の状態集合 F ををを指定することにより定まる 5 項組 (5-tuple) のシステム

$$M = (Q, \Sigma, \delta, q_0, F) \tag{3.6}$$

である.

Q	状態の有限集合
Σ	入力記号の有限集合
$\delta : Q \times \Sigma \to Q$	状態遷移関数
$p_0 \in Q$	初期状態
$F \subseteq Q$	受理状態の集合

これは, Moore 型順序機械の形式的定義において, 出力記号の有限集合を $\Delta = \{0, 1\}$ と固定して省略し, かつ, $\lambda(p) = 1$ となる出力を出す状態 p の集合を最終状態の集合 F, つまり $F = \{p \in Q \mid \lambda(p) = 1\}$ とし, 出力関数 λ を省略したものである. このような有限オートマトンにおいて現在の状態とそれへの入力に対してその遷移先状態は一意的に必ず定められているため, このことを強調するときには**決定性有限オートマトン** (deterministic finite automaton : **DFA**) という. 以下に例を挙げる.

$\Sigma = \{0, 1\}$

$Q = \{p, q, r\}$

$q_0 = p$: 初期状態は p

$\delta(p, 0) = p,\ \delta(p, 1) = q,\ \delta(q, 0) = p,\ \delta(q, 1) = r,\ \delta(r, 0) = p,\ \delta(r, 1) = r$

$F = \{r\}$

状態遷移表は最終状態 (受理状態) のところを二重丸で囲んで明示する. 以下のようになる.

表 3.3 決定性有限オートマトンの状態遷移表の例

状態＼入力		0	1
→	r	r	s
	s	r	t
	◎ t	r	t

状態遷移図に関しても最終状態 (受理状態) のところを二重丸で囲んで明示する. 以下のようになる.

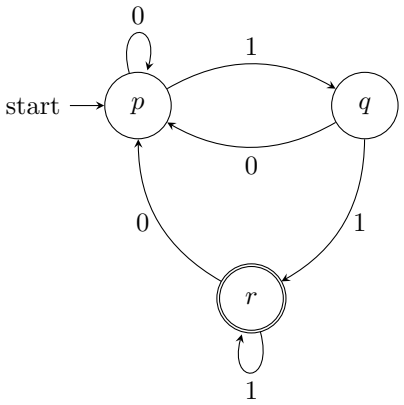


図 3.17 有限オートマトンの状態遷移図の例

この決定性有限オートマトンは 11 で終わる記号列を受理する. 以下が動作例である.

入力列		0	1	0	1	1
状態列	p	p	q	p	q	r

図 3.18 11 で終わる記号列を受理するオートマトンの動作例

有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ は, 最初にその状態を初期状態 q_0 に設定し, そこから入力記号の読み込みを開始する. 有限オートマトン M のある時点における状態が p で, そのときに読み込んだ入力記号が $a(\in \Sigma)$ であり, 状態遷移関数 δ において $\delta(p, a) = q$ であったとすると, M は次の時点で状態を p から q に一意的に変化させる. このことを, M が状態 p から状態 q へ入力記号 a により**状態遷移** (state

transition) するという. このことを,

$$p \xrightarrow[M]{a} q \tag{3.7}$$

記号列 $w = a_1 a_2 \cdots a_n \ \ (a_i \in \Sigma)$ について考える. このとき, $0 \leq i \leq n$ において $p_i \in Q$ であるとき,

$$p_0 \xrightarrow[M]{a_1} p_1, \ \ p_1 \xrightarrow[M]{a_2} p_2, \ \ \cdots, \ \ p_{n-1} \xrightarrow[M]{a_n} p_n$$

ならば, これを

$$p_0 \xrightarrow[M]{a_1} p_1 \xrightarrow[M]{a_2} \cdots \xrightarrow[M]{a_n} p_n \tag{3.8}$$

あるいは,

$$p_0 \xRightarrow[M]{w} p_n \tag{3.9}$$

と表す. ここで, 式 (3.9) を文字列の遷移であるため, 一回の遷移ではなくて複数の遷移であることを強調するために

$$p_0 \xRightarrow[M]{w}{}^{*} \ p_n \tag{3.10}$$

と表現することもある. またこれらの遷移は状態 p_0 から p_n への入力記号列 w による状態遷移という. これを状態遷移図で表すと

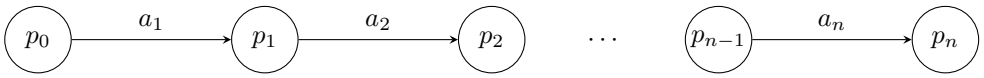


図 3.19 入力記号列 w の状態遷移の様子

図 3.17 について, 入力記号 0 で状態 p から状態 p へ遷移する. ゆえに,

$$p \xrightarrow[M]{0} p$$

となる. 同様にして,

$$p \xrightarrow[M]{1} q, \ \ q \xrightarrow[M]{0} p, \ \ q \xrightarrow[M]{1} r, \ \ r \xrightarrow[M]{0} p, \ \ r \xrightarrow[M]{1} r$$

である.
図 3.17 おいて受理する入力記号列か否かを以下に示す.
$p \xRightarrow[M]{11} r$ は, $r \in F$ より M は 11 を受理する. また, $p \xRightarrow[M]{011} r$ は $r \in F$ より M は 011 を受理する. しかし, $p \xRightarrow[M]{1001} q$ は $q \notin F$ より M は 1011 を受理しない.

状態遷移関数 δ を拡張した関数 $\hat{\delta}$(単に δ と表現することもある) というものがある. これは $\delta : Q \times \Sigma \to Q$ を定義していたものを $\hat{\delta} : Q \times \Sigma^* \to Q$ と定義したものである. つまり記号だけでなく記号列に対しても状態遷移を表現する. 状態 p から記号列 x 分進んだ先の状態は $\hat{\delta}(p, x)$ とかける. 定義は以下である.

$\forall p \in Q, \ \forall x \in \Sigma^*, \ \forall a \in \Sigma$ について

- $\hat{\delta}(p, \varepsilon) = p$
- $\hat{\delta}(p, xa) = \delta(\hat{\delta}(p, x), a)$

3.2.2 正則言語

$\hat{\delta} : Q \times \Sigma^* \to Q$ より決定性有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ は $\hat{\delta}(q_0, w)$ のとき $w \in \Sigma^*$ を受理する. ここで, 受理する言語と受理しない言語として M は Σ 上の記号列全体の集合 Σ^* を二つに分けるこ

とが出来る. $L(M)$ を M に受理される記号列全体の集合とし, $\Sigma^*\backslash L(M)$ を M に受理されない記号列全体の集合となる.
$M = (Q, \Sigma, \delta, q_0, F)$ を DFA とする. この M が受理する入力記号列全体の集合を $L(M)$, あるいは $L(q_0)$ で表し, M の**受理する言語** (language accepted (by M)) あるいは M の**認識する言語** (language recognized (by M)) という. つまり,

$$L(M) = L(q_0) = \left\{ x \in \Sigma^* \,\middle|\, q_0 \overset{x}{\underset{M}{\Longrightarrow}} r,\ r \in F \right\} \tag{3.11}$$

または, 拡張した状態遷移関数 $\hat{\delta}$ を用いて表すと,

$$L(M) = L(q_0) = \left\{ x \in \Sigma^* \,\middle|\, \hat{\delta}(q_0, x) \in F \right\} \tag{3.12}$$

である.
受理する有限オートマトンが存在する言語を**正則言語**または**正規言語** (regular language : **RL**), あるいは**有限状態言語** (finite state language) とよばれる.
状態 q_0 を一般化し, 任意の状態 p に対しても

$$L(p) = \left\{ y \in \Sigma^* \,\middle|\, p \overset{y}{\underset{M}{\Longrightarrow}} r,\ r \in F \right\} \tag{3.13}$$

とする. 正則言語に関して例を以下に挙げる.
次の状態遷移図によって有限オートマトンが受理する正則言語を考える.

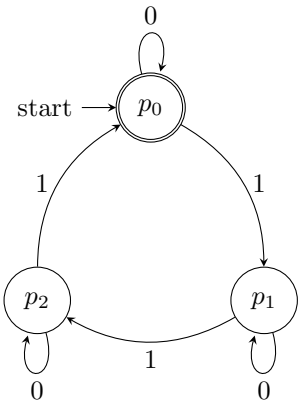


図 3.20　3 進カウンタ

図 3.20 で受理する言語は

$$\{\varepsilon, 0, 00, 111, 0111, 1011, 1101, 1110, 00111, ..., 111111, ...\}$$

となる無限集合である. これは “0 か 1 から成る入力記号列で, その中の 1 の個数が 3 の倍数 ($3n$, $n \geq 0$) であるようなもの全体の集合” である. したがって, この有限オートマトンは 3 進カウンタをモデル化したものである. ここで, 初期状態 q_0 は受理状態ともなっているので, 空入力記号列 ε も受理される.

M が DFA なら M の受理する言語 $L(M)$ は正則言語である. Σ 上の言語 (つまり Σ^* の部分集合)L は, ある DFA M が存在して $L = L(M)$ なら正則言語である. そのような M が存在しなければ L は正則言語ではないといえる. ある正則言語について受理する DFA M は無限に存在する. それは正則言語だから受理する DFA は少なくとも一つは存在することから初期状態からたどり着けない状態をいくらでも増やして異なる DFA をいくらでも作ることができるためである.

3.2.3 DFA の設計

$\Sigma = \{0, 1\}$ として, 奇数個の 1 を含む文字列のみを受理する DFA を設計することを考える. 順序機械の「記憶」は, 「今どの状態にいるか」ということしかない. 状態の数は有限であり, 1 文字ずつ読み込み状態遷移を行うため 1 の数を数えるわけにはいかない (数えられたところでその 1 の個数を数える DFA を設計することになる). 現在までに 1 が奇数個だったか偶数個だったかという状態を作る. ゆえに状態の集合を

$$Q = \{q_{\text{even}}, q_{\text{odd}}\}$$

とする.
状態遷移関数は, それぞれの状態である入力記号のときにどうすればよいかを考えて決めるので, それぞれの状態での遷移は以下のようになる.

- q_{even} : いままでの 1 の数が偶数.
  $\to$0 が来ても 1 の数は変わらないから q_{even} へ
  $\to$1 が来たら 1 の数が奇数個になるので q_{odd} へ
- q_{odd} : いままでの 1 の数が奇数.
  $\to$0 が来ても 1 の数は変わらないから q_{odd} へ
  $\to$1 が来たら 1 の数は偶数個になるので q_{even} へ

受理状態は 1 が奇数個入っていた場合だから $F = \{q_{\text{odd}}\}$ であり, 初期状態は 1 が 0 個 (偶数) だから, $q_0 = q_{\text{even}}$ である. まとめると

- $M = (\{q_{\text{even}}, q_{\text{odd}}\}, \{0, 1\}, \delta, q_{\text{even}}, \{q_{\text{odd}}\})$
- $\delta(q_{\text{even}}, 0) = q_{\text{even}}$, $\delta(q_{\text{even}}, 1) = q_{\text{odd}}$, $\delta(q_{\text{odd}}, 0) = q_{\text{odd}}$, $\delta(q_{\text{odd}}, 1) = q_{\text{even}}$

これによって設計された有限オートマトンの状態遷移図は以下のようになる.

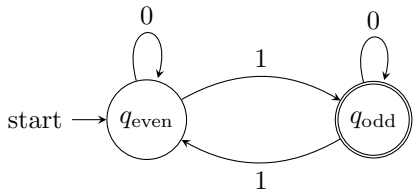


図 3.21 奇数個の 1 が含まれている入力列を受理する DFA

3.2.4 言語族の閉包性

言語の集合を**言語族**という. たとえば, 正則言語の集合を正則言語族という. 以下の式が成り立つとき, 言語演算 O に対して言語族 $\mathcal{F}$ が閉じているという.

$$L \in \mathcal{F} \Longrightarrow O(L) \in \mathcal{F} \qquad \text{(1 項演算の場合)} \tag{3.14}$$
$$L_1, L_2 \in \mathcal{F} \Longrightarrow O(L_1, L_2) \in \mathcal{F} \qquad \text{(2 項演算の場合)} \tag{3.15}$$

言語演算 O として, 様々な演算がある.
対称差演算

$$A \triangle B = (A - B) \cup (B - A) \tag{3.16}$$

連接演算 (concatenation)
Σ 上の記号列 x, y に対して, x の後に y をつなぎ合わせてできる記号列 $x \cdot y$ を x と y との**連接**という. 紛

らわしくない場合はには $x \cdot y$ を単に xy で表す. $x = 100$, $y = 101$ のとき, $xy = 100101$ である. 特に長さが 0 の記号列, つまり空記号列 ε に関しては, 任意の記号列 x に対して

$$x\varepsilon = \varepsilon x = x$$

となる. 連接の概念を Σ 上の言語に対して拡張すると, つまり L_1, L_2 を Σ 上の言語としたとき, $L_1 \cdot L_2$ あるいは単に L_1L_2 を, L_1 中の記号列と L_2 中の記号列とをこの順に連接してできる記号列全体の集合と定義し, これを L_1 と L_2 との連接という.

$$L_1L_2 = \{xy \,|\, x \in L_1, \text{かつ } y \in L_2\} \tag{3.17}$$

例を以下に挙げる.
$\Sigma = \{0, 1\}$, $L_1 = \{1, 01\}$, $L_2 = \{0, 11, 101\}$ とすると

$$L_1L_2 = \{10, 111, 1101, 010, 0111, 01101\}$$

となる.

クリーネ閉包 (Kleene closure)
同じ言語に対して連接の演算を無制限に任意の回数適用することで得られる閉包であり, 言語 L に対して L を n 回連接したものを L^n と表す. また

- $L^0 = \{\varepsilon\}$
- $n \geq 1$ である任意の n に対して $L^n = LL^{n-1}$

とする. すると L^* を L の**クリーネ閉包** (Kleene closure) または**スター閉包** (star closure), 単に閉包 (closure) といい, 以下の式で定義する.

$$L^* = \bigcup_{n=0}^{\infty} L^n = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \cdots \tag{3.18}$$

L^* というのは L に属する任意個の記号列を任意回数, 任意の順序で並べて得られる記号列のすべてから成る無限集合である. つまり記号の有限集合 Σ を考えた時, Σ^* は Σ 上のすべての記号列の集合を表す. 例を以下に挙げる.
$L = \{1, 00\}$ とすると
$L^2 = \{11, 100, 001, 0000\}$
$L^3 = \{111, 1100, 1001, 10000, 0011, 00100, 00001, 000000\}$
$L^* = \{\varepsilon, 1, 00, 11, 100, 001, 0000, 111, 1100, 1001, 10000, 0011, 00100, 00001, 000000, ...\}$

$L = \{ab, c\}$ とすると
$L^* = \{\varepsilon, ab, c, abab, abc, cab, cc, ababab, ababc, abcab, abcc, cabab, cabc, ccab, ccc, ...\}$

準同型写像 (homomorphism)
まず**同型写像** (isomorphism) とは単射であり全射である写像のことである. 一方で**準同型写像** (homomorphism) は複数の対象 (おもに代数系) に対して, それらの特定の数学的構造に関する類似性を表す概念で, 構造を保つ写像である. 集合 A, B と (A, B に対して定義された) 演算 $*$ があったとき, $f : A \to B$ が準同型であるとは, $\forall x \in A, \forall y \in A$ において

$$f(x * y) = f(x) * f(y) \tag{3.19}$$

が成り立つことである.
つまり, A の演算と B の演算とが集合上の写像 f のみで 1 対 1 に対応させることができる.

正則言語の閉包性に対して, 正則言語全体のなす族は, 補集合演算 (c) に関して閉じている. L を認識する DFA $M = (Q, \Sigma, \delta, q_0, F)$ の受理状態と非受理状態を入れ替えた DFA $M' = (Q, \Sigma, \delta, q_0, Q \backslash F)$ を考えると, $M' = (Q, \Sigma, \delta, q_0, Q \backslash F)$ を考えると, M' は L^c(L の補集合) を認識する. L は認識する DFA M が存在しているので正則言語であるとすると L^c は認識する DFA M' が存在するため正則言語であるといえる. これは Σ^* は $L(M)$ または $L(M')$ である. ここで $L = L(M)$ であるときには L^c は $L(M')$ になることがいえるので, 補集合演算に関して閉じているといえる.

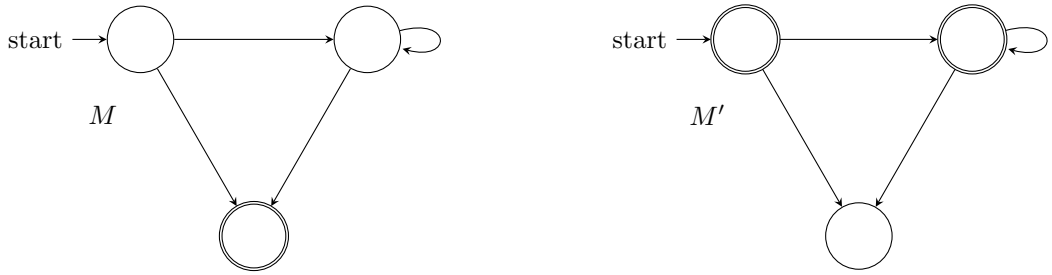


図 3.22 補集合演算における閉包性

正則言語は共通部分演算 $(\cap)$ に関しても閉じている.
$M = (Q, \Sigma, \delta, q_0, F)$, $M' = (Q', \Sigma, \delta', q_0', F')$ とする. $L(M) \cap L(M')$ を受理する DFA M^D をつくる. それで以下のように状態を定義すればよい.

- $M^D = (Q^D, \Sigma, \delta^D, q_0^D, F^D)$
- $Q^D = Q \times Q' = \{(q, q') \,|\, q \in Q, q' \in Q'\}$ (両 DFA の状態の組)
- $\delta((q, q'), c) = (\delta(q, c), \delta'(q', c)), \quad c \in \Sigma$ (それぞれの成分毎に遷移)
- $q_0^D = (q_0, q_0')$ (両 DFA の初期状態の組)
- $F^D = \{(q, q') \,|\, q \in F, q' \in F'\} \subseteq Q \times Q'$ (両方受理状態のとき)

それぞれの状態が 3 つずつであれば, 新しく作られる DFA の状態数は $3 \times 3 = 9$ である. また遷移数がそれぞれ m, n であれば, 新しく作られる DFA の遷移数は mn である.

3.2.5 等価性

$M = (Q, \Sigma, \delta, q_0, F)$, $M' = (Q', \Sigma', \delta', q_0', F')$ のおのおの受理する言語 $L(M)$, $L(M')$ が等しいとき, つまり $L(M) = L(M')$ であるとき, M と M' は**等価** (equivalent) であるといい

$$M \equiv M' \tag{3.20}$$

と表す. 一方で $L(M) \neq L(M')$ のとき, 一方には受理されるがもう一方には受理されない入力記号列が存在する. このとき M と M は等価でないといい

$$M \not\equiv M' \tag{3.21}$$

この等価性については有限オートマトン以外にも一般のオートマトンに対しても同様の概念をもつ. 有限オートマトンの透過性の概念を, 個々の状態に対しても一般化して用いる. つまり任意の 2 状態 p, q に対して $L(p) = L(q)$ であるとき, 2 つの状態は等価であるといい, $p \equiv q$ で表す. そうでないときは $p \not\equiv q$ と表す.

有限オートマトン $M = (Q, \Sigma, \delta, q_0, F)$ の状態 p に対して, 初期状態 q_0 から遷移してそこへ至る文字列が存在するとき, つまり適当な入力文字列 x に対して $q_0 \underset{M}{\overset{x}{\Longrightarrow}}$ であると, 状態 p は (初期状態から) **到達可能** (reachable) であるという. そうでない場合は到達不可能という. 以下に例を挙げる.

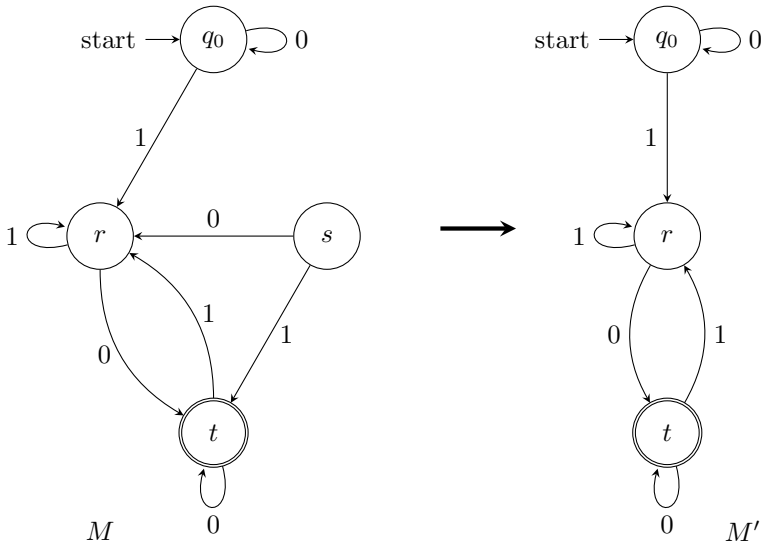


図 3.23 到達不可能状態 s の除去

到達不可能状態は有限オートマトンの動作にはまったく関与しない冗長なものであり, その状態およびそれから出る遷移を除いても元のオートマトンと等価である. 図 3.23 において, $M \equiv M'$ である. 特に与えられた有限オートマトンにおけるどの受理状態っも到達不可能状態であるときには, それの受理する言語は**空** (empty) であることになる.

到達不可能状態の除去について初期状態から逐次的に状態を遷移させていく. 新たな状態に遷移した場合その状態から移りうる状態を列挙する. 一旦出た状態に遷移する場合はそれ以上の列挙をやめる. 移りうる状態が一旦出た状態のみになったときに検出を終了する. 図 3.23 について, M の初期状態 q_0 から 0 が入力されると q_0 のままである. 1 が入力されると r に遷移する. つぎに r から 0 が入力されると t に遷移し, 1 が入力されると r のままである. つぎに t について 0 が入力されると t のままである. 1 が入力されると r に遷移する. これで新たに出てくる状態がなくなったので, 初期状態から遷移出来る状態は全て列挙できているので, 残された状態 s については到達不可能状態である. 以下, 図 3.23 における到達可能状態検出を図式化したものである.

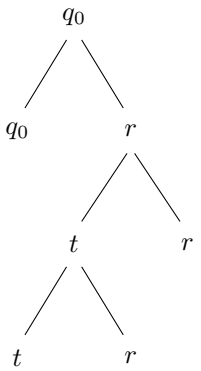


図 3.24 到達可能状態検出木

図 3.24 の手法によって, すべての到達可能状態を検出することができる. この正当性を以下に示す.

対象とする DFA を M, その初期状態を q_0, 入力記号集合を Σ としたとき, 任意の入力記号列 $x \in \Sigma^*$ に対して, q_0 から x による遷移先の状態は必ず 1 個が一意的に存在する. ここで, 次の命題 $\boldsymbol{E}(n)$ $(n \geq 0)$ が成立し, その結果として正当性が示される.

[命題 $\boldsymbol{E}(n)$] $|x| \leq n$ である任意の $x \in \Sigma^*$ に対して, もし,

$$q_0 \overset{x}{\underset{M}{\Longrightarrow}} p$$

(x により状態 p は q_0 より到達可能) であるならば, 到達可能状態検出木の構成が完了後, その中の節点としてラベルが p である節点が必ず存在する. 以下, 命題 $\boldsymbol{E}(n)$ を帰納法によって証明する.
$n = 0$ のとき, $p = q_0$ は到達可能状態検出器の根のラベルであり, $\boldsymbol{E}(0)$ の成立は自明である. つぎに $\boldsymbol{E}(k)$ $(k \geq 0)$ が成立すると仮定したとき, $\boldsymbol{E}(k+1)$ が成立することを証明する. ある任意の

$$y = xa \ \in \Sigma^* \ \ (|y| = k+1, \ a \in \Sigma)$$

を考える. $|x| = k$ であるので, $x \in \Sigma^*$ による遷移先の状態を p としたとき, 帰納法の仮定より, ラベルが p である節点は到達可能状態検出木の節点として存在する. このラベル p をもつ節点のうちで, 到達可能状態検出木の構成仮定で一番最初に出現した節点に着目すると, その節点は内部節点となっている. そこで, $\delta(p, a) = p'$ としたとき, 入力記号 a による遷移結果より, ラベルが p' である節点は, 到達可能状態検出木でラベルが p である内部節点の子節点としてとりいれられて存在している. 以上より, 命題 $\boldsymbol{E}(k)$ の前提のもとで, 命題 $\boldsymbol{E}(k+1)$ の成立が証明される. よって帰納法として任意の非負整数 n に対して, 命題 $\boldsymbol{E}(n)$ が成立する. ゆえに正当性が証明された.

2 つの DFA である M, M' に対し, M と M' が等価 $(L(M) = L(M'))$ かどうかを調べる. ここで, $M \equiv M'$, つまり $L(M) = L(M')$ が成立するためには全ての入力記号列 x について

$$q_0 \overset{x}{\underset{M}{\Longrightarrow}} p$$

かつ

$$q'_0 \overset{x}{\underset{M'}{\Longrightarrow}} p'$$

によって定まる状態 $p(\in Q), p'(\in Q')$ に対して, p, p' は共に受理状態 $(p \in F, p' \in F')$ か, あるいは共に非受理状態である $(p \notin F, p' \notin F')$ かを満足する必要がある. 理由として, もしある入力記号列 x に対して, 初期状態 q, q' からの遷移後として定まる状態対 p, p' の一方のみが受理状態となるなら M, M' の一方のみがその x を受理して $L(M) \neq L(M')$, つまり $M \not\equiv M'$ となるためである. ゆえに, M と M' の等価性を判定するためには, M と M' が共通の入力記号列によって到達する状態ついで, 一方のみが受理状態となるようなことが起こらないか否かをチェックすればよいこととなる. ここで, M と M' の状態数を $|Q|$, $|Q'|$ としたとき, このようなチャックをする対象となる状態対は高々 $|Q| \times |Q'|$ 種類であるから, 以下のようにしてそれを逐次的に求めあげれば良い.

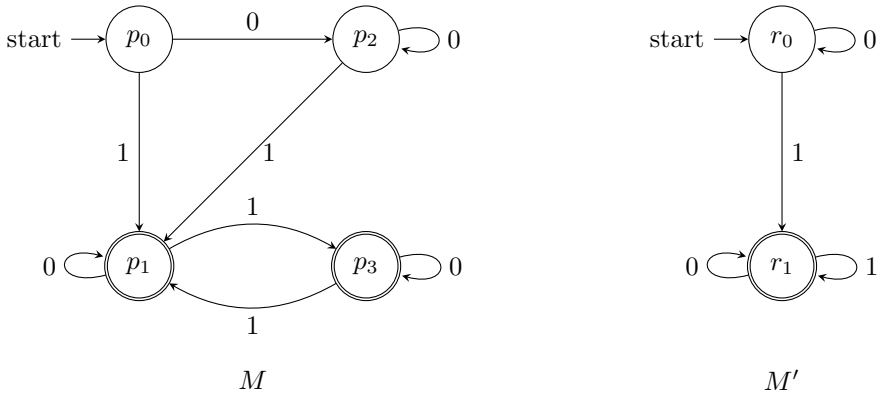


図 3.25 等価な DFA M と M'

図 3.25 の M と M' について考察してみる．到達可能状態検出木と同様な考え方を用いて, 初期状態からすべての入力に応じた遷移を出すことで等価かどうかを判断する．M と M' の状態の対を考え, 木を作っていく上で同じ状態の対が出てきたらそれ以降木を書くことを終了する．新しい状態の対が出てきたらそのときそれぞれの状態に応じた遷移先を考えることで新しい状態対を作る．図 3.25 において, 初期状態は p_0 と r_0 である．この状態の対は何も入力しないときは同じであるので等価であると言えるため, $p_0 \equiv r_0$ とする．ここで 0 が入力された場合を考える．p_0 に関しては遷移先が p_2 であり, r_0 に関しては遷移先が r_0 である．ゆえに入力記号列 0 が入力された場合 p_2 と r_0 は等価であるので $p_2 \equiv r_0$ と表す．同様にして 1 が入力された場合はそれぞれの遷移先が p_1 と r_1 であるので, $p_1 \equiv r_1$ と表せる．つぎに, 状態対 $p_2 \equiv r_0$ のときの入力記号列が与えられた時の状態対の変化をみる．この状態において 0 が入力された場合, p_2 からは p_2 に遷移し, r_0 からは r_0 に遷移する．ゆえに状態対は $p_2 \equiv r_0$ となる．ここで, $p_2 \equiv r_0$ という状態対はすでに出ているのでこれ以上のチェックを行わない．また 1 が入力され場合, p_2 からは p_1 に遷移し, r_0 からは r_1 に遷移する．ゆえに状態対は $p_1 \equiv r_1$ となる．これに関しても, $p_1 \equiv r_1$ という状態対はすでに出ているのでこれ以上のチェックを行わない．$p_1 \equiv r_1$ の状態遷移について考える．これは先の $p_2 \equiv r_0$ から状態遷移によって出てきた $p_1 \equiv r_1$ ではなく, 初期状態の対 $(p_0 \equiv r_0)$ から 1 が入力されて遷移した $p_1 \equiv r_1$ である．これに入力記号列が与えられた場合について考える．0 が入力されると p_1 からは p_1 に遷移し, r_1 からは r_1 に遷移する．ゆえに状態対は $p_1 \equiv r_1$ となる．ここで, $p_1 \equiv r_1$ という状態対はすでに出ているのでこれ以上チェックを行わない．一方で 1 が入力されると p_1 からは p_3 に遷移し, r_1 からは r_1 に遷移する．ゆえに状態対は $p_3 \equiv r_1$ となる．これは新しい状態対なのでチャックを続ける．$p_3 \equiv r_1$ から入力記号列が与えられた時の状態対の変化をみる．0 が入力されると p_3 からは p_3 に遷移し, r_1 からは r_1 に遷移する．ゆえに状態対は $p_3 \equiv r_1$ となる．これはすでに出ているのでこれ以上のチェックを行わない．1 が入力されると p_3 からは p_1 に遷移し, r_1 からは r_1 に遷移する．ゆえに状態対は $p_1 \equiv r_1$ となる．これはすでにでているのでこれ以上のチェックを行わない．以上によって初期状態から遷移してできる状態対のパターンは全て列挙されたため木の構成が終わった．ここで現れた状態対は $p_0 \equiv r_0$, $p_2 \equiv r_0$, $p_1 \equiv r_1$, $p_3 \equiv r_1$ である．この状態対から 0 または 1 の入力が与えられてもこのどれかからの状態対に遷移する．ゆえにこの後にどのような入力記号列が与えられてもその遷移先は既出の状態対である．M と M' が等価であるかどうかを調べるには, この状態対が受理状態の対または非受理状態の対となってなければいけない．一方の状態が受理状態でもう一方の状態が非受理状態であるとある入力記号列では一方が受理し, もう一方が非受理となってしまう．$p_0 \equiv r_0$ は p_0 は非受理状態, r_0 も非受理状態なので, 非受理状態対である．つぎに $p_2 \equiv r_0$ について p_2, r_0 ともに非受理状態なので, これも非受理状態対である．$p_1 \equiv r_1$ については p_1, r_1 はどちらも受理状態なので, これは受理状態対である．最後に $p_3 \equiv r_1$ について, p_3, r_1 はどちらも受理状態なので, これも受理状態対である．そのため全ての状態対において受理状態対または非受理状態対に分けることができたので等価であることがわかる．ゆえに $M \equiv M'$ であると判定することができる．この操作を木によって表現したものを**等価性判定木** (comparison tree) という．この根はそれぞれの初期状態の対である (図 3.25 においては $p_0 \equiv r_0$)．以下に等価性判定木を示す．

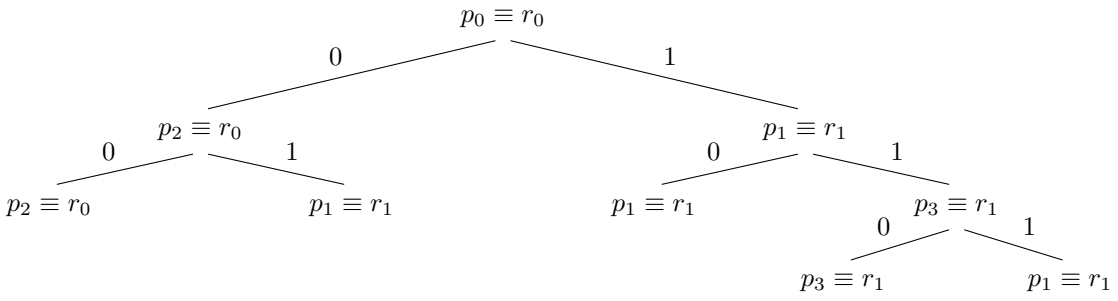


図 3.26　等価性判定木

この方法は直接的に M と M' が等価かどうかを調べる方法である. つぎに等価かどうかを調べる方法として DFA を新たに構築することで等価かどうかを調べる. 方法として $L(M)$ と $L(M')$ の対称差を受理するDFAM^D をつくり, $L(M^D)$ が ϕ であるかどうかを調べる. 集合 A, B の対称差 $A \triangle B$ とは, 片方だけに入っている元の集合のことである ($A \triangle B = (A \cap B^c) \cup (A^c \cap B)$). $M = (Q, \Sigma, \delta, q_0, F),\ M' = (Q', \Sigma, \delta', q'_0, F')$ とする. $L(M) \triangle L(M')$ を受理する DFA M^D をつくる. それで以下のように状態を定義すればよい.

- $Q^D = Q \times Q' = \{(q, q') \,|\, q \in Q, q' \in Q\}$ (両 DFA の状態の組)
- $\delta^D((q, q'), c) = (\delta(q, c), \delta'(q', c)),\ c \in \Sigma$ (それぞれの成分毎に遷移)
- $q_0^D = (q_0, q'_0)$ (両 DFA の初期状態の組)
- $F^D = \{(q, q') \,|\, (q \in F \wedge q' \notin F') \vee (q \notin F \wedge q' \in F)\}$ (片方だけ受理状態のとき)

M と M' が等価であるとき, 受理状態の場所も同じであるので対称差を利用した DFAM^D は受理状態が存在しないはずである. つまり

$$L(M^D) = \phi \Longrightarrow L(M) = L(M') \tag{3.22}$$

ということになる. 調べ方は以下のような手順である.

1. M^D の初期状態に印をつける
2. 新しく印をつけられなくなるまで 3 をくり返す.
3. 既に印をつけられている状態から遷移してこられる状態に印をつける
4. 受理状態のどれにも印がつけられていなければ $L(M^D) = \phi$, そうでなければ $L(M^D) \neq \phi$

この印というのは, 遷移したかどうかを調べるためのものである. ゆえに DFA を形成する上では必ず遷移する状態のみにしておけば (死状態ではない) 必ずどの状態にも印がつくはずである. またこのような状態において等価な DFA で形成された対称差を用いる DFA には受理状態は存在しない.

3.2.6 正則でない言語

ある有限オートマトンによって (ちょうど正確に) 受理される言語は, 正則言語とよんだ. これに対していかなる有限オートマトンによってもちょうどそれだけを受理することは不可能であるような言語は, **非正則言語**または**非正規言語** (non-regular language) とよばれる. このような言語の例を理解することは, 有限オートマトンの能力の限界を認識する上で重要である. よく知られている例として

$$L = \{a^n b^n \,|\, n \geq 1\} \tag{3.23}$$

この言語列は, 前半列と後半列において, おのおのちょうど同じ個数の a および b が続いたものだけである. したがって, そのようなものだけを受理し, それ以外は非受理とするためには, 最初の部分の a の個数を何らかの方法で記憶しておく必要がある. 一方で有限オートマトンにおける記憶状態は状態だけであり, もしその状態数よりも多い多数個の a が先頭部分に続いた入力記号列が入力されたときには, もはやその個数を正確に記憶しきれなくなり後に続く b の個数との照合が正確にできなくなってしまう. これを一般的に背理法によって証明する. もし $L(M) = L$ となる有限オートマトン M が存在したと仮定すると, 以下のように矛盾が導ける.

L を認識する DFA$M = (Q, \Sigma, \delta, q_0, F)$ が存在したとする. $|Q| = m$ とおいて $w = a^m b^m$ が受理される経路を考える. すると最初の $m + 1$ の状態のうち 2 つは同じ状態である. $q_i, q_j\ (i < j)$ とすると, $a^{m-(j-i)} b^m \in L$ となり矛盾.

状態数を m と仮定すると,

$$a^m b^m \in L(M)$$

である. この前半部分 a について着目すると

$$q_0 \xrightarrow[M]{a} q_1 \xrightarrow[M]{a} q_2 \ \cdots \ q_{m-1} \xrightarrow[M]{a} q_m \overset{b^m}{\underset{M}{\Longrightarrow}} q_{2m} \in F$$

ここで, M の状態数を m 個としているので, 遷移経過中のこれら $(m+1)$ 個の状態 $q_0, q_1, q_2, ..., q_{m-1}, q_m$ のうち, 少なくとも 1 この状態は重複して出現している. そのような重複出現を $q_j = q_k$ $(0 \leq j < k \leq n)$ とする. これは状態遷移中にループが存在することを意味している.

$$q_j \overset{a^{k-j}}{\underset{M}{\Longrightarrow}} q_k \quad (q_j = q_k)$$

つまり以下のような状態となる.

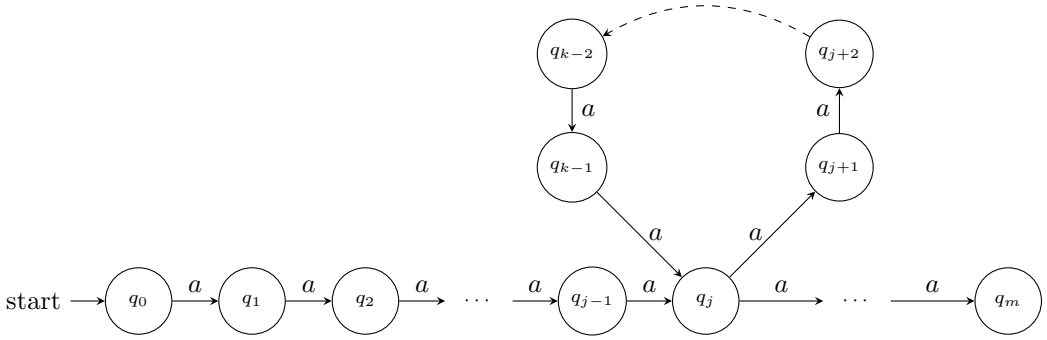


図 3.27 $w = a^n b^n$ の状態遷移の様子

閉包性を応用して正則言語でないことを証明することができる. たとえば正則言語は積集合演算に関して閉じている. ゆえにある正則言語とその他の正則言語の積集合演算でできた言語も正則言語である. 以下の言語について考える.

$$L = \{w \in \{a, b\}^+ \,|\, w \text{ に現れる } a \text{ と } b \text{ の個数が等しい } \} \tag{3.24}$$

この言語 L に関しても正則言語ではない. 正則言語である

$$L'' = \{a^m b^n \,|\, m, n \geq 1\}$$

について考えてみる. これは以下の DFA によって構成可能である.

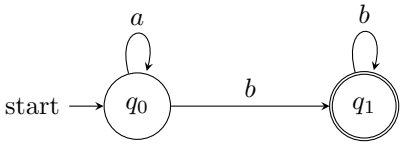


図 3.28 $L'' = \{a^m b^n \,|\, m, n \geq 1\}$ を受理する DFA

したがって, L を正則言語であると仮定すると

$$L' = L \cap L'' = \{a^n b^n \,|\, n \geq 1\}$$

も正則言語となり矛盾する. ゆえに L は正則言語ではない.

他にもいくつか例を挙げる.
$\Gamma = \{a, b\}$, $\Sigma = \{0, 1, 2\}$ とする. このとき Σ 上の言語

$$L = \{0^n 1^{2m} 2^n \,|\, n \geq 1, m \geq 1\} \tag{3.25}$$

は正則言語ではない.

写像 $h : \Sigma \to \Gamma^*$ を

$$h(0) = a$$
$$h(1) = \varepsilon$$
$$h(2) = b$$

で定義される準同型写像 $h : \Sigma^* \to \Gamma^*$ を考えると, これにより定義される言語の準同型写像により

$$L' = h(L) = \{h(w) \,|\, w \in L\} = \{a^n b^n \,|\, n \geq 1\}$$

が正則言語となる. しかしこの言語 L' は正則言語ではないので矛盾することになるので言語 L は正則言語ではない.

A を Σ 上の正則言語, B を Σ 上の任意の言語とすると

$$C = \{x \in \Sigma^* \,|\, \exists y \in B, xy \in A\} \tag{3.26}$$

は正則言語である.

A を受理する DFA を $M = \{Q, \Sigma, \delta, q_0, F\}$ とする. δ の定義を $Q \times \Sigma^*$ に自然に拡張して $F' = \{q \in Q \,|\, \exists y \in B, \delta(q, y) \in F\}$ とすると, DFA$M' = \{Q, \Sigma, \delta, q_0, F'\}$ は C を受理する. 実際, $x \in C$ とすると定義より $\exists y \in B,\ xy \in A$ だから $\delta(q_0, xy) \in F$ で, $q = \delta(q_0, x)$ とすると,, $\delta(q, y) \in F$ だから $q \in F'$ より $x \in L(M)$. 逆に $x \in L(M')$ とすると, $q = \delta(q_0, x) \in F'$ だから F' の定義より $\exists y \in B, \delta(q, y) \in F$. したがって, $\delta(q_0, xy) \in F$ すなわち $xy \in A$ であり $x \in C$.

参考文献

[1] 鈴木武・山田義雄・柴田良弘・田中和永：“理工系のための微分積分 I”, 内田老鶴圃, 2010.

[2] Charles H. Roth,Jr. and Larry L.Kinney. : *Fundamentals of. Logic Design*, CENGAGE Learning, 2014.

[3] 富田悦司・横森貴：“オートマトン・言語理論”, 森北出版, 2016.

索引

アルファベット

日本語