離散構造

筑波大学 情報学群 情報科学類

はじめに

離散構造 (Discrete Structures) は，その名の通り，離散的な構造のことであり，通常の数学 (解析，幾何，代数など) が主として連続系を対象としているのに対して，必ずしも連続的でない構造を持つものの総称である[1]．

現代のコンピュータ[2]はハードウェアのみならずソフトウェアも $0/1$ の二値に代表される離散的な構造を基礎として構成されているため，現代コンピュータ科学を理解するために必要な数学の多くは，離散的な数学 (離散数学) である．

このテキストは，離散構造の代表的なものである論理，集合，関係，関数，グラフに対する入門書である．

このテキストの読み方と注意事項

- このテキストは，基本的な内容と，発展的な内容とから構成されている．各章の題目のあとに (†) の記号がついているものは，発展的な内容であり，余力がある者が進んだ内容を学習するためのものである．(†) の記号がついていない章は，全ての者が学習すべき内容を表す．
- 数学的な議論では，事実を主張するためには必ず厳密な証明を必要とする．証明においては，何を仮定して，何を導き出そうとしているかを常に確認すること．しばしば「証明したいことを仮定している」ような誤りが見受けられる．また，どのような定義，定理，推論規則を使って証明しているのかを意識する必要がある．「自分はそのことを事実として知っている」や「当たり前」では証明にならない．
- このテキストに，誤りや不適切な説明等がある場合は，担当教官 または 担当 Teaching Assistant に連絡してほしい．
- このテキストは，『離散構造』の講義のために作成したものであり，許可なく，目的外利用・複製・再配布等をしてはいけません．

[1] 離散/連続の分類は対象ごとに厳格に決まっているものではなく，対象を見る立場に依存することがある．この授業でも，集合の例として実数全体の集合を考えたり，連続関数全体の集合を考えることがある．

[2] 将来のコンピュータがすべて離散的な原理で構成されているとは限らない．たとえば，現在活発に研究されている量子コンピュータは連続的な確率分布が基礎にある．

目 次

第 1 章　論理 **1**

1.1　非論理的な思考 1
1.2　命題と真理値 1
1.3　命題を構成する方法 2
1.3.1　「でない」 2
1.3.2　「かつ」と「または」 2
1.3.3　「ならば」 3
1.3.4　「同値」 3
1.4　命題の例 3
1.5　括弧 (かっこ) の用法 4
1.6　逆と対偶 4
1.7　必要条件と十分条件 5
1.8　基本的な証明技法 5
1.8.1　すべての場合をチェックする (しらみつぶし法) 5
1.8.2　変数を使う 5
1.8.3　「ならば」の鎖 5
1.8.4　間接法 6
1.8.5　$A \Leftrightarrow B$ の証明 7
1.8.6　避けるべき証明法 7
1.9　「すべて」と「ある」 7
1.9.1　「すべて」 7
1.9.2　「ある」 8
1.9.3　暗黙の全称記号 8
1.9.4　進んだ例 (†) 8
1.9.5　全称記号・存在記号を含む命題 (†) 9
1.9.6　全称，存在に関する命題の証明 10
1.9.7　さらに進んだ学習のために 10

第 2 章　集合 **11**

2.1　集合の構成と表現 11
2.1.1　要素を一つ一つ書き並べる方法 11
2.1.2　要素が満たすべき性質から集合を作る方法—内包的な表現 11
2.1.3　内包的な表現の発展 12
2.2　集合の等しさ 12
2.3　集合の包含関係 13
2.4　集合に関する証明法 13
2.4.1　$A \subset B$ の証明法 13
2.4.2　$A \not\subset B$ の証明法 13

2.4.3 $A = B$ の証明法 . . . 13
2.4.4 鳩の巣原理 (Pigeon hole principle) (†) . . . 14
2.5 集合の演算 . . . 14
2.5.1 べき集合 . . . 14
2.5.2 和集合と共通部分 . . . 14
2.5.3 差集合 . . . 16
2.5.4 補集合 (†) . . . 16
2.5.5 組と直積集合 . . . 17
2.6 可算集合と対角線論法 (†) . . . 18

第 3 章 関数 **21**
3.1 定義域と値域 . . . 21
3.2 複数の引数をもつ関数 . . . 22
3.3 像 (image) . . . 22
3.4 逆像 (inverse image) (†) . . . 22
3.5 関数の等しさ . . . 23
3.6 関数の合成 (composition) . . . 23
3.7 恒等関数 (identity function) . . . 24
3.8 単射 (一対一写像, one to one, injection) . . . 24
3.9 全射 (上への写像，onto, surjection) . . . 25
3.10 全単射 (bijection) と逆関数 (inverse function) . . . 25
3.11 部分関数 (partial function) . . . 25

第 4 章 関係 **27**
4.1 関係 . . . 27
4.2 二項関係の性質 . . . 27
4.3 順序 . . . 28
4.4 同値関係 . . . 29
4.5 関係の合成 . . . 30
4.6 閉包 (closure) (†) . . . 31

第 5 章 グラフと木 **34**
5.1 グラフ . . . 34
5.1.1 無向グラフ . . . 34
5.1.2 有向グラフ . . . 34
5.1.3 位数とサイズ . . . 35
5.1.4 道 (パス) と閉路 (サイクル) . . . 35
5.1.5 グラフの同型 . . . 36
5.1.6 部分グラフ . . . 36
5.1.7 連結，連結成分 (†) . . . 37
5.1.8 完全グラフ，完全 2 部グラフ (†) . . . 37
5.1.9 グラフの応用 (†) . . . 37
5.2 木 (tree) . . . 38
5.2.1 順序木 . . . 39

第 6 章　帰納的定義と帰納法 **40**
6.1　帰納的に定義された集合 40
6.1.1　リスト 41
6.1.2　文字列 42
6.1.3　2 分木 (binary Tree) 44
6.1.4　BNF 記法 (†) 46
6.2　帰納的に定義された関数 46
6.3　帰納法による証明 49
6.3.1　数学的帰納法 49
6.3.2　リストに関する帰納法 51
6.3.3　2 分木に関する帰納法 (†) 52

第1章　論理

1.1　非論理的な思考

推論は人間の思考の中心をなす行為である．しかし，日常的な会話は非論理的な推論を多く含んでいる．

例 **1** A さんと B さんの会話:

A 「雨が降ったら傘を持ってきてね．」
B 「わかった．」
（しばらく後に，B が傘を持ってきたのを見て）
A 「どうして，雨が降っていないのに傘を持ってくるの!」

もし B が傘を持ってきていなければ，(B が約束を守る人である限り) 雨は降っていないということはわかる．

しかし，B が傘を持ってきたからといって，雨が降っているとは限らない．B は，雨が降っていないときにどうするかは約束していないのだから，傘を持ってきても持ってこなくても良い．それなのに怒られては B は困惑するばかりである．

ここで用いられた論理を整理してみよう．「雨が降ったら，傘を持っていく」と「雨が降った」ということからは「傘を持っていく」ということを導くことができるが，「雨が降ったら，傘を持っていく」と「傘を持っていく」ということからは「雨が降った」ということを導けない．ここで大事なことは，上記の推論 (および非論理的な推論) は，「雨が降る」や「傘を持っていく」という個々の文が成立しているかどうかに関わらない，ということである．

論理学とは，個々の文が成立するかどうかに関係なく成立する推論について研究する学問である．

1.2　命題と真理値

命題 (proposition): 「正しい」かどうかを考える対象となる文のことである．

例 **2** 以下の文は命題である．

- 「今日は雨が降っている．」
- 「2 は 1 より大きい．」
- 「素数は有限個しか存在しない．」

以下の文は命題ではない．

- 「富士山」
- 「2 倍すると 10 になる整数」

命題が正しいことを「真 (T, true)」と言い，正しくないことを「偽 (F, false)」と言う．真と偽の値を合わせて真理値 (truth value) という．

我々が日常使う論理[1]では，命題は真か偽のいずれか一方の真理値を取る．たとえば，「2 は 1 より大きい．」という命題は真という値を取り，「素数は有限個しか存在しない．」という命題は偽という値を取る．

ところで，命題は，上記のような基本的なものだけではない．「雨が降っているならば，B は傘を持っていく．」というように「雨が降る．」と「B は傘を持っていく．」という基本的な命題を「ならば」で組み合わせたものも命題である．命題に関する論理 (命題論理) は，複合的な命題を構成する方法についての論理である．

以下では，命題 (基本的な命題または複合的な命題) を A, B, C などの文字で表す．

1.3 命題を構成する方法

「ならば」のように，命題を組み合わせて複合的な命題を構成する記号を論理結合子 (logical connective) という．論理結合子には，「かつ」「または」「ならば」「でない」「同値」などがある．

1.3.1 「でない」

A でない: $\neg A$

「でない」(not) は，否定 (negation) とも呼ばれる．命題 $\neg A$ は，A が真のとき偽となり，A が偽のとき真となる命題である．このことを，以下のような表 (真理値表，Truth Table) で表現することができる．

A	$\neg A$
真 (T)	偽 (F)
偽 (F)	真 (T)

1.3.2 「かつ」と「または」

$$\begin{cases} A \text{ かつ } B & : A \wedge B \\ A \text{ または } B & : A \vee B \end{cases}$$

「かつ」(and，論理積) は，連言 (conjunction) とも呼ばれる．「または」(or，論理和) は選言 (disjunction) とも呼ばれる．

命題 $A \wedge B$ は，A と B が両方真のときに真となり，そうでないときに偽となる命題を表す．命題 $A \vee B$ は，A と B の少なくとも一方が真のときに真となり，そうでないときに偽となる命題を表す．なお，A と B の両方とも真のときも $A \vee B$ は真である．

- $\neg(A \wedge B)$ と $\neg A \vee \neg B$ の真理値は一致する．
- $\neg(A \vee B)$ と $\neg A \wedge \neg B$ の真理値は一致する．
- $\neg\neg A$ と A の真理値は一致する．

[1] 真理値が真と偽の 2 つである論理を二値論理という．コンピュータの計算に対応する論理では，二値だけでなく「真か偽か有限時間に判定できない」ということを表す値を持つものがある (三値論理，多値論理)．

A	B	$A \wedge B$	$A \vee B$
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

1.3.3 「ならば」

A ならば B : $A \Rightarrow B$ ($A \supset B$ と書くこともある)

「ならば」(implication) は含意 (がんい) とも呼ばれる．命題 $A \Rightarrow B$ は，命題 A が真で，命題 B が偽であるときに偽となり，そうでないとき真となる．

A	B	$A \Rightarrow B$	$B \Rightarrow A$
T	T	T	T
T	F	F	T
F	T	T	F
F	F	T	T

A が偽のとき，$A \Rightarrow B$ は (B の真偽値によらず) 真となることに注意せよ．(例: 「雨が降れば傘を持っていく」という命題は，雨が降っていなければ，傘を持っていこうがいくまいが，真となる．)

$A \Rightarrow B$ と $(\neg A) \vee B$ の真理値は一致する．

1.3.4 「同値」

「同値」(equivalence) は以下の形の命題である．

A と B は同値 : $A \Leftrightarrow B$ ($A \equiv B$ と書くこともある)

命題 $A \Leftrightarrow B$ は，A と B の真偽値が一致するとき真となり，そうでないとき偽となる命題である．

A	B	$A \Leftrightarrow B$
T	T	T
T	F	F
F	T	F
F	F	T

$A \Leftrightarrow B$ と $(A \Rightarrow B) \wedge (B \Rightarrow A)$ の真理値は一致する．

$A \Leftrightarrow B$ が常に真となるとき，命題 A と 命題 B は同値である，という．

1.4 命題の例

命題論理では，基本的な命題がどのようなものであるかは特に定めないが，整数の大小関係に関する命題を基本的な命題とすれば，以下のような命題が構成できる．

例 **3**

「2 は 1 より大きく，3 より小さい」

$$1 < 2 \ \wedge \ 2 < 3$$

「正の整数 x と負の整数 y を掛けた数 $x \cdot y$ は負の整数である」

$$(x > 0 \wedge y < 0) \ \Rightarrow \ x \cdot y < 0$$

1.5 括弧 (かっこ) の用法

命題の記述において，曖昧さが生じるときは，かっこを用いる．

例えば，$A \wedge B \vee C$ と書くと，$(A \wedge B) \vee C$ のことか $A \wedge (B \vee C)$ のことかわからない．

自然言語においても「A かつ B でない」というと「(A かつ B) でない」ことなのか「A かつ (B でない)」ことなのか曖昧である．

$A \wedge B \vee C$ にように，かっこを省略した場合にどのようにかっこを補って考えるかについては，一定の約束事をする必要がある．

標準的な論理学では「結合力」が強い順に $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ とする．また，同じ記号同士であれば，$\wedge, \vee, \Leftrightarrow$ は左側にあるものが強く，$\Rightarrow$ は右側にあるものが強い，とする．たとえば，$A \wedge B \Rightarrow C \Rightarrow D$ は $(A \wedge B) \Rightarrow (C \Rightarrow D)$ のことである．

ただし，この約束事をきちんと習得するためには訓練を要するので，現段階では，必ずかっこをつけて書く習慣を付けるのがよい．

1.6 逆と対偶

$A \Rightarrow B$ という形の命題に対して，その「逆」と「対偶」と呼ばれる命題が存在する．

逆

命題 $B \Rightarrow A$ を $A \Rightarrow B$ の逆という．

(表:1.3.3) から分かるように，$A \Rightarrow B$ が真でも，その逆 $B \Rightarrow A$ が真であるとは限らない．

例 **4** A を「 x と y は奇数」，B を「$x + y$ は偶数」とする．

- $A \Rightarrow B$ は真．
- $B \Rightarrow A$ は偽．なぜならば $x = 2, y = 4$ のとき $x + y = 6$

対偶

命題 $\neg B \Rightarrow \neg A$ を $A \Rightarrow B$ の対偶という．命題 $A \Rightarrow B$ とその対偶の真理値表は一致する．

例 **5** x と y が奇数ならば，$x + y$ は偶数である．

対偶: $x + y$ が偶数でなければ，x と y が共に奇数であることはない．

1.7　必要条件と十分条件

命題 $A \Rightarrow B$ が真であるとき，命題 A は命題 B の十分条件であるという．これは，命題 B が成立することを調べるとき，命題 A が成立することが十分であることから，その名がつけられている．

また，同じ状況のとき，命題 B は命題 A の必要条件であるという．これは，命題 A が成立することを調べるとき，命題 B が成立することが必要であることから，その名がつけられている．

A が B の必要条件であるからといって十分条件とは限らないし，また，十分条件であるからといって必要条件とは限らない．

$A \Leftrightarrow B$ が真であるとき，(これは $A \Rightarrow B$ と $B \Rightarrow A$ が両方真であることと同じ) 命題 A は命題 B の必要十分条件であるという．このとき，命題 B は命題 A の必要十分条件である．

1.8　基本的な証明技法

証明を構成する場合に大事なことは，どういう推論方法を使ってよいか意識することである．

1.8.1　すべての場合をチェックする (しらみつぶし法)

組み合わせが有限個しかない場合に使える技法である．

例 **6** 命題「$1, 3, 5$ の中から 2 つの数を取って加えれば，どの場合も偶数になる」を証明する．

証明: すべての組み合わせである $1+1, 1+3, 1+5, 3+3, 3+5, 5+5$ がすべて偶数であることを示せばよい．

1.8.2　変数を使う

しらみつぶし法は，組み合わせが無限個あるときには使えない．そこで変数を使う方法がある．

例 **7** 命題「2 つの奇数の和は偶数である 」

証明: 任意の 2 つの奇数を $2n+1, 2m+1$ (ただし，n, m は任意の整数) とする．

$(2n+1)+(2m+1) = 2(n+m)+2 = 2(n+m+1)$ これは偶数．

1.8.3　「ならば」の鎖

$A \Rightarrow B$ の証明法として，以下のものがある．

1. A が正しい (真である) ことを前提として B が真であることを示せばよい．

2. これを直接示せないときは，中間に C をもってきて，「A が真であることを前提として C が真であること」と「C が真であることを前提として B が真であること」を両方示せばよい．

3. 同様に，中間に $C, D, \cdots$ をもってきて，間をつないでいけばよい．

4. 左から右を導く方法だけでなく，両方から攻めていく方法もある．例えば，$A \Rightarrow B$ を示すのに，

(a)　$C \Rightarrow B$ となる C を探し，

(b)　$A \Rightarrow C$ を示す．

ここで，「A が真である」「A が正しい」「A が成立する」という形の表現が出てきたが，このことを単に「A である」ということがある．たとえば，「$x = 0$ と仮定する」という．

例 **8** 整数 m が整数 n で割り切れることを $n \mid m$ と表す．このとき，以下が成り立つ．

(a) $n \mid m \Rightarrow n \mid a \cdot m$

(b) $n \mid m_1 \wedge n \mid m_2 \Rightarrow n \mid (m_1 + m_2)$

これらを使って，「$d \mid m \wedge d \mid n \Rightarrow d \mid (a \cdot m + b \cdot n)$」を証明する.

- A: $d \mid m \wedge d \mid n$
- B: $d \mid (a \cdot m + b \cdot n)$

A から $d \mid m$ と $d \mid n$ が導かれる．性質 (a) より，「$d \mid a \cdot m$」と「$d \mid b \cdot n$」が導かれる．しかし，「$d \mid a \cdot m \wedge d \mid b \cdot n$」はまだ求める B ではない.

性質 (b) より，「$d \mid (a \cdot m + b \cdot n)$」となり，$B$ が示せるので証明が終了する.

1.8.4 間接法

対偶による証明

$A \Rightarrow B$ を示すのに $\neg B \Rightarrow \neg A$ を示す.

例 **9** 「x^2 が偶数ならば x は偶数」を示したい．

対偶:「x が奇数ならば x^2 が奇数」を示す.

$x = 2k + 1$ とすると $x^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$ となり，奇数であることがわかる．

背理法

命題 A を証明するのに，$\neg A$ から矛盾 (contradiction) を導く方法である．ただし，矛盾とはある命題 B に対して $B \wedge \neg B$ のことである[2]．

例 **10** 命題 A:「素数は無限個ある」を証明したい．

$\neg A$ を仮定する．すると，素数は有限個なので最大の素数 p が存在する.

m を 1 から p までのすべての整数の積とする. $m = 2 \cdot 3 \cdot \cdots \cdot p$

$m + 1$ を考えると $m + 1 > p$ なので $m + 1$ は素数でない．また $m + 1 > 1$ なので，$m + 1$ はある素数 q で割り切れる. すなわち，

$$q \mid m + 1 \tag{1.1}$$

一方， $1 \leq q \leq p$ であるので，

$$q \mid m \tag{1.2}$$

(1.1)，(1.2) と $\mid$ に関する性質より，

$$q \mid 1$$

q は素数なので矛盾した．従って素数は無限個ある．

[2]「ある命題 B に対して」という部分がわかりにくいかもしれない．実は，ある命題 B に対して $B \wedge \neg B$ が導ければ，どんな命題 C に対しても $C \wedge \neg C$ が導けるので，B としてどの命題を取るかは問題ではない．

1.8.5 $A \Leftrightarrow B$ の証明

$A \Leftrightarrow B$ と $(A \Rightarrow B) \wedge (B \Rightarrow A)$ の真理値は一致するので，$A \Rightarrow B$ と $B \Rightarrow A$ を証明すればよい.

例 **11** 「x が偶数 $\Leftrightarrow$ x^2が偶数」を証明する．

$\Rightarrow$ の証明: $x = 2k$ とすると，$x^2 = 4k^2 = 2(2k^2)$ となる．

$\Leftarrow$ の証明: 対偶の例 9 による証明のところで証明を行った.

1.8.6 避けるべき証明法

(a) 「自明である.」として終らせる.

(b) 「証明は容易である.」として終らせる.

1.9 「すべて」と「ある」

命題論理では，基本命題を P や Q といった文字で表し，その内部構造は考えなかった．しかし，「x は y より小さい」といった命題を，より精密に記述するためには，述語を導入する必要がある．述語とは，$x > y^2$ における $>$ のように，データ (自然数などの「もの」) に関する性質のことである．$>$ は，データ x とデータ y の間に成立する性質 (関係) を表している．述語とデータから命題を構成することにより，命題論理より精密な表現ができるようになり，これを述語論理という．

述語論理では，命題論理の論理記号のほかに，「すべて」と「ある」を表す論理記号を新しく導入する．

1.9.1 「すべて」

「すべて...」「任意の...」「勝手な...」を表すために全称記号 (universal quantifier) $\forall$ を使う．たとえば，

$$\forall x(\mathrm{Odd}(x) \Rightarrow \mathrm{Odd}(x^2))$$

ここで $\mathrm{Odd}(x)$ は x が奇数であることを表す．

$\forall x P(x)$ が真であるとは，すべてのデータ v に対して x を v で置き換えた $P(v)$ が真であることである．そうでない時，すなわち，$P(v)$ が偽になるような v が存在するとき $\forall x P(x)$ は偽となる．

「データ」が何であるかは，議論の対象としている理論ごとに定まる．この例では，整数について議論しているので x を任意の整数で置き換える．

$$\mathrm{Odd}(0) \Rightarrow \mathrm{Odd}(0^2)$$

$$\mathrm{Odd}(1) \Rightarrow \mathrm{Odd}(1^2)$$

$$\mathrm{Odd}(2) \Rightarrow \mathrm{Odd}(2^2)$$

例 **12** 「p が素数」とは，「$p > 1$ で p を割り切る数が 1 と p のみである」ということである．

$$p > 1 \wedge \forall x(x \mid p \Rightarrow x = 1 \vee x = p)$$

1.9.2 「ある」

「ある...」「... が存在する」を表すために存在記号 (existential quantifier) $\exists$ を使う.

$$\exists x(4 = x^2)$$

$\exists x P(x)$ が真であるとは，あるデータ v に対して x を v で置き換えた $P(v)$ が真であることである．そうでないとき，すなわち，$P(v)$ が真になるような v が 1 つも存在しないとき $\exists x P(x)$ は偽となる．

例 **13** $m \mid n$: 「n が m で割り切れる.」
これは，「$m \neq 0$ かつ ある整数 k が存在して，$n = m \cdot k$ となる」ことを意味するので，以下の命題と同値である．

$$m \neq 0 \wedge \exists k(n = m \cdot k)$$

一般に，データは無限にたくさんあるので，全称記号や存在記号に対する真理値表は書くことができない．

1.9.3 暗黙の全称記号

定理の記述においては，しばしば全称記号が省略されている．たとえば，例 8 においては，

$$n \mid m \Rightarrow n \mid a \cdot m$$

等を仮定したが，これでは，任意の n, m に対してこの命題が成立することなのか，ある n, m に対して成立するということなのかわからない．ここで仮定したのは前者であるので，厳密にいうと，以下のように書くべきである．

$$\forall n \forall m(n \mid m \Rightarrow n \mid a \cdot m)$$

数学や工学の文献における定理の記述では，しばしば，このような全称記号は省略されるので適宜補って考える必要がある．

1.9.4 進んだ例 (†)

例 **14** [有理数の稠密性]
$\mathcal{R}$ を実数の集合，$\mathcal{Q}$ を有理数の集合とするとき，「任意の 2 つの実数の間に有理数が存在する.」は以下のように表現できる．

$$\forall x \forall y(x \in \mathcal{R} \wedge y \in \mathcal{R} \wedge x < y \Rightarrow \exists z(z \in \mathcal{Q} \wedge x < z \wedge z < y))$$

例 **15** [関数の連続性] 「関数 f が x において連続である.」は以下のように表現できる．
任意の $\epsilon > 0$ に対して，ある $\delta > 0$ があって，任意の y について

$$|x - y| < \delta \quad \text{ならば} \quad |f(x) - f(y)| < \epsilon$$

$$\forall \epsilon(\epsilon > 0 \Rightarrow \exists \delta(\delta > 0 \wedge \forall y(|x - y| < \delta \Rightarrow |f(x) - f(y)| < \epsilon))$$

略して書くと

$$\forall \epsilon > 0\ \exists \delta > 0\ \forall y(|x-y| < \delta \Rightarrow |f(x)-f(y)| < \epsilon)$$

「関数 f が x によらず (定義域全体で) 連続である」ことは，以下のようになる．

$$\forall x \forall \epsilon > 0\ \exists \delta > 0\ \forall y(|x-y| < \delta \Rightarrow |f(x)-f(y)| < \epsilon)$$

例 **16** [関数の一様連続性]「関数 f が (定義域全体で) 一様連続である.」は以下のように表現できる．

$$\forall \epsilon > 0\ \exists \delta > 0\ \forall x \forall y(|x-y| < \delta \Rightarrow |f(x)-f(y)| < \epsilon)$$

単なる連続性とは全称記号の位置が異なる．

1.9.5 全称記号・存在記号を含む命題 (†)

- $\forall x \forall y P(x,y)$ と $\forall y \forall x P(x,y)$ は同値．
- $\exists x \exists y P(x,y)$ と $\exists y \exists x P(x,y)$ は同値．

  「偶数と奇数の和は奇数」

  $$\forall x \forall y(\mathrm{Even}(x) \wedge \mathrm{Odd}(y) \Rightarrow \mathrm{Odd}(x+y))$$

  $$\forall y \forall x(\mathrm{Even}(x) \wedge \mathrm{Odd}(y) \Rightarrow \mathrm{Odd}(x+y))$$

  ただし，$\mathrm{Even}(x)$ は「x は偶数である」を表す．
- $\neg \forall x P(x)$ と $\exists x \neg P(x)$ は同値．
- $\neg \exists x P(x)$ と $\forall x \neg P(x)$ は同値．

  「すべての整数が偶数とは限らない」

  $$\neg\ \forall\ x \mathrm{Even}(x)$$

  「ある整数 x が存在して，x は偶数でない.」

  $$\exists x\ \neg\ \mathrm{Even}(x)$$

- $\forall x \exists y P(x,y)$ と $\exists y \forall x P(x,y)$ は同値ではない．

  「任意の整数 x より大きい整数 y が存在する.」(真)

  $$\forall x \exists y(x < y)$$

  「ある整数 y が存在して，任意の整数 x より大きい.」(偽)

  $$\exists y \forall x(x < y)$$

- $\exists x(P(x) \wedge Q(x))$ と $\exists x P(x) \wedge \exists x Q(x)$ は同値ではない．

1.9.6　全称，存在に関する命題の証明

全称に関する証明

$\forall x P(x)$ を証明するには，(x に対して何も条件を仮定せずに) $P(x)$ を証明すればよい．

存在に関する証明

$\exists x P(x)$ を証明するためには，$P(v)$ を満たす v が存在することを示せばよい．

例 **17** 「80 と 88 の間に素数が存在する.」を示すためには，具体的に例を 1 つ示せばよい．たとえば，83 は素数.

「$x^2 - 3x + 2 = 0$ を満たす整数 x が存在する」具体的に例を示せばよい. $x = 1$ のとき条件を満たす.

なお，具体的な v が見つからないときでも，背理法と組み合わせることによって存在を証明することができる場合がある．

例 **18** 「実数上の連続関数 f に対して，$f(0) = 0$ かつ $f(x) = 1$ ならば $f(x) = 0.5$ となる x で $0 < x < 1$ となるものが存在する」

このような x が存在しなければ矛盾することを示せばよい．

全称命題が偽であることを示す証明

$\neg\forall x P(x)$ を示すために，$\exists x \neg P(x)$ を示せばよい．

このような x を反例 (counterexample) という．

「素数は奇数である.」は偽である．なぜならば，2 は素数．しかし奇数でない.

1.9.7　さらに進んだ学習のために

本章の内容は，論理学の入口にはいるまで (まだ論理学とは言えない段階) である．参考書としては，守屋悦朗「コンピュータサイエンスのための離散数学」，pp 25-31 をあげておく．

論理学の世界の入口から中にはいってみたい人は，同書の pp. 137-152 を参考にするとよい．ただし，この本のこの部分はいかにも技術的な内容を列挙してあり，わかりにくい．

本格的な論理学者がきちんと書いた本の中には，かえって，非常にわかりやすいものがある．ここでは戸田山和久「論理学をつくる」，名古屋大学出版会をあげておく．この本の最初の方だけでも眺めてみると，目から鱗が落ちるであろう．

論理学をもう少し学びたい人には，論理に関係する情報学類の授業として，2 年生向けの「論理と形式化」，3-4 年生向けの「計算論理」などがある．

第2章　集合

何らかのデータに対する理論を展開するとき，データの集まりを考えると便利である．集合 (set) は，「データの集まり」のことである．

この章では，集合を A, B, C などの記号であらわす．集合に入っている「もの」を, その集合の要素 (element) という. また，元 (げん) ということもある．a が集合 A の要素であることを, $a \in A$ と書く. a が集合 A の要素でないことを, $a \notin A$ と書く. 左右を逆にして，$A \ni a$ や $A \not\ni a$ と書くこともある．

例 **19** 自然数の集合を $\mathcal{N}$ とする. $3 \in \mathcal{N}$, $-2 \notin \mathcal{N}$

2.1　集合の構成と表現

2.1.1　要素を一つ一つ書き並べる方法

有限個の要素を持つ集合 (有限集合) の場合，要素を全て書き並べることで集合を定めることができる．

例 **20**

$\{1, 2, 3, 4, 5\}$
$\{5, 2, 1, 3, 4\}$
$\{1, 1, 1\}$
$\{\{1, 2\}, \{2, 3, 4\}\}$

要素が 1 つもない集合は空集合 (empty set) とよばれ，ϕ もしくは $\{\}$ と表される．

なお，無限集合に対しても同様の表現を使うことがあるが，これはあくまで読み手の直感に訴える便宜上のものであり，数学的に厳密な定義ではない．

厳密な定義でない例: $\mathcal{N} = \{0, 1, 2, \cdots\}$

自然数の集合を厳密に構成する方法は後の帰納的定義の章で説明するが，さしあたり，この章では，例を記述する際には，$\mathcal{N}$, $\mathcal{Z}$, $\mathcal{Q}$, $\mathcal{R}$ を，それぞれ，自然数の集合 (0 を含む)，整数の集合，有理数の集合，実数の集合を表す記号とする．

2.1.2　要素が満たすべき性質から集合を作る方法—内包的な表現

A が集合であるとき，ある性質を満たす A の要素を全て集めた集合を作ることができる．ここで，「ある性質」は命題で記述する．$P(x)$ を満たす $x \in A$ を全て集めた集合は以下のように表記する．

$$\{x \in A \mid P(x)\}$$

たとえば，$\{x \in \mathcal{R} \mid 0 \leq x \leq 1\}$ は，$[0, 1]$ 区間 (0 以上 1 以下の実数の集合) を表す．

$S = \{x \in A \mid P(x)\}$ とするとき，$x \in S$ と $x \in A \wedge P(x)$ は同値である．すなわち，$(x \in S) \Leftrightarrow (x \in A \wedge P(x))$ が成立する．

注意: 同じ集合が何通りもの方法で表現されることがある．たとえば，$\{1,2,3\}$ と $\{1,1,2,3,2\}$ と $\{x \in \mathcal{N} \mid 1 \leq x \leq 3\}$ とはすべて同じ集合を表現する．このことは，後で集合同士の等しさを定義するときに明らかになる．

2.1.3 内包的な表現の発展

集合の内包的な表現には，様々な変種 (バリエーション) がある．

$\{x \in A \mid P(x)\}$ を，$\{x \mid x \in A \wedge P(x)\}$ や $\{x \mid P(x) \wedge x \in A\}$ と書くことがある．また，$x \in A$ が前後の文脈から明らかなときは省略することがある[1]．たとえば，実数について記述していることが明らかなときは，$[0,1]$ 区間を $\{x \mid 0 \leq x \ \wedge \ x \leq 1\}$ と書くことがある．

また，$\mid$ の左側に，変数以外の式を書くことがある．たとえば，$\{e \mid P(x)\}$ は，$\{y \mid \exists x.(y = e \wedge P(x))\}$ という集合を表す．

例 **21** 偶数の集合，奇数の集合は，それぞれ $\{2k \mid k \in \mathcal{Z}\}, \{2k+1 \mid k \in \mathcal{Z}\}$ と表される．

内包的表記 $\{x \mid P(x)\}$ を用いるとき，命題 $P(x)$ に x 以外の変数が現れることがあるが，その場合は存在記号を補って考える．

例 **22** 集合 $\{x \mid x = y^2 \wedge y \in \mathcal{N}\}$ は $\{x \mid \exists y(x = y^2 \wedge y \in \mathcal{N})\}$ のことである．

2.2 集合の等しさ

集合 A と集合 B は次の命題が成立するとき等しいといい, $A = B$ と書く.

$$\forall x \ (x \in A \quad \Leftrightarrow \quad x \in B)$$

すなわち，A のすべての要素が B の要素であり, B のすべての要素が A の要素であるとき A と B は集合として等しい．

この定義から，集合の表記においては，要素を並べる順番や要素の重複は気にしなくて良いことがわかる．

例 **23**

- $\{1,2,3\} = \{2,3,1\}$
- $\{1,1,2,3\} = \{1,2,3\}$
- $\{1,2\} = \{1,2,2\} = \{1,1,2\} = \{1,2,1,2\}$
- $\{1,2\} = \{x \mid x = 1 \vee x = 2\}$
- $\phi = \{x \mid x = x + 1\}$

集合 A と集合 B が等しくないとき $A \neq B$ と表す. 例えば，$\{1,2,3\} \neq \{1,2\}$．

[1] これはあくまで省略であって，常に $x \in A$ を補って考えなければいけない．$x \in A$ となる A がない場合，つまり，$\{x \mid P(x)\}$ という集合を作る原理からは，Russell の逆理 (パラドックス) とよばれる矛盾をひきおこすことが知られている．

2.3 集合の包含関係

集合 A が集合 B に含まれるとは，次の命題が成立することであり，このことを $A \subset B$ とあらわす．また，$B \supset A$ と書くこともある.

$$\forall x \ (x \in A \quad \Rightarrow \quad x \in B)$$

これは，A のすべての要素が B の要素になっていることを意味している. このとき A は B の部分集合 (subset) であるという.
$A = B$ は，$A \subset B \wedge B \subset A$ と同値である．

例 **24**
$\{a\} \subset \{a, b\}$
$\mathcal{N} \subset \mathcal{R}$

2.4 集合に関する証明法

2.4.1 $A \subset B$ の証明法

任意の $x \in A$ が $x \in B$ であることを示せばよい．

例 **25** $\texttt{Prime}(x)$ を「x は素数 (prime number) である」ことを表す命題とする．$A = \{x \mid \texttt{Prime}(x) \wedge 42 \leq x \leq 51\}$, $B = \{x \mid x = 4k + 3 \wedge k \in \mathcal{N}\}$ のとき，$A \subset B$ を証明する.

$a \in A$ とすると，$a = 43 \vee a = 47$ である.
$a = 43$ のとき，
$a = 4 \times 10 + 3, 10 \in \mathcal{N}$ であるから $a \in B$
$a = 47$ のとき，
$a = 4 \times 11 + 3, 11 \in \mathcal{N}$ であるから $a \in B$
したがって, $A \subset B$ である．

2.4.2 $A \not\subset B$ の証明法

$A \subset B$ が成立しないことを $A \not\subset B$ と書く．これを証明するには，$x \in A$ であって $x \notin B$ である x を見出せばよい．

例 **26** $A = \{3k + 1 \mid k \in \mathcal{N}\}$, $B = \{4k + 1 \mid k \in \mathcal{N}\}$ のとき, $A \not\subset B$ かつ $B \not\subset A$ を証明する

$4 \in A$ かつ $4 \notin B$
$5 \in B$ かつ $5 \notin A$

2.4.3 $A = B$ の証明法

$A \subset B$ かつ $B \subset A$ を示す.

例 **27** $A = \{x \mid x$ は素数かつ $12 \leq x \leq 18\}$, $B = \{x \mid \exists k \in \{3, 4\} \ x = 4k + 1\}$ の時, $A = B$ を示す.

$A \subset B$ を示す. $x \in A \wedge (x = 13 \vee x = 17)$ である．$13 = 4 \times 3 + 1, 17 = 4 \times 4 + 1$
$B \subset A$ を示す. $x \in B, x = 4 \times 3 + 1$ か $x = 4 \times 4 + 1$ いずれも素数で, $12 \leq x \leq 18$ を満たす.
従って, $A = B$

2.4.4 鳩の巣原理 (Pigeon hole principle) (†)

有限集合に対する証明技法に鳩の巣原理と呼ばれるものがある．n 個の箱に m 個の手紙が配達されているとする. $m > n$ とすると少なくとも 1 つの箱には 2 つ以上の手紙が配達されることがわかる，という原理である．

例 **28** 集団が 367 人からなるとき，その中には同じ誕生日を持つ人が少なくとも 1 組はいる．

例 **29** 1 以上 $2n$ 以下の整数の集合の中から $n+1$ 個の整数を取ってくると，その中に必ず，片方が他方を割り切るものがある．

これを示すために，まず，$B = \{k \in \mathcal{N} \mid 1 \leq k \leq 2n\}$ とする．また，$1 \leq k < 2n$ なる奇数 k に対して，$A_k = \{k \cdot 2^a \mid (0 \leq a) \ \wedge \ (k \cdot 2^a \leq 2n)\}$ とする．このような A_k は k が 1 以上 $2n$ 未満の奇数であるためちょうど n 個である．また，$B = A_1 \cup A_3 \cup \cdots \cup A_{2n-1}$ である．したがって，集合 B から $n+1$ 個の要素を取ってくると，必ず，ある A_k から 2 個以上の要素を取ってくることになる．同じ A_k に属する 2 つの数は必ず一方が他方を割り切るので，そのような整数の組が見つかった．

2.5 集合の演算

集合の構成方法の 3 番目として，既にある集合から演算を行って集合を作る方法がある．集合上の演算の主なものには「べき集合」「和集合」「共通部分」「差集合」「補集合」「直積」がある．

2.5.1 べき集合

集合 A のべき集合 (power set) とは，A の部分集合をすべて集めた集合のことである．これを，2^A や $\mathcal{P}(A)$ と表記する．すなわち，以下の命題が成立する．

$$\forall x. \ (x \in 2^A \Leftrightarrow x \subset A)$$

この式に現れる x 自身がまた集合であることに注意せよ．

例 **30** $A = \{a, b, c\}$ とすると

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

2.5.2 和集合と共通部分

2 つの集合の和集合と共通部分をあらわす集合を表す記号は以下の通りである．

$\cup$ 和集合 (結び, union)

$\cap$ 共通部分 (交わり, intersection)

すなわち，A, B が集合であるとき，$A \cup B$ と $A \cap B$ は，それぞれ，和集合と共通部分を表す集合である．

これらの集合は以下の性質を満たす．

$\forall x.(x \in A \cup B \Leftrightarrow (x \in A \vee x \in B))$
$\forall x.(x \in A \cap B \Leftrightarrow (x \in A \wedge x \in B))$

例 31 $A = \{1, 2\}, B = \{2, 3, 4\}$ **とすると，**
$A \cup B = \{1, 2, 3, 4\}$
$A \cap B = \{2\}$

和集合の性質

(a) $A \cup \phi = A$

(b) $A \cup B = B \cup A$ (**交換法則**)

(c) $A \cup (B \cup C) = (A \cup B) \cup C$ (**結合法則**)

(d) $A \cup A = A$ (**べき等法則**)

(e) $A \subset B \Leftrightarrow A \cup B = B$

(e) **の証明**

(1) $A \subset B \Rightarrow A \cup B = B$ **を証明する．**
$A \subset B$ **と仮定する．**

(a) ($A \cup B \subset B$ **を示す**)
$x \in A \cup B$ **とすると** $x \in A \vee x \in B$ **である．**
$x \in A$ **ならば,** $A \subset B$ **より** $x \in B$
したがっていずれにしても $x \in B$ **である．**

(b) ($B \subset A \cup B$ **を示す**)
$x \in B$ **とすると** $x \in A \vee x \in B$ **である．**
よって $x \in A \cup B$ **である．**

(2) $A \cup B = B \Rightarrow A \subset B$ **を証明する．**
$x \in A$ **とすると上と同様の推論により** $x \in A \cup B$ **である．**
$A \cup B = B$ **より** $x \in B$ **である．**

共通部分の性質

(a) $A \cap \phi = \phi$

(b) $A \cap B = B \cap A$ (**交換法則**)

(c) $A \cap (B \cap C) = (A \cap B) \cap C$ (**結合法則**)

(d) $A \cap A = A$ (**べき等法則**)

(e) $A \subset B \Leftrightarrow A \cap B = A$

分配法則

(a) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

(b) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

2.5.3 差集合

集合 A と集合 B の差集合を $A-B$ と表記する．これは以下の性質を満たす集合である．

$$\forall x.(x \in A-B \Leftrightarrow (x \in A \wedge x \notin B))$$

例 **32** $A=\{a,b,c\}, B=\{c,d\}$ とすると $A-B=\{a,b\}$

なお，集合における演算は数の上の演算と異なり，「和」と「差」は逆演算とはならない．たとえば，$(A \cup B)-B=A$ や $(A-B)\cup B=A$ は必ずしも成立しない．

2.5.4 補集合 (†)

考察の対象としているものの全体の集合が 1 つに定まっているとき，それを全体集合という．たとえば，整数についての議論をしているときは整数全体の集合 $\mathcal{N}$ が全体集合である．

全体集合 U の中で集合 A を考えているとき，A に属さない要素の集合を A の補集合 (complement) といい，$\overline{A}$ で表す．補集合は，集合の差の記号を使って，$\overline{A}=U-A$ と表現することができる．すなわち，以下の命題が成立する．

$$\forall x.(x \in \overline{A} \Leftrightarrow x \notin A)$$

ただし，この式において，$\forall x$ の x が動く範囲は全体集合 U である．

補集合の性質

(a) $\overline{\overline{A}}=A$

(b) $\overline{\phi}=U, \quad \overline{U}=\phi$

(c) $A \cap \overline{A}=\phi, \quad A \cup \overline{A}=U$

(d) $A \subset B \Leftrightarrow \overline{B} \subset \overline{A}$

(e) $\overline{A \cup B}=\overline{A} \cap \overline{B}, \quad \overline{A \cap B}=\overline{A} \cup \overline{B}$ (ド・モルガンの法則)

(d) の証明

$A \subset B \Rightarrow \overline{B} \subset \overline{A}$ の証明
$x \in \overline{B}$ とすると $x \notin B$ である．ここで $x \notin \overline{A}$ と仮定する．すなわち，$x \in A$
$A \subset B$ より $x \in B$ なので，矛盾する．
したがって，$x \in \overline{A}$ である．

$\overline{B} \subset \overline{A} \Rightarrow A \subset B$ の証明
$x \in A$ とする．$x \notin B$ とすると (すなわち $x \in \overline{B}$), $\overline{B} \subset \overline{A}$ より, $x \in \overline{A}$. ここで, $x \notin A$ となり, 矛盾. したがって, $x \in B$. すなわち $A \subset B$.

2.5.5 組と直積集合

組 (タプル , tuple) とは , 有限個の「もの」を一列に並べたものである . ここでは , 組を $\langle 1,2,3\rangle$ のようにあらわす . 組は集合と異なり , 要素の重複 , 要素の順序が意味を持つ .

例 **33**

- $\langle 0\rangle$ ... 0 だけからなる組
- $\langle 0,1,1\rangle$ ... 3 個の要素からなる組 ($\langle 0,1\rangle$ とは異なる)
- $\langle 0,1,1,1,2\rangle$. ... 5 個の要素からなる組 ($\langle 2,1,1,0,1\rangle$ とは異なる組である .)
- $\langle\rangle$ (0 個の要素からなる組) .

長さが 2 の組を , 対 (つい , pair) という. 組 $\langle x_1, x_2, \ldots, x_n\rangle$ に対して, x_i を i 番目の要素という.

集合 A と B の直積集合 (cartesian product) $A \times B$ は以下の性質を見たす集合である .

$$\forall z.((z \in A \times B) \Leftrightarrow \exists x\ \exists y\ ((z = \langle x, y\rangle) \wedge (x \in A) \wedge (y \in B)))$$

この定義は , やや複雑だが , 以下のように書きかえると理解しやすい .

$$\forall v\ \forall w\ (\langle v, w\rangle \in A \times B \Leftrightarrow (v \in A \wedge w \in B))$$

例 **34** $A = \{a, b\}$, $B = \{0, 1, 2\}$ とする.

$$A \times B = \{\langle a, 0\rangle, \langle a, 1\rangle, \langle a, 2\rangle, \langle b, 0\rangle, \langle b, 1\rangle, \langle b, 2\rangle\}$$

$$\emptyset \times B = B \times \emptyset = \emptyset$$

集合 $A_1, \ldots, A_n$ の直積は, 下のように定義される.

$$A_1 \times \cdots \times A_n = \{\langle a_1, \ldots, a_n\rangle \mid a_1 \in A_1 \wedge \cdots \wedge a_n \in A_n\}$$

集合 A の n 個の直積 $\overbrace{A \times \cdots \times A}^{n}$ を A^n と書く .

$$A^0 = \{\langle\rangle\}(\text{これは } \emptyset \text{ とは異なる})$$

$$A^1 = \{\langle a\rangle \mid a \in A\}$$

例 **35** $A = \{0, 1\}$ とする.

$$A^0 = \{\langle\rangle\}$$

$$A^1 = \{\langle 0\rangle, \langle 1\rangle\}$$

$$A^2 = \{\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle, \langle 1, 1\rangle\}$$

A^1 が A と等しくないことに注意せよ[2] .

[2]ただし , 文献によっては , $A^1 = A$ として定義することもある .

2.6 可算集合と対角線論法 (†)

コンピュータは，真の無限をそのまま扱うことはできないが，有限的な要素の極限としての無限は，コンピュータ科学にとって必要な概念である．有限集合の極限としての無限は，可算無限と呼ばれ，それより大きな無限 (非可算無限) とは区別される．

集合 A が数え挙げられる (列挙できる, denumerable) とは，A の全ての要素を，1 番目，2 番目，3 番目，$\cdots$ という (有限もしくは無限の) 列として列挙できる[3]ということである．この列に A の要素が 2 回以上現れてもよいが，A の要素はすべてこの列に現れなければならない．

例 **36** 有限集合，自然数の集合 $\mathcal{N}$は列挙できる．自然数の集合の任意の部分集合は列挙できる．

整数の集合 $\mathcal{Z}$は列挙できる．なぜなら，$0, 1, -1, 2, -2, \cdots$ という列で数え挙げられるから．

C 言語で書かれたコンピュータプログラムを全て集めてきた集合 (過去に書かれたものだけでなく，C のプログラムと見なされるもの全てを集めた集合) は，列挙できる．なぜなら，コンピュータプログラムは記号列であり，その中に含まれる記号を，ASCII コードなどの符号化により bit 列として表現すれば，全体として (非常に大きな)1 つの自然数と見なすことができるから．この場合，C 言語のプログラムの集合全体は，自然数の集合の部分集合となる (自然数の中には，その bit 列の表現が，C 言語のプログラムになっていないものもある)．

定義 **1**

- 集合 A が数え挙げられるとき，A を可算集合 (countable set) という．
- 有限集合でない可算集合を可算無限集合という．
- 可算でない集合を, 非可算 (uncountable) 集合という．

例 **37** 有限集合および自然数の集合 $\mathcal{N}$は順番をつけて列挙できるので可算集合である．

例 **38** 正の有理数の集合 $\{x \in \mathcal{Q} \mid 0 < x\}$ は可算集合である.

$$\begin{array}{cccc} \frac{1}{1} & & & \\ \frac{1}{2} & \frac{2}{1} & & \\ \frac{1}{3} & \frac{2}{2} & \frac{3}{1} & \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

というように数え挙げればよい. (同じ有理数が 2 回以上現れるが，数え上げにおいては，2 回以上現れてもよいので問題ない．)

例 **39** 可算個の可算集合に対して，それらの和集合は可算集合である．

$$A = A_0 \cup A_1 \cup A_2 \cup \cdots \cup A_n \cup \cdots$$

とした時, $A_0, A_1, \cdots$ を以下のようにすれば,

$$\begin{array}{rllll} A_0 = & a_{00}, & a_{01} & a_{02} & \cdots \\ A_1 = & a_{10}, & a_{11} & \cdots & \\ A_2 = & a_{20}, & \cdots & & \\ \vdots & & & & \end{array}$$

[3]正確には，A が空集合であるか，または，$\mathcal{N} \to A$ となる全射が存在する，という条件となる．「全射」については，後の関数の章を参照のこと．

$a_{00}, a_{01}, a_{10}, a_{02}, a_{11}, \cdots$ というように対角線にしたがって数え挙げることができる. よって, A は可算集合[4] .

正の有理数の集合が可算であることと同様に負の有理数の集合が可算であることが導ける . それらと上記の例を合わせると , 有理数全体の集合 $\mathcal{Q}$が可算であることがわかる .

一方 , 非可算集合も存在する . ここでは , 例として , 自然数の集合のべき集合 $2^{\mathcal{N}}$ が非可算であることを示す .

定理 1 $2^{\mathcal{N}}$ は非可算である.

証明

$T = 2^{\mathcal{N}}$ とおく . T が可算であると仮定して矛盾を導くことにする .

可算の定義より, T の要素は, $S_0, S_1, \cdots, S_n, \cdots$ と数え挙げられる . ここで , 以下のような集合 V を作る.

$$V = \{n \in \mathcal{N} \mid n \notin S_n\}$$

V は確かに集合であり , $V \subset \mathcal{N}$ であるので , $V \in T$ である . 従って , V は $S_0, S_1, \cdots$ の列に現れる . そこで , $V = S_k$ とする . すると ,

$$\begin{aligned} k \in V &\Leftrightarrow k \notin S_k \\ &\Leftrightarrow k \notin V \end{aligned}$$

となる . これは , $k \in V$ とその否定が同値であることになり , 矛盾である .

したがって , 最初の仮定である「T が可算である」が否定され , T が非可算であることがわかった .

この証明で用いた証明技法は , 対角線論法とも呼ばれ , 広い範囲で応用される方法である .

なお , この定理の応用として , 実数の集合 $\mathcal{R}$も非可算であることがいえる . 実数の定義を数学的に与えることはこのテキストの範囲を越えるので , ここでは , $\mathcal{R}$が非可算であることの直感的な説明を与えるのみとする .

集合 $2^{\mathcal{N}}$ は , 以下の対応により実数の集合 $\mathcal{R}$に埋め込むことができる .

任意の $A \in 2^{\mathcal{N}}$ に対して ,

$$0.d_0\ 0\ d_1\ 0\ d_2\ 0 \cdots$$

という 2 進数表記の実数を対応付ける[5] . ただし , d_i は A に応じて以下のように定める数字 (0 または 1 となる数字) である .

$$d_i = \begin{cases} 1 & i \in A \text{ のとき} \\ 0 & i \notin A \text{ のとき} \end{cases}$$

たとえば , 偶数の集合には ,

$$0.1000100010001000 \cdots$$

という実数が対応付けられ , 奇数の集合には ,

$$0.0010001000100010 \cdots$$

[4] ここに書いたものは証明ではなく , 証明のアイディアに過ぎない . 読者は自力で厳密な証明を完成してほしい .

[5] ここで小数点以下を $d_0 d_1 ...$ とせず , 1 桁おきに 0 をはさんでいるのは , S ごとに異なる実数を対応付けたいからである .

という実数が対応付けられる．

このようにすると，$A, B \in 2^{\mathcal{N}}$ なる集合 A, B が異なれば，異なる実数が対応付けられることがわかる．したがって，もし，実数の集合 $\mathcal{R}$が列挙できれば，$2^{\mathcal{N}}$ も列挙でき，前述の定理と矛盾する．よって，実数の集合 $\mathcal{R}$ は可算でない．

コンピュータ上で表現可能なプログラムやデータが可算であるという事実と，$\mathcal{R}$が可算でないという事実から，全ての実数をそのままコンピュータ上で操作可能なデータとして表現することはできないことがわかる．

第3章　関数

A, B を集合とする．集合 A の各々の要素に対して集合 B の要素を対応させる仕方を関数 (function)，あるいは，写像 (map または mapping) という.

より正確にいうと，集合 A から集合 B への関数は，集合 A の全ての要素を集合 B の要素に対応付け，かつ，集合 A の全ての要素に対して，それに対応付けられる集合 B の要素は唯 1 つとなるものである．

例 **40** 関数と関数でないものの例.

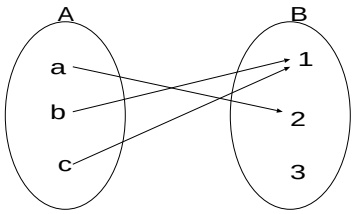


図 3.1: 関数の例 (集合 $\{a, b, c\}$ から集合 $\{1, 2, 3\}$ への関数)

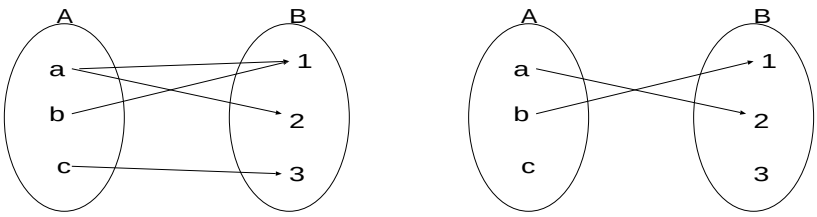


図 3.2: 関数でない例 (左: a に対応する値が 2 つある．右: c に対応する値がない．)

3.1　定義域と値域

f が集合 A から集合 B への関数であることを $f : A \to B$ と表す．このとき，A を f の定義域 (始域, ソースとも言う, domain), B を f のコドメイン (終域, ターゲット とも言う, codomain)[1] と呼ぶ. また，f によって $x \in A$ が $y \in B$ に対応付けられる (写される) ことを $f(x) = y$ と書く．x を f の引数 (ひきすう，argument)，y を f による x の値 (あたい，value) という．

[1] 高校までで習った「値域 (range)」と、コドメインは異なるものである。たとえば、実数から実数への関数 f が $f(x) = 0$ で定義されるとき f のコドメインは実数の集合だが、f の値域は $\{0\}$ である。ややこしいことに、昔は、「値域」という言葉をコドメインの意味にも使っていたので、用語が混乱していることがある。このテキストも去年までは「昔」の用語を使っていた。

3.2 複数の引数をもつ関数

関数の定義域を集合の直積とすることにより，引数を 2 個以上もつ関数を表すことができる．

二引数関数 (二項関数, binary function): $f : (A \times B) \to C$

なお，通常は，$A \times B \to C$ のように，かっこを省略する．$x \in A$, $y \in B$ に対して $f(\langle x, y \rangle)$ のことを $f(x, y)$ とも書く．

例 **41** 二つの整数の和を求める関数を考える. $plus : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$. $plus(\langle x, y \rangle) = x$ と y の和. あるいは, $plus(x, y)$ と書く.

二引数関数と同様に多引数関数 (多項関数) も定義できる．

$f : A_1 \times A_2 \times \cdots \times A_n \to B$

例 **42** n 個の自然数の最大値を求める関数 max は，$max : \mathcal{N}^n \to \mathcal{N}$ と書ける．

コンピュータ科学においては，引数の個数 n が不定の場合を扱うこともある．たとえば，いくつかの (何個かわからない) 自然数を与えられて，その中の最大値を返す関数を考えることもある．本講義資料の範囲内では，そのようなものは関数とは考えない．

3.3 像 (image)

関数 $f : A \to B$ と 集合 $C \subset A$ に対して，f による C の像 $f(C)$ は以下で定義される．

$$f(C) = \{f(x) \in B \mid x \in C\}$$

図 3.3 参照．

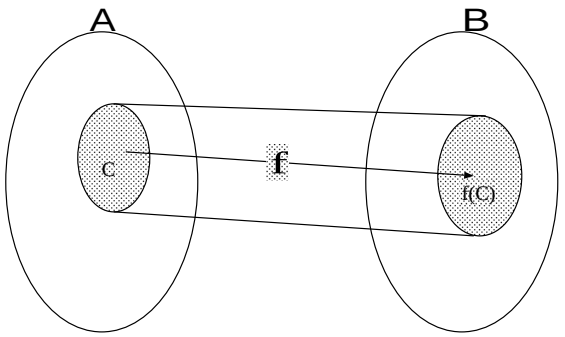


図 3.3: 関数 f による集合 C の像 $f(C)$.

像に関して, 以下の性質が成り立つ.

$$y \in f(C) \Leftrightarrow \exists x \in C \; y = f(x)$$

3.4 逆像 (inverse image) (†)

$f : A \to B$ と $D \subset B$ に対して，f による D の逆像 $f^{-1}(D)$ は以下のように定義される．

$$f^{-1}(D) = \{x \in A \mid f(x) \in D\}$$

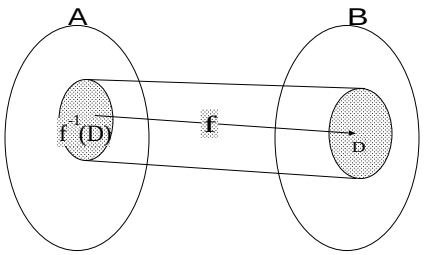


図 3.4: $f^{-1}(D)$.

逆像に関して, 以下の性質が成り立つ.

$$x \in f^{-1}(D) \Leftrightarrow f(x) \in D$$

例 43 図 3.5 に対して，関数 f の定義域は $A = \{a, b, c\}$, f の値域は $B = \{1, 2, 3\}$, $f(\{a, c\}) = \{1, 2\}$, $f(A) = \{1, 2\}$, $f(\{a\}) = \{1\}$, $f(\{a, b\}) = \{1\}$, $f^{-1}(\{1, 2\}) = \{a, b, c\}$, $f^{-1}(\{1, 3\}) = \{a, b\}$, $f^{-1}(\{3\}) = \phi$, $f^{-1}(B) = \{a, b, c\} = A$.

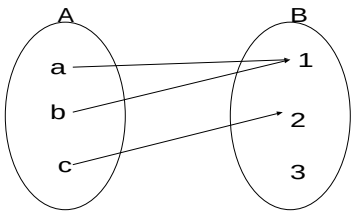


図 3.5: 関数 f.

3.5 関数の等しさ

定義域と値域が同じ 2 つの関数 $f : A \to B$, $g : A \to B$ が等しいとは，以下が成立することである．

$$\forall x \in A \quad f(x) = g(x)$$

このとき $f = g$ と書く. 関数の等しさとして他の定義を考えることもあるので，この等しさの定義を特に外延的な等しさ (extensional equality) という．

例 44 $f(x) = 2x$, $g(x) = x + x$ とすると $f = g$ である．

3.6 関数の合成 (composition)

2 つの関数 $f : A \to B$, $g : B \to C$ があるとき，f と g の合成を定義することができる．すなわち，$g \circ f : A \to C$ は，$(g \circ f)(x) = g(f(x))$ で定義される関数をあらわす．

ここで，$g \circ f$ における f, g の順番に注意する必要がある．f を先に適用して次に g を適用した合成関数を表すときに $g \circ f$ と表記する．これは，直感に反するが，$(g \circ f)(x) = g(f(x))$ とい

う自然な定義に対応付けるためのものである．一方，第 4 章では，関係の合成 $R \circ T$ を定義するが，この時は R を先に適用して次に T を適用する．すなわち，関数の合成と関係の合成は同じ $\circ$ という記号を使うが，逆の順番であることに注意されたい．関係と関数で合成の順番を変える絶対的な理由はなく，過去の慣習によるものである．

合成 $\circ$ は，以下の法則 (結合法則) を満たす:

$$(h \circ (g \circ f))(x) = h((g \circ f)(x)) = h(g(f(x))) = (h \circ g)(f(x)) = ((h \circ g) \circ f)(x)$$

しかし，$f \circ g = g \circ f$ とは限らない．

例 **45** $f(x) = x^2$, $g(x) = x + 1$. $(f \circ g)(x) = f(x + 1) = (x + 1)^2$. $(g \circ f)(x) = g(x^2) = x^2 + 1$.

3.7 恒等関数 (identity function)

集合 A 上の恒等関数 $id_A : A \to A$ とは，$id_A(x) = x$ で定義される関数である．
$f : A \to B$ のとき, $f \circ id_A = f = id_B \circ f$.

3.8 単射 (一対一写像, one to one, injection)

関数 $f : A \to B$ が，以下の条件を満たすとき，単射という．

$$\forall x \in A \; \forall y \in A \; (f(x) = f(y) \Rightarrow x = y)$$

この条件は対偶を取って $\forall x \in A \; \forall y \in A \; (x \neq y \Rightarrow f(x) \neq f(y))$ とも書くことができる. つまり，A の異なる要素が B の異なる要素に写されるとき，単射という．

例 **46** (図 3.6)

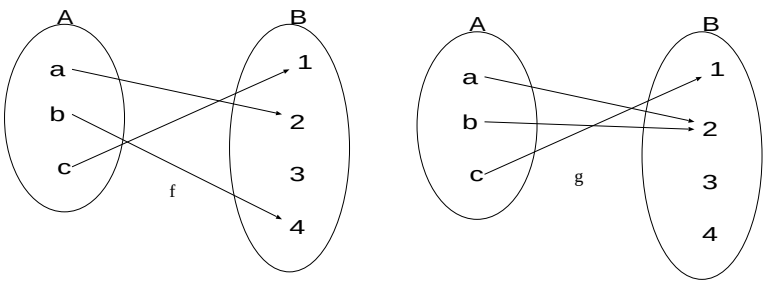


図 3.6: 単射 $f : A \to B$ と単射でない関数 $g : A \to B$.

例 **47** $f(x) = ax + b$ (ただし，$a \neq 0$ とする) で定義される関数 $f : \mathcal{R} \to \mathcal{R}$ は単射である.
$g(x) = ax^2 + bx + c$ (ただし，$a \neq 0$ とする) で定義される関数 g は単射でない.

例 **48** $A = \{6k + 4 \mid k \in \mathcal{N}\}$, $B = \{3k + 4 \mid k \in \mathcal{N}\}$, $f : A \to B$, $f(x) = x + 3$ と定義する. f は単射. なぜなら，$x \neq y \Rightarrow x + 3 \neq y + 3$.

3.9 全射 (上への写像，onto, surjection)

関数 $f : A \to B$ が，以下の条件を満たすとき，全射という．

$$\forall y \in B \ \exists x \in A \ f(x) = y$$

この条件は，$f(A) = B$ とも書くことができる．つまり，f による A の像が値域 B 全体になるとき，全射である．

例 **49** (図 3.7)

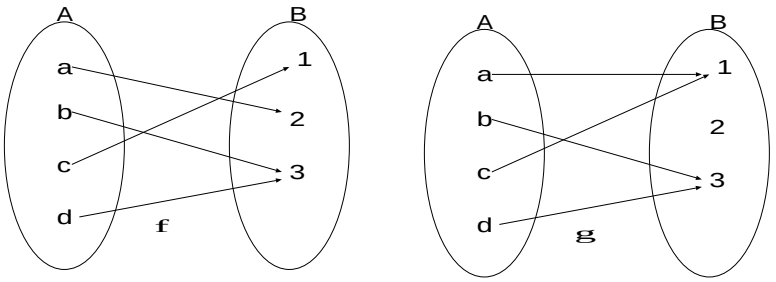


図 3.7: 全射 $f : A \to B$ と全射でない関数 $g : A \to B$.

例 **50** $f : A \times B \to A, f(x, y) = x$ となる関数 f は単射でないが全射である.

例 **51** $g : A \to A \times A, g(x) = \langle x, x \rangle$ となる関数 g は単射であるが全射でない.

3.10 全単射 (bijection) と逆関数 (inverse function)

単射かつ全射となる関数を全単射という．

$f : A \to B$ が全単射のとき，以下を満たす関数 $g : B \to A$ が存在する．g のことを f の逆関数という．

$$\forall x \in A \ \forall y \in B \ (f(x) = y \Leftrightarrow x = g(y))$$

このとき，$g \circ f = id_A$, $f \circ g = id_B$ が成立する．

例 **52** E, O を、それぞれ偶数の集合と奇数の集合とする．$f : O \to E$, $f(x) = x - 1$，$g : E \to O$, $g(x) = x + 1$ とする．

f, g は全単射である．g は f の逆関数で，f は g の逆関数となる．

3.11 部分関数 (partial function)

関数は，定義域の要素すべてに対して，対応する値が唯一に定まるものであった．この条件を緩めて，集合 A のすべての要素に対して，集合 B の要素がたかだか 1 つ[2]対応付けられる場合，この対応付けを部分関数という．f が集合 A から集合 B への部分関数であることを，$f; A \to B$ と書く．

[2]「たかだか 1 つ」というのは，0 個または 1 個という意味である．

例 **53** div を実数上の割り算をあらすとすると，div は $\mathcal{R} \times \mathcal{R}$ から $\mathcal{R}$ への部分関数である．すなわち，$div; R \times R \to R$ である．

割算 $div(x, 0)$ (ただし $x \in R$) に対して，対応する値がない．これを「未定義の値」という．

関数は部分関数であるが，部分関数は関数とは限らない．関数を部分関数と明示的に区別したいときに，特に，全域関数 (total function) と呼ぶことがある．

コンピュータのプログラムは，いくつかの入力を与えて，出力を返すものと考えられる．このとき，プログラムは，関数とは限らない．なぜなら，プログラムの実行は，入力の値によっては停止しない (無限に計算を続ける) ため，出力が得られないことがあるからである．このように，コンピュータのプログラムは，関数ではなく部分関数としてモデル化することができる．

この章の他の節で述べた定義・性質等は関数についてのものであったが，多くのものは，部分関数に対しても拡張可能である．たとえば，部分関数同士の等しさ，合成関数，像，逆像などを，関数に対するものと同様に定義することができる．

第4章　関係

4.1　関係

$A_1, \cdots, A_n$ を集合とするとき，$A_1 \times \cdots \times A_n$ の部分集合を $A_1, \cdots, A_n$ 上の n 項関係 (relation) と呼ぶ．すなわち，R が $A_1, \cdots, A_n$ 上の n 項関係であるとは，

$$R \subset A_1 \times \cdots \times A_n$$

であることである．$A_1 = \cdots = A_n$ のとき，単に「A_1 上の n 項関係」という．また，$n = 2$ のとき，「A_1 上の二項関係 (binary relation)」という．本講義資料では，主として，二項関係を扱う．

例 54 $A = \{a, b\}$, $B = \{1, 2, 3\}$ とする．$R_1 = \{\langle a, 1\rangle, \langle b, 2\rangle\}$ や $R_2 = \{\langle a, 1\rangle, \langle a, 2\rangle, \langle a, 3\rangle\}$ は二項関係である．

R が二項関係のとき，$\langle x, y\rangle \in R$ のことを xRy とも書く. 上の例では，aR_11 や aR_23 が成立する．

例 55 $A = \{1, 2, 3\}$ とする．以下の R_3, R_4 は A 上の二項関係である．

$$\begin{aligned} R_3 &= \{\langle 1,1\rangle, \langle 2,1\rangle, \langle 2,2\rangle, \langle 3,1\rangle, \langle 3,2\rangle, \langle 3,3\rangle\} \\ R_4 &= \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle\} \end{aligned}$$

R_3 は $\geq$ (以上) という関係を表し，R_4 は $=$ (等しい) という関係を表す．

4.2　二項関係の性質

R を集合 A 上の二項関係とする．すなわち，$R \subset A \times A$ である．
二項関係 R に対して，以下の性質は特に有用なので，名前がついている．

- R が反射的 (reflexive): $\forall x(xRx)$
- R が対称的 (symmetric): $\forall x \forall y(xRy \ \Rightarrow \ yRx)$
- R が推移的 (transitive): $\forall x \forall y \forall z((xRy \wedge yRz) \ \Rightarrow \ xRz)$
- R が反対称的 (antisymmetric): $\forall x \forall y((xRy \wedge yRx) \Rightarrow x = y)$

例 56 自然数の集合 $\mathcal{N}$上の二項関係として, $=, \leq, <, \neq, R, S$ を考える．ただし，R, S は以下で定義される関係である．

$$xRy \Leftrightarrow x \text{ と } y \text{ は } 7 \text{ で割った余りが同じ}$$

$$xSy \Leftrightarrow |x - y| \leq 1$$

これらの関係に対する性質を表にすると以下のようになる．

関係	反射的	対称的	推移的	反対称的
$=$	○	○	○	○
$\leq$	○	×	○	○
$<$	×	×	○	○
$\neq$	×	○	×	×
R	○	○	○	×
S	○	○	×	×

たとえば，関係 R が推移的であることの証明は以下のようになる．xRy かつ yRz とする．すると，$(x-y)\ mod\ 7=0$ かつ $(y-z)\ mod\ 7=0$ である．これらより $(x-z)\ mod\ 7=0$ となるので，xRz が成立する．

一方，関係 S は推移的ではない．$0S1$ かつ $1S2$ であるが，$0S2$ でない．

4.3 順序

集合 A 上の二項関係で，反射的かつ推移的かつ反対称的であるものを，A 上の順序 (order) という．なお，後で定義する全順序と区別するため，一般の順序のことを特に半順序 (partial order) ということがある．

例 **57** $\mathcal{N}$上の二項関係 $\geq$ と $\leq$ と $=$ はどれも順序である．

一方，$\mathcal{N}$上の二項関係 $<$ は反射的でないので，順序ではない．

例 **58** A を集合とするとき，2^A は A の部分集合からなる集合である．2^A の要素の間の包含関係 $\subset$ は順序である．$\subset$ が推移的であることは，$B\subset C$ かつ $C\subset D$ から $B\subset D$ が導けることからわかる．

最大，最小，極大，極小

集合 A 上の順序 R は一種の大小関係と見なすことができる．すなわち，xRy が成立するとき「x が R に関して y と等しいか小さい」と見なすことができる．このとき，以下の要素が定義される．

- 最大元 (maximum element) a とは $\forall x\in A(xRa)$ を満たす A の元のことである．
- 最小元 (minimum element) b とは $\forall x\in A(bRx)$ を満たす A の元のことである．
- 極大元 (maximal element) c とは $\forall x\in A(cRx\Rightarrow c=x)$ を満たす A の元のことである．
- 極小元 (minimal element) d とは $\forall x\in A(xRd\Rightarrow x=d)$ を満たす A の元のことである．

ただし，これらの元は存在しないこともある．最大元は極大元の 1 つであり，最小元は極小元の 1 つである．最大元，最小元は，存在すれば唯一だが，極大元，極小元は，1 つとは限らない．

例 **59** $\mathcal{N}$上の順序 $\leq$ に関する最大元，極大元は存在しない．最小元，極小元は 0 だけである．

例 **60** $A=\{1,2,3\}$ として，2^A 上の二項関係として，集合の包含関係を考えると，これは順序となる．

この順序に関する最小元は ϕ である．(2^A のどんな要素 S に対しても $\phi\subset S$ が成立するので．)

この順序に関する最大元は A である．(2^A のどんな要素 S に対しても $S\subset A$ が成立するので．)

例 **61** $A = \{1, 2, 3\}$ として，$2^A - \{A\}$ という集合上の二項関係として，集合の包含関係を考える．

今度は，A がはいっていないので，最大元は存在しない．(2^A のどんな要素 S に対しても $S \subset T$ が成立するような T は存在しない．) 極大元は，$\{1, 2\}, \{2, 3\}, \{3, 1\}$ の 3 つある．

全順序

集合 A 上の順序 R で，以下の条件を満たすものを全順序 (total order) という．

$$\forall x \in A \forall y \in A \ (xRy \ \vee \ yRx)$$

すなわち，任意の 2 つの要素が，R に関して比較可能 (どちらかが小さい) のとき，R を全順序という．先の例では，$\mathcal{N}$ 上の $\geq$ や $\leq$ は全順序であるが，$2^{\mathcal{N}}$ 上の包含関係 $\subset$ は全順序でない．たとえば，$\{1, 2\}$ と $\{2, 3\}$ は，$\subset$ では関係付けられない．

直積集合上の順序 (†)

R_1, R_2 をそれぞれ集合 A_1, A_2 上の順序とする．このとき，直積集合 $A_1 \times A_2$ 上の順序として以下の 2 つのものが考えられる[1]．

- 直積順序 P:

$$\langle x_1, y_1 \rangle \ P \ \langle x_2, y_2 \rangle \Leftrightarrow (x_1 R_1 x_2 \wedge y_1 R_2 y_2)$$

- 辞書式順序 L:

$$\langle x_1, y_1 \rangle \ L \ \langle x_2, y_2 \rangle \Leftrightarrow ((x_1 \neq x_2 \wedge x_1 R_1 x_2) \vee (x_1 = x_2 \wedge y_1 R_2 y_2))$$

R_1, R_2 がいずれも全順序であるとき，辞書式順序も全順序である．一方，R_1, R_2 が全順序であっても，直積順序は全順序とは限らない．国語辞典などの辞書では，全ての単語を一列に並べる必要があるため，辞書式順序を (任意の長さの文字列に対して拡張した) 全順序を用いている．

4.4 同値関係

集合 A 上の二項関係で，反射的, 対称的, 推移的であるものを A 上の同値関係 (equivalence relation) という. 同値関係は, 要素が等しいことを表す関係を一般化したものになっている.

例 **62** 自然数の集合 $\mathcal{N}$ 上の以下の二項関係は同値関係である．

$$\{\langle n, n \rangle \mid n \in \mathcal{N}\}$$

実数の集合 $\mathcal{R}$ 上の以下の二項関係は同値関係である．

$$\{\langle a, b \rangle \mid a, b \in \mathcal{R} \text{ かつ } |a| = |b|\}$$

一方，$\mathcal{N}$ 上の $\geq$ や $<$ は同値関係ではない．

[1] ここでは，見やすさのために，一番外側の $\forall x_1$ 等を省略した．正確に書くならば，$\forall x_1 \forall x_2 \forall y_1 \forall y_2$ を一番外に補う必要がある

同値類 (†)

R を集合 A 上の同値関係とする. $x \in A$ なる x に対して，x の同値類 $[x]$ とは，以下の集合のことである．

$$[x] = \{y \mid \ y \in A \ \wedge \ xRy\}$$

同値関係 R を「等しい」という概念と見なしたとき，R に関する同値類は x と等しい要素を全て集めた集合である．

商集合 (†)

集合 A 上の同値関係 R に対して，商集合 A/R は，同値類をすべて集めた集合である．

$$A/R = \{[x] \mid x \in A\}$$

例 **63** 自然数上の二項関係 R を次のように定義する.

$$a \, R \, b \Leftrightarrow (a \bmod 5 = b \bmod 5)$$

この時，同値類は以下のようになる.

$$[0] = \{0, 5, 10, 15, \ldots\}$$

$$[1] = \{1, 6, 11, 16, \ldots\}$$

$$[2] = \{2, 7, 12, 17, \ldots\}$$

$$[3] = \{3, 8, 13, 18, \ldots\}$$

$$[4] = \{4, 5, 14, 19, \ldots\}$$

$\mathcal{N}/R$ は, 以下の集合になる.

$$\mathcal{N}/R = \{[0], [1], [2], [3], [4]\}$$

4.5 関係の合成

R, S をそれぞれ A, B 上の二項関係，B, C 上の二項関係とする．すなわち，$R \subset A \times B, S \subset B \times C$ とする. R と S を合成した関係 $R \circ S$ は次のように定義される[2]．

$$x(R \circ S)y \ \Leftrightarrow \exists z \in B(xRz \wedge zSy)$$

例 **64** 以下の自然数上の関係 R, S を考える.

$$\begin{aligned} R &= \{\langle x, x+1\rangle \mid x \in \mathcal{N}\} \\ S &= \{\langle x, 2x\rangle \mid x \in \mathcal{N}\} \end{aligned}$$

このとき, 関係の合成, $R \circ R$, $R \circ S$, $S \circ R$ は以下のようになる.

$$\begin{aligned} R \circ R &= \{\langle x, x+2\rangle \mid x \in \mathcal{N}\} \\ R \circ S &= \{\langle x, 2(x+1)\rangle \mid x \in \mathcal{N}\} \\ S \circ R &= \{\langle x, 2x+1\rangle \mid x \in \mathcal{N}\} \end{aligned}$$

[2]なお，一部の教科書では，この定義の R と S を逆に置いた定義を採用しているものがあり，大変混乱させられる．参考書の [4] は良くまとまった本であるが『逆』の定義を採用しているから注意されたい．本書では，国際的に標準的な定義を採用した．

例 **65** $\mathcal{R}$ 上の二項関係 $>$ と $<$ に対して，

$$(<) \circ (>) = \mathcal{R} \times \mathcal{R}$$

なぜならば, 任意の x, y に対して, ある z が存在して, $x < z$ かつ $z > y$ となるように z を取ることができるから．($z = x + y + 1$ と取ればよい．)

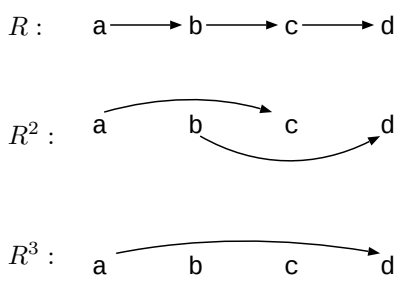
関係の合成に関して以下の性質 (結合律) が成り立つ.

$$R \circ (S \circ T) = (R \circ S) \circ T$$

定義 **2** R が A 上の二項関係で n が自然数の時,「R の (関係としての)n 乗」という関係 R^n が以下のように定義できる[3]．

$$\begin{aligned} R^0 &= \{\langle a, a\rangle \mid a \in A\} \\ R^{n+1} &= R^n \circ R = R \circ R^n \end{aligned}$$

例 **66** $R = \{\langle a, b\rangle, \langle b, c\rangle, \langle c, d\rangle\}$



4.6 閉包 (closure) (†)

閉包というのは,「何らかの性質に関して閉じたもの」という意味である．p を関係に関する性質とする. 関係 R の p に関する閉包とは，R を含み, p の性質を満たす最小の関係のことである．(ただし，p によっては，そのような最小の関係が存在しないこともある．)

p として,「反射的である」「対称的である」「推移的である」などの性質を取ることが多い．

- R の反射閉包 $r(R)$ とは R を含み，反射的である最小の関係．
- R の対称閉包 $s(R)$ とは R を含み，対称的である最小の関係．
- R の推移閉包 $t(R)$ とは，R を含み，推移的である最小の関係．

また,「最小の関係」というのは，関係は集合の一種なので，集合として一番小さいもの (集合の包含関係に関する最小元) という意味である．

例 **67** $A = \{a, b, c\}$, $R = \{\langle a, a\rangle, \langle a, b\rangle, \langle b, a\rangle, \langle b, c\rangle\}$ とする．

[3]ここで定義したものは，集合の章で定義した「集合としての n 乗」とは異なるものである．異なる概念に同じ記号を使うのは混乱の元であるので避けるべきであるが，ここでは伝統的表記に従った．

$$
\begin{aligned}
r(R) &= \{\langle a,a\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle b,c\rangle, \langle b,b\rangle, \langle c,c\rangle\} \\
s(R) &= \{\langle a,a\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle b,c\rangle, \langle c,b\rangle\} \\
t(R) &= \{\langle a,a\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle b,c\rangle, \langle a,c\rangle, \langle b,b\rangle\}
\end{aligned}
$$

また， $S = \{\langle a,b\rangle, \langle b,c\rangle, \langle c,d\rangle\}$ とすると，

$$
\begin{aligned}
r(S) &= \{\langle a,b\rangle, \langle b,c\rangle, \langle c,d\rangle, \langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle\} \\
s(S) &= \{\langle a,b\rangle, \langle b,c\rangle, \langle c,d\rangle, \langle b,a\rangle, \langle c,b\rangle, \langle d,c\rangle\} \\
t(S) &= \{\langle a,b\rangle, \langle b,c\rangle, \langle c,d\rangle, \langle a,c\rangle, \langle b,d\rangle, \langle a,d\rangle\}
\end{aligned}
$$

閉包の構成法

R を A 上の二項関係とすると，反射閉包，対称閉包，推移閉包は以下のようにして計算できる．

(a) $r(R) = R \cup R^0$

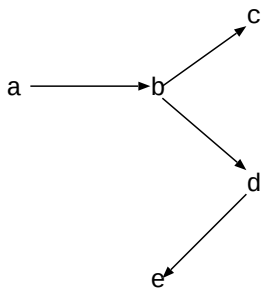
(b) $s(R) = R \cup R^c$
ただし R^c は，R の左右を逆にした関係であり R の逆関係と呼ばれる．

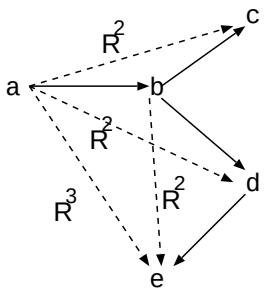
$$R^c = \{\langle x,y\rangle \mid \langle y,x\rangle \in R\}$$

(c) A が n 個の要素を持つ有限集合の時
$t(R) = R \cup R^2 \cup \cdots \cup R^n$

(d) A が無限集合のとき，R の推移閉包 $t(R)$ は以下のように，無限個の集合の和集合 (極限) として表される．
$t(R) = R \cup R^2 \cup R^3 \cup \cdots$

例 **68** $A = \{a,b,c,d,e\}$
$R = \{\langle a,b\rangle, \langle b,c\rangle, \langle b,d\rangle, \langle d,e\rangle\}$





この場合，$R^4 = R^5 = \phi$ である．

例 **69** $R = \{\langle x, x+1\rangle \mid x \in \mathcal{N}\}$ とすると，

$$
\begin{aligned}
R &= \{\langle x, x+1\rangle \mid x \in \mathcal{N}\} \\
R^2 &= \{\langle x, x+2\rangle \mid x \in \mathcal{N}\} \\
&\vdots \\
R^k &= \{\langle x, x+k\rangle \mid x \in \mathcal{N}\}
\end{aligned}
$$

従って，$t(R)$ は関係 $<$ (「より小さい」, "less-than") となる．

例 **70** $\mathcal{N}$ 上の二項関係 less-than (より小さい), not-equal (等しくない) を考える．それらの閉包は以下の通りである．

$$
\begin{aligned}
r(\texttt{less-than}) &= \{\langle x, y\rangle \mid x \leq y\} \\
s(\texttt{less-than}) &= \{\langle x, y\rangle \mid x \neq y\} \ (= \texttt{not-equal}) \\
t(\texttt{less-than}) &= \{\langle x, y\rangle \mid x < y\} \ (= \texttt{less-than}) \\
r(\texttt{not-equal}) &= \mathcal{N} \times \mathcal{N} \\
s(\texttt{not-equal}) &= \{\langle x, y\rangle \mid x \neq y\} \ (= \texttt{not-equal}) \\
t(\texttt{not-equal}) &= \mathcal{N} \times \mathcal{N}
\end{aligned}
$$

例 **71** インターネットの Web page 全体の集合を W とし，その上の二項関係 R として，以下のものを採用する．

$$aRb \ \Leftrightarrow \ \text{a から b へのリンクが } 1 \text{ 本以上ある．}$$

R に関する推移閉包 $r(R)$ を考えると，$\langle a, b\rangle \in r(R)$ は「a からリンクをたどって b に到達可能である」ことを表す．

第5章　グラフと木

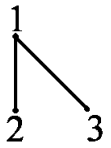
5.1　グラフ

グラフ (graph) は，いくつかの頂点 (点，vertex) と，それらを結ぶいくつかの辺 (edge) から構成された図形である．

グラフは，無向グラフ (辺が向きを持たないグラフ) と有向グラフ (辺が向きを持つグラフ) に大別される．

5.1.1　無向グラフ

無向グラフでは，各々の辺は 2 つの頂点を結びつける．たとえば，以下のグラフは 3 点 1,2,3 と，それらを結ぶ 2 本の辺からなる無向グラフである．



無向グラフ G は，頂点の集合 V と辺の集合 E の対として表現できる．[1]たとえば，上のグラフは以下の G として表現される．

$$G = \langle V, E \rangle$$
$$V = \{1, 2, 3\}$$
$$E = \{\{1, 2\}, \{1, 3\}\}$$

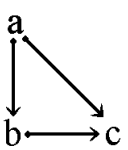
例 **72** 町と道路を想像せよ. 町が頂点で道路が辺である.

無向グラフにおいて，頂点 a の次数 (degree) とは，a と他の頂点を結ぶ辺の本数のことである．上の例では，頂点 1 の次数は 2, 頂点 2 および頂点 3 の次数は 1 である．

5.1.2　有向グラフ

辺に向きがついているグラフを，有向グラフ (directed graph) という.

例 **73** 3 つの頂点 a,b,c からなる有向グラフの例．

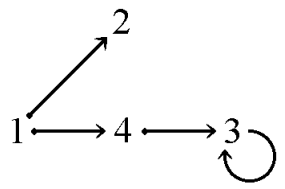


[1]この場合，同じ 2 点を結ぶ辺が複数あることはないと仮定している．用途によっては，同じ 2 点の間を結ぶ辺が 2 本以上あるグラフを考えることもある．

このグラフの場合，a から b には辺があるが，b から a への辺はない．a から b への辺に対して，頂点 a を始点といい，頂点 b を終点という．

有向グラフ G も，頂点の集合と辺の集合の対として表現できる. ただし，辺の表現において，始点と終点は区別しなければいけないので，始点と終点からなる集合ではなく，始点と終点の対 (つい) として表現する．

例 **74** 有向グラフとその表現．



$$\begin{aligned} G &= \langle V, E \rangle \\ V &= \{1, 2, 3, 4\} \\ E &= \{\langle 1, 2 \rangle, \langle 1, 4 \rangle, \langle 4, 3 \rangle, \langle 3, 3 \rangle\} \end{aligned}$$

E は $V \times V$ の部分集合であるので，V 上の二項関係である[2]．逆に，集合とその上の二項関係が与えられれば，それに対応する有向グラフが決まる．

有向グラフにおける頂点の次数は「入次数」と「出次数」の 2 通りある．頂点 a の入次数とは，a を終点とする (a にはいってくる) 辺の本数であり，出次数とは，a を始点とする (a から出ていく) 辺の本数である．

以下では，無向グラフと有向グラフを総称して「グラフ」という．

5.1.3 位数とサイズ

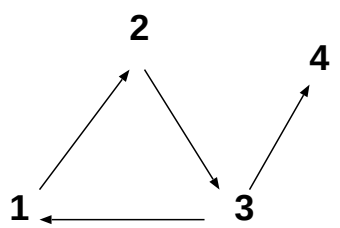
グラフ $G = \langle V, E \rangle$ に対して，頂点の数 (集合 V の要素数) を G の位数といい，辺の数 (集合 E の要素数) を G のサイズという．位数やサイズが無限大であるグラフを扱うこともある．

5.1.4 道 (パス) と閉路 (サイクル)

グラフにおいて，$\langle v_0, v_1 \rangle, \langle v_1, v_2 \rangle, \langle v_2, v_3 \rangle, \cdots, \langle v_{n-1}, v_n \rangle$ がすべて辺であるとき，これを頂点 v_0 から頂点 v_n への道 (パス, path) と言う．v_0 をこの道の始点，v_n を終点といい，n をこの道の長さという．

また，道を，辺の列ではなく，頂点の列 $\langle v_0, v_1, v_2 \cdots v_{n-1}, v_n \rangle$ として表現することもある．

例 **75** 次のグラフについて考える．



[2]この場合，同じ 2 点を結ぶ同一の方向の辺が複数あることはないと仮定している．

$\langle 1,2\rangle, \langle 2,3\rangle, \langle 3,4\rangle$ は頂点 1 から頂点 4 への道である．これを $\langle 1,2,3,4\rangle$ と書くこともある．この道の長さは 3 である．

一方，頂点 4 から頂点 1 への道は存在しない．

同じ辺が 2 回以上現れない道を単純道という．なお，グラフ理論の専門書によっては，ここで定義した「道」を「歩道」と呼び，「単純道」を「道」と呼ぶものもある．そのほか，細かな定義・用語が違うことがあるので注意されたい．

閉路 (サイクル, cycle) とは，始点と終点が同じ道のことである．先ほどの例では，$\langle 1,2\rangle, \langle 2,3\rangle, \langle 3,1\rangle$ という道は閉路になっている．

閉路を持たない有向グラフを非循環グラフ (directed acyclic graph, 略して dag) という．dag は応用上重要なグラフである．

5.1.5 グラフの同型

グラフ $G_1 = \langle V_1, E_1\rangle$ と $G_2 = \langle V_2, E_2\rangle$ が同型 (isomorphic) であるとは，頂点の名前を除いて 2 つのグラフが一致することである．

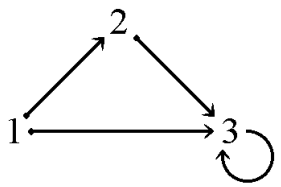
グラフの同型は，「関数」の言葉を使えば，以下のように厳密に定義できる (詳細は、第 3 章を参照のこと)．すなわち，関数 $f : V_1 \to V_2$ が存在して，以下の 2 つの条件が成立することである．

- f は全単射．
- $\forall x \in V_1.\forall y \in V_1.((\langle x, y\rangle \text{ が } E_1\text{の辺}) \Leftrightarrow (\langle f(x), f(y)\rangle \text{ が } E_2\text{の辺}))$

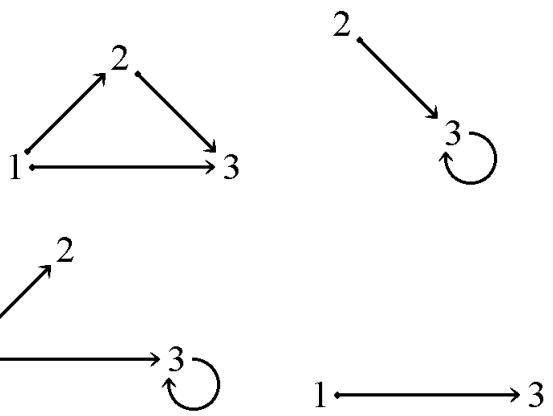
5.1.6 部分グラフ

グラフ $G = \langle V, E\rangle$ の部分グラフとは，$G' = \langle V', E'\rangle$ で，$V' \subset V$ かつ $E' \subset E$ となるグラフである．

例 **76** 次のグラフの部分グラフを考える.



このグラフの部分グラフの例としては下のようなものがある. (この他にもある．)



5.1.7　連結，連結成分 (†)

グラフが連結 (connected) であるとは，どの 2 つの頂点 a,b を取っても，a から b への道が存在することである．

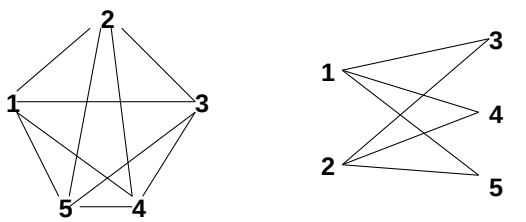
グラフ G の連結成分とは，G の極大な連結部分グラフのことである. すなわち，G の連結部分グラフ G' であって，G' を真に含む G の部分グラフで，連結なものが存在しないもののことである．

5.1.8　完全グラフ，完全 2 部グラフ (†)

無向グラフで，全ての頂点の間に辺があるグラフを完全グラフという．頂点数が n の完全グラフを K_n と書く．たとえば，K_5 は，頂点が 5 個, 辺が 10 本あるグラフである．

無向グラフにおいて，辺の集合が A と B に分割され，A に属する全ての頂点と B に属する全ての頂点の間に辺があり，それ以外には辺がないグラフを完全 2 部グラフという．A の頂点の数が n 個で，B の頂点の数が m 個であるような完全 2 部グラフを $K_{n,m}$ と書く．これは，頂点が $n+m$ 個で，辺が nm 本あるグラフである．

例 **77**



完全グラフ　　完全2部グラフ

5.1.9　グラフの応用 (†)

グラフは，コンピュータ科学で頻繁に使われる構造であり，グラフに対する効率のよいアルゴリズムの開発は重要な課題である. ここでは，無向グラフの一筆書きの例をあげる．

例 **78** [Königsberg の橋 (図 5.1)] 7 つの橋全部を一度ずつ渡って町の見物ができるか. Euler は，

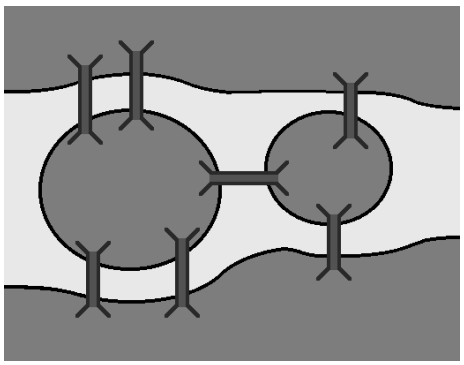
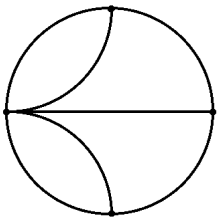


図 5.1: Königsberg の橋

このような道が存在しないことを示した. この研究がグラフ理論誕生の契機である．

Königsberg の橋の問題は，グラフの一筆書き問題の一例である．与えられたグラフに対して，そのグラフの全ての辺をちょうど 1 回だけ含む道があるとき，そのグラフを一筆書き可能という．

連結な無向グラフが一筆書き可能かどうかは，次数が奇数となる頂点の個数を調べることでわかり，そのような頂点が 2 個以下であるときに一筆書き可能，そうでないとき一筆書き不能である．Königsberg の橋をグラフとして表現すると，次のようになる。



次数が奇数の頂点が 4 個あるため一筆書き不能である．

5.2 木 (tree)

木はいろいろな定式化が可能である．ここでは無向グラフの一種としての定義を与える．

無向グラフで，以下の条件を満たすものを木という．

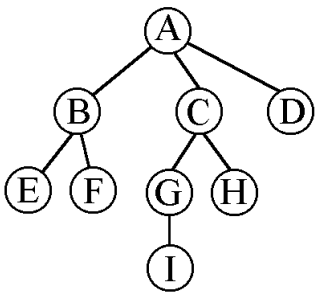
- 根 (root) と呼ばれる特別な頂点が 1 つだけある．
- 全ての頂点から根に至る単純道が唯一つ存在する．

この定義は以下の定義と同値である．

- 根 (root) と呼ばれる特別な頂点が 1 つだけある．
- 連結で閉路がない．

木を図形として表現するときは，通常，根を一番上に書く．

例 **79** A を根とする木



木の定義より，根以外の頂点に対して，根に至る単純道があり，その長さは 1 以上である．この道の長さを，その頂点の深さ (depth) という．この道の上で，頂点 v の次の頂点を v の親という．根以外の頂点に対してその親は一意的に定まる．上の例では E の深さは 2 で親は B, B の深さは 1 で親は A, A の深さは 0 で親は存在しない．

頂点 A が 頂点 B の親であるとき，B は A の子という．子のない頂点を葉 (leaf) という．上の例では，E, F, I, H, D が葉である．葉以外の頂点のことを節 (せつ，node)，あるいは，節点という．

また，ある頂点に対して，それと根を結ぶ道の上にある頂点をその頂点の祖先という．つまり，祖先とは，親，親の親，親の親の親，等の総称である．(『関係』の章で学習した言葉を使えば，「親である」という関係の逆関係が「子である」という関係であり，「親である」という関係の推移的閉包が「祖先である」という関係である．) 同様に子孫も定義される．同じ親をもつ頂点同士を兄弟 (あるいは姉妹) という．

木の高さ (height) とは，木に含まれる頂点の深さの最大値である．上の例の木では，その高さは 3 である．

木の中の全ての節点において，その子が n 個以下である場合，n 分木 (エヌぶんぎ) という．上の例は，各節点の子の数は 3 以下であるので，3 分木という．

木をいくつか集めたグラフを森 (forest) という．すなわち，連結成分が全て木になっているグラフを森という．

5.2.1 順序木

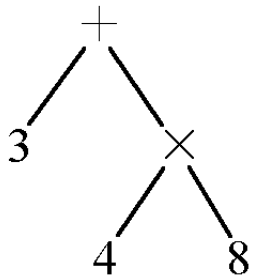
前節の木の定義では，1 つの節点に対する子の間に順序はなく，左右をひっくり返した木も同じであった．(左右をひっくり返したグラフも，グラフとしては同型である．)

しかし，コンピュータ科学における多くの応用では，子の間に左右の順序関係があった方が便利である．すなわち，先の例では，A の子である B, C, D の間には，「B が一番左の子であり，C が次に左の子であり，D が一番右の子である」というように順序付けられている方が都合がよい．この場合，C, B, D の順番に並べられている木とは異なる木であると考える．

このように，木において左右の順序関係を区別する場合を順序木という．コンピュータ科学では，ほとんどの場合，順序木を使うので，順序木のことを単に木と呼ぶことが多い．

順序木の応用として，コンピュータによる記号処理を考える．記号表現は，論理式や数式など，記号の列で表現されたものであり，木と見なすことができる．ここでは，数式が木で表わせることを見る．

例 **80** $3 + (4 \times 8)$ に対応する木.



木として表現することにより，$3 + (4 \times 8)$ と $(3 + 4) \times 8$ が異なる構造を持つことが明確になる．また，$3 + (4 \times 8)$ と $3 + (8 \times 4)$ は異なる表現として扱いたいので，単純な木ではなく，順序木として表現するのが普通である．

記号表現以外にも，コンピュータ科学では木の概念を使うことが多い．たとえば，1 つのコンピュータの中のファイルシステム (ファイルとディレクトリ全体) は木の構造を持つ．また，インターネットのドメイン名 (メールアドレスの @より後の部分) の全体も一種の木を構成している[3]．

[3] トップレベルのドメインが 1 つではないので，木ではなく森である，という見方もできる．

第6章　帰納的定義と帰納法

この章では，形式的な定義方法・推論方法としての「帰納」 (induction) について学ぶ．

よく知られた数学的帰納法 (mathematical induction) は，「全ての自然数に対して，性質 P が成立する」ことを示すための推論法である．これは，まず「P(0)」が成立することを示す，次に，「P(x) を仮定して，P(x+1) を示す」ことにより，「全ての自然数 n に対して，P(n) が成立する」ことが言える，という推論である．

しかし，帰納法は，自然数に対する推論法だけではない．自然数以外にも，リストや木などの集合を「帰納的に定義」することができ，それぞれに対する推論法としての帰納法がある．

実は，数学的に厳密な意味で，無限に多くの要素を持つ集合を作りだしたり，無限に多くの要素に対する性質を示す方法というのは，それほどたくさんあるわけではない．むしろ，そのような定義・推論方法は，ほとんどの場合，「帰納」によると言える．

6.1　帰納的に定義された集合

「自然数の集合」などの無限集合を厳密に定義するために，帰納的定義 (inductive definition) を用いる．

集合 A の帰納的定義とは，以下のように集合 A を定義する方法である．

- (basis，基礎) いくつかのもの (あらかじめわかっているもの) は，集合 A の要素であることを定める．
- (induction step，ステップ) すでに A の要素であることがわかっているものから，新たな A の要素を作る操作を定める．
- (closure，限定句) 上の操作を，有限回適用して作られた要素のみが A の要素であると定める．

なお，帰納的定義では，closure 条件を常に必要とするので，省略して書かないことも多い．

例 **81** 3 以上の奇数の集合 A の帰納的定義．

- $3 \in A$
- $x \in A \Rightarrow x + 2 \in A$
- 上記の操作で作られるものだけが A の要素である．(あるいは，A は上記を満たす最小の集合である．)

なお，3 番目の closure 条件がないと，集合 A が一意に定まらない．たとえば，自然数の集合 $\mathcal{N}$ も 1, 2 番目の条件を満たしている．「定義」であるためには，一意に定まらなければ意味がないため，帰納的定義においては，closure 条件の記述を省略してあっても必ず設定していると考える．

例 **82** 自然数の集合 $\mathcal{N}$.

- $0 \in \mathcal{N}$.
- $n \in \mathcal{N} \Rightarrow n+1 \in \mathcal{N}$

例 **83** $A = \{1, 3, 7, 15, 31, \ldots\}$

- $1 \in A$.
- $x \in A \Rightarrow 2x+1 \in A$

6.1.1 リスト

リスト (list) は，列 (sequence) とも呼ばれ，コンピュータ科学で頻繁に現れるデータ構造である．

リストの非形式的な (厳密でない) 定義: 集合 A の要素からなる任意の長さの組を A のリストという. すなわち，A のリストの集合 $List_A$ は以下のように定義される．

$$List_A = \{\langle x_1, \ldots, x_n\rangle \mid n \geq 0 \wedge \forall i.(1 \leq i \leq n \Rightarrow x_i \in A)\}$$

ただし，この定義は『. . .』を使っているので，厳密な定義ではない．

例 **84** 自然数のリスト:

$$\langle\rangle, \langle 1, 2\rangle, \langle 2\rangle, \langle 3, 2, 1\rangle$$

長さが 0 のリスト $\langle\rangle$ を空リストと呼ぶ．これを nil と書くことがある．

リストに対する基本的な操作として cons, head, tail がある.

$$\mathrm{cons}(x, \langle x_1, \ldots, x_n\rangle) = \langle x, x_1, \ldots, x_n\rangle$$

$$\mathrm{head}(\langle x_1, \ldots, x_n\rangle) = x_1$$

$$\mathrm{tail}(\langle x_1, \ldots, x_n\rangle) = \langle x_2, \ldots, x_n\rangle$$

head と tail は空リストに対しては定義できないので，これらは ($List_A$ を定義域と考えた場合) 関数ではなく部分関数である．

例 **85**

$$\mathrm{head}(\langle a, b, c\rangle) = a$$

$$\mathrm{tail}(\langle a, b, c\rangle) = \langle b, c\rangle$$

$$\mathrm{cons}(a, \langle\rangle) = \langle a\rangle$$

$$\mathrm{cons}(a, \langle b, c\rangle) = \langle a, b, c\rangle$$

任意の $x \in A$ と $L \in List_A$ に対して，$x = \mathrm{head}(\mathrm{cons}(x, L))$ と $L = \mathrm{tail}(\mathrm{cons}(x, L))$ が成り立つ. また，L が空でないリストのとき，$L = \mathrm{cons}(\mathrm{head}(L), \mathrm{tail}(L))$ が成り立つ.

上記の定義では，リストを組と考え，cons 等はそれに対する操作であったが，逆に，cons 操作を基本操作と考えることにより，リストの集合を帰納的に定義することができる．

リストの厳密な定義: 集合 A に対して，A のリストの集合 $List_A$ は，以下のように帰納的に定義される．

- $\langle\rangle \in List_A$

- $x \in A \wedge L \in List_A \Rightarrow \mathrm{cons}(x, L) \in List_A$

$\mathrm{cons}(x, L)$ に対する省略記法として，中置演算子 :: を使って $x :: L$ と書くことにする．

$$\begin{aligned} & a :: (b :: (c :: \langle\rangle)) \\ = \ & \mathrm{cons}(a, \mathrm{cons}(b, \mathrm{cons}(c, \langle\rangle))) \\ = \ & \mathrm{cons}(a, \mathrm{cons}(b, \langle c \rangle)) \\ = \ & \mathrm{cons}(a, \langle b, c \rangle) \\ = \ & \langle a, b, c \rangle \end{aligned}$$

例 **86** 0 と 1 が交互に現れるリストの集合 S の帰納的定義

- $\langle 0 \rangle \in S$.
- $\langle 1 \rangle \in S$.
- $L \in S \wedge \mathrm{head}(L) = 0 \Rightarrow \mathrm{cons}(1, L) \in S$.
- $L \in S \wedge \mathrm{head}(L) = 1 \Rightarrow \mathrm{cons}(0, L) \in S$

この定義により，$S = \{\langle 0 \rangle, \langle 1 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 0, 1 \rangle, \ldots\}$ となる．

本講義では，リストを組と考え $\langle 1, 2, 3 \rangle$ のように表現するが，プログラミング言語によって異なる表記が用いられる．

- Lisp，Scheme 言語におけるリスト: (1 2 3)
- ML 言語におけるリスト: $[1, 2, 3]$

6.1.2 文字列

文字列 (string) もコンピュータ科学において重要なデータ構造である．この講義では文字列を abc のように書く[1]．長さが 0 の文字列，すなわち，空文字列を Λ で表す.

文字 (アルファベット) の集合を Σ と書くと，Σ 上の文字列とは，Σ に含まれる文字の (有限長の) 列のことである.

例 **87** アルファベット $\{a, b\}$ 上の文字列.

$$\Lambda, a, b, aa, ab, ba, bb, \cdots$$

Σ 上の文字列の集合 Σ^* は，以下のように帰納的に定義できる．
(Σ 上の文字列の集合に対する定義 1)

- $\Lambda \in \Sigma^*$
- $x \in \Sigma \wedge s \in \Sigma^* \Rightarrow xs \in \Sigma^*$

文字列に対して，別の定義を考えることもできる．
(Σ 上の文字列の集合に対する定義 2)

[1] プログラミング言語では，文字列を表現する際は "abc" のように二重引用符で囲うことが多い．

- $\Lambda \in \Sigma^*$
- $x \in \Sigma \wedge s \in \Sigma^* \Rightarrow sx \in \Sigma^*$

(Σ 上の文字列の集合に対する定義 3)

- $\Lambda \in \Sigma^*$
- $x \in \Sigma \Rightarrow x \in \Sigma^*$
- $s \in \Sigma^* \wedge t \in \Sigma^* \Rightarrow st \in \Sigma^*$

これら 3 種類の定義は同等であることが証明できる．

定義 1・定義 2 においては，データが与えられたときに，それが Σ^* 上の文字列であることを導く方法は一意的である．たとえば，$bab \in \{a,b\}^*$ は次のように導ける．

- $\Lambda \in \{a,b\}^*$.
- これと $b \in \{a,b\}$ より，$b \in \{a,b\}^*$.
- これと $a \in \{a,b\}$ より，$ab \in \{a,b\}^*$.
- これと $b \in \{a,b\}$ より，$bab \in \{a,b\}^*$.

これ以外に $bab \in \{a,b\}^*$ を導く方法はない．

一方，定義 3 においては，$s,t,u \in \Sigma^*$ という 3 つの文字列に対して，それらを結合した stu が文字列であることを 2 通り (以上) の方法で導くことができる．

- 「$s \in \Sigma^*$」と「$t \in \Sigma^*$」から「$st \in \Sigma^*$」を導き，これと「$u \in \Sigma^*$」から，「$stu \in \Sigma^*$」を導く．
- 「$t \in \Sigma^*$」と「$u \in \Sigma^*$」から，「$tu \in \Sigma^*$」を導き，これと「$s \in \Sigma^*$」から，「$stu \in \Sigma^*$」を導く．

このように同一のデータに対して複数の導出がある事を「曖昧な導出を持つ」と言う．曖昧な導出がある場合は，$(ab)c$ や $a(bc)$ のように括弧をつけて区別する必要がある[2]．

例 **88** $A = \{0,1\}$ 上の文字列で，右端の文字のみが 0 である文字列の集合 L の帰納的定義．

- $0 \in L$.
- $s \in L \Rightarrow 1s \in L$.

例 **89** $S = \{a, b, ab, ba, aab, bba, aaab, bbba, \ldots\}$ の帰納的定義．

- $a \in S$, $b \in S$.
- $$s \in S \Rightarrow \begin{cases} bs \in S & (s = a \text{ の時}) \\ as \in S & (s = b \text{ の時}) \\ as \in S & (s \neq a \text{ かつ } strhead(s) = a \text{ の時}) \\ bs \in S & (s \neq b \text{ かつ } strhead(s) = b \text{ の時}) \end{cases}$$

ここで $strhead$ は文字列の先頭の文字を取る操作 (部分関数) とする．

[2]括弧をつけない表記を用いる場合は，どちらの省略形であるかを決める必要がある．論理式の場合，たとえば，$A \wedge B \wedge C$ は $(A \wedge B) \wedge C$ の省略である，と定めるのと同様である．

6.1.3 2 分木 (binary Tree)

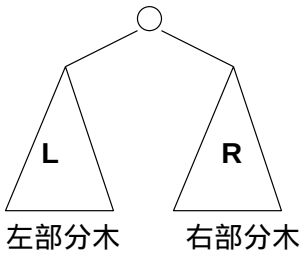
以前の章で，2 分木をグラフの一種として定義した．本節では，順序木としての 2 分木 (兄弟の間の順序を考慮に入れた 2 分木) の集合を帰納的に定義する．本節では，順序木しか扱わないので，単に 2 分木といえば順序木としての 2 分木である．

まず，単純な 2 分木，すなわち，2 分木の頂点にラベルがついていない 2 分木を定義する．葉を $\circ$ と表す．

定義 **3** [単純な 2 分木 BinTree]

- $\circ \in \mathrm{BinTree}$
- $(L \in \mathrm{BinTree} \wedge R \in \mathrm{BinTree}) \Rightarrow \langle L, R\rangle \in \mathrm{BinTree}$.
- $L \in \mathrm{BinTree} \Rightarrow \langle L\rangle \in \mathrm{BinTree}$.

  最後の項は，子を 1 つだけ持つ節に対応する．(2 分木では，それぞれの節の子が 0 ~ 2 個であるので，子が 1 個だけの節の場合も考慮する必要がある．)



ここで $\langle L, R\rangle$ は，2 分木 L と R を組み合わせて作った 2 分木である．この定義では，兄弟の順序関係を区別するため，$\langle L, R\rangle$ というように対を使っている．もし，兄弟の順序関係を区別しないのであれば $\{L, R\}$ と集合を使えばよい．木 $\langle L, R\rangle$ において，L のことを左部分木，R のことを右部分木という．

次に，ラベルつき 2 分木を定義する．すなわち，ラベル (名前) の集合を A とする時，2 分木の節 (葉でない頂点) に集合 A の要素が 1 つ付けられている 2 分木を定義する．

- $\circ \in \mathrm{BinTree}_A$
- $(x \in A \wedge L \in \mathrm{BinTree}_A \wedge R \in \mathrm{BinTree}_A) \Rightarrow \langle L, x, R\rangle \in \mathrm{BinTree}_A$.

  ここで x がこの節のラベル，L が左の部分木，R が右の部分木を表している.
- $(x \in A \wedge L \in \mathrm{BinTree}_A) \Rightarrow \langle L, x\rangle \in \mathrm{BinTree}_A$.

  これは，子が 1 つしかない節に対応している．

例 **90** $\mathrm{BinTree}_{\mathcal{N}}$ の要素 (自然数のラベルがつけられた 2 分木)

$$\circ$$

$$\langle \circ, 1, \circ\rangle$$

$$\langle\langle \circ, 2, \circ\rangle, 1, \circ\rangle$$

$$\langle\langle \circ, 2\rangle, 1, \circ\rangle$$

以下では，ラベルつき 2 分木を表すとき，$\langle L, x, R\rangle$ の代わりに $\mathrm{Tree}(L, x, R)$ と書き，$\langle L, x\rangle$ の代わりに $\mathrm{Tree}(L, x)$ と書く.

例 **91**

$$\mathrm{Tree}(\mathrm{Tree}(\circ, 2, \circ), 1, \circ)$$

$$\mathrm{Tree}(\mathrm{Tree}(\circ, 2), 1, \circ)$$

リストに対する head, tail と同様に，ラベルつき 2 分木に対する操作を以下のように定義する.

$$\mathrm{label}(\mathrm{Tree}(t_1, x, t_2)) = x$$

$$\mathrm{label}(\mathrm{Tree}(t_1, x)) = x$$

$$\mathrm{left}(\mathrm{Tree}(t_1, x, t_2)) = t_1$$

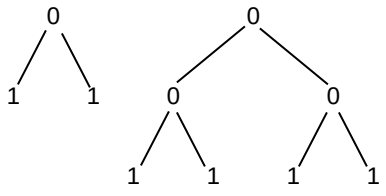
$$\mathrm{left}(\mathrm{Tree}(t_1, x)) = t_1$$

$$\mathrm{right}(\mathrm{Tree}(t_1, x, t_2)) = t_2$$

これらの操作は根だけからなる木に対しては定義されない．また right 操作は，根の子が 1 つしかない木に対しては定義されない．

例 **92** $\mathrm{BinTree}_A$ のうち左右の部分木が同じ木の集合 T_A は以下のように定義できる．

- $\circ \in T_A$.
- $x \in A \wedge t \in T_A \Rightarrow \mathrm{Tree}(t, x, t) \in T_A$.

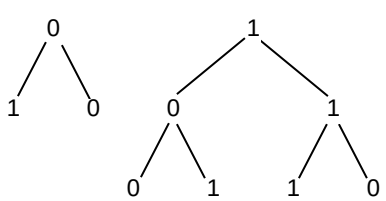


(葉を省略した．)

例 **93** $\{0, 1\}$ の要素をラベルとして持つラベルつき 2 分木で，左右の部分木が同じ形をしているが，ラベルの $0, 1$ が入れ替わっている木の集合 $Opps$.

- $\circ, \langle\circ, 0, \circ\rangle, \langle\circ, 1, \circ\rangle \in Opps$.
-

$$x \in \{0,1\} \ \wedge \ t \in Opps \Rightarrow \begin{cases} \mathrm{Tree}(t, x, \mathrm{Tree}(\mathrm{right}(t), 1, \mathrm{left}(t))) \in Opps, & \text{if } \mathrm{label}(t) = 0 \\ \mathrm{Tree}(t, x, \mathrm{Tree}(\mathrm{right}(t), 0, \mathrm{left}(t))) \in Opps, & \text{if } \mathrm{label}(t) = 1 \end{cases}$$



(葉を省略した．)

6.1.4 BNF 記法 (†)

BNF 記法 (Backus-Naur Form または Backus Normal Form) は，言語の構文を簡潔に定義する方法の 1 つであり，特に，プログラミング言語の構文を記述する際によく用いられる．形式言語理論の言葉では，BNF 記法は文脈自由文法という種類の言語を定義するものであるが，これは帰納的定義の一種と見なすことができる．

ここでは，一般的な BNF 記法を定義するかわりに，簡単なプログラミング言語の構文の定義の例を与える．

例 **94** [簡単なプログラム言語の構文]

変数の集合と定数の集合はあらかじめ定まっているものとする．たとえば，$\{a, b, c, \cdots, z\}$ 上の文字列を変数とし，$\{0, 1, 2, \cdots, 9\}$ 上の文字列を定数とする．この時，以下の BNF 記法によって，式 (expression) の集合を帰納的に定義することができる．

$$\langle\, \text{式} \,\rangle \quad ::= \quad \langle\, \text{変数} \,\rangle \mid \langle\, \text{定数} \,\rangle \mid \langle\, \text{式} \,\rangle\texttt{+}\langle\, \text{式} \,\rangle \mid \langle\, \text{式} \,\rangle\texttt{-}\langle\, \text{式} \,\rangle$$

$\langle$ 式 $\rangle$ や $\langle$ 変数 $\rangle$ は，それぞれ，式の集合の要素，変数の集合の要素を表す．縦棒 $\mid$ は，帰納的定義の定義節 (basis や induction step) の区切りを表す．上記の BNF は以下の帰納的定義を表している．

- 変数は式である．
- 定数は式である．
- x が式で y が式ならば x+y は式である．
- x が式で y が式ならば x-y は式である．

たとえば，abc+231-50 が式となる．

BNF の例をもう 1 つ与える．

$$\begin{aligned}
\langle\, \text{文} \,\rangle \quad &::= \quad \langle\, \text{変数} \,\rangle\texttt{:=}\langle\, \text{式} \,\rangle \mid \texttt{begin}\ \langle\, \text{文の列} \,\rangle\ \texttt{end} \\
&\qquad\quad \mid \texttt{while}\ \langle\, \text{式} \,\rangle\ \texttt{do}\ \langle\, \text{文} \,\rangle \mid \texttt{if}\ \langle\, \text{式} \,\rangle\ \texttt{then}\ \langle\, \text{文} \,\rangle\ \texttt{else}\ \langle\, \text{文} \,\rangle \\
\langle\, \text{文の列} \,\rangle \quad &::= \quad \langle\, \text{文} \,\rangle \mid \langle\, \text{文の列} \,\rangle\texttt{;}\langle\, \text{文} \,\rangle
\end{aligned}$$

これは while プログラムと呼ばれる簡単なプログラム言語の文 (statement) の定義を与えるものである．ここで「文」と「文の列」という 2 つの集合を同時に帰納的に定義している．たとえば，以下のものが「文の列」になる．

$$x\texttt{:=}3;\ y\texttt{:=}0;\ \texttt{while}\ x\ \texttt{do begin}\ y\texttt{:=}y\texttt{+}3; x\texttt{:=}x\texttt{-}1\ \texttt{end}$$

6.2 帰納的に定義された関数

前節では，リストを操作する部分関数 head, tail を用いたが，これらは非形式的に意味を与えただけであり，厳密な定義は与えていなかった．本節では，リストや木など帰納的に定義された集合を定義域とする関数 (または部分関数) を定義する方法を与える．

定義 **4** [関数 $f : A \to B$，ただし A は帰納的に定義された集合]

- (basis) A の帰納的定義の basis で与えた要素 x に対して，$f(x)$ の値を B の要素となるように定める．
- (induction step) A の帰納的定義の induction step が「$a \in A$ から $b \in A$ を作る操作」であったとすると，$f(a)$ の値を用いた式で $f(b) \in B$ の値を定める．

集合 A の帰納的定義に曖昧さがなければ，このように定義することにより，f は関数となることが保証される．すなわち，$f : A \to B$ である．

自然数の集合 $\mathcal{N}$ に対して，$f : \mathcal{N} \to B$ となる関数 f は次のように帰納的に定義される．

- $0 \in \mathcal{N}$ に対して $f(0)$ の値を定める．
- $n \in \mathcal{N}$ に対して，$f(n)$ の値を用いて，$f(n+1)$ の値を定める．

このことを，以下のように表記する．

$$f(n) = \begin{cases} \cdots & (n = 0 \text{ の時}) \\ \cdots f(m) \cdots & (n = m+1 \text{ の時}) \end{cases}$$

ここで，$n = m + 1$ のときに $f(m)$ 以外の f の値を使ってはいけない (後述する拡張の場合を除く)．たとえば，$f(n+1)$ の値を使って

$$f(n) = \begin{cases} 0 & (n = 0 \text{ の時}) \\ f(n+1) + 3 & (n = m+1 \text{ の時}) \end{cases}$$

としたものは，帰納的定義にならない[3]．

例 **95** $f : \mathcal{N} \to \mathcal{N}$, $n \in \mathcal{N}$ に対して 1 から $2n+1$ までの奇数の和を求める関数 $f(n)$ を定義しよう．非形式的に考えると，$n \geq 1$ ならば

$$\begin{aligned} f(n) &= 1 + 3 + \cdots + (2(n-1)+1) + (2n+1) \\ &= f(n-1) + (2n+1) \end{aligned}$$

である．これをもとにして，以下の帰納的定義で f を定義できる．

$$f(n) = \begin{cases} 1 & (n = 0 \text{ の時}) \\ f(m) + 2n + 1 & (n = m+1 \text{ の時}) \end{cases}$$

例 **96** 加算 $\mathrm{add} : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$.

加算は引数が 2 つあり，両方とも帰納的に定義された集合 $\mathcal{N}$ の要素であるので，加算を 2 通りに定義することができる．

$$\mathrm{add}(n, m) = \begin{cases} m & (n = 0 \text{ の時}) \\ \mathrm{add}(p, m) + 1 & (n = p+1 \text{ の時}) \end{cases}$$

$$\mathrm{add}(n, m) = \begin{cases} n & (m = 0 \text{ の時}) \\ \mathrm{add}(n, p) + 1 & (m = p+1 \text{ の時}) \end{cases}$$

厳密に言うと，加算の定義の右辺に現れる $+1$ は加算ではなく，自然数の帰納的定義に現れる $+1$ (ある自然数から次の自然数を構成する操作) である．

[3] 多くのプログラミング言語では $f(n) = \cdots f(n+1) \cdots$ のように f を定義する式の右辺で f を自由に使ってよい．すなわち，再帰的関数 (recursive function) を自由に定義して使うことが許されている．一方，本章で述べている「帰納的に定義された関数」(inductively defined function) は，再帰的関数の一種であるが，右辺で使ってよい f の値が制限されている点が異なる．

例 97 乗算 $\mathrm{times} : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$.

乗算も同様に 2 通りの定義が考えられるが，ここでは 1 つのみ与える．

$$\mathrm{times}(n, m) = \begin{cases} 0 & (n = 0 \text{ の時}) \\ \mathrm{times}(p, m) + m & (n = p + 1 \text{ の時}) \end{cases}$$

例 98 階乗 $\mathrm{factorial}(n)$.

$$\mathrm{factorial}(n) = \begin{cases} 1 & (n = 0 \text{ の時}) \\ n \times \mathrm{factorial}(m) & (n = m + 1 \text{ の時}) \end{cases}$$

例 99 引数の長さに応じたリストを生成する関数 $f : \mathcal{N} \to List_{\mathcal{N}}$ で $f(n) = \langle n, \ldots, 1, 0 \rangle$ となるもの．

$$f(n) = \begin{cases} \langle 0 \rangle & (n = 0 \text{ の時}) \\ \mathrm{cons}(n, f(p)) & (n = p + 1 \text{ の時}) \end{cases}$$

$$\begin{aligned} f(3) &= \mathrm{cons}(3, f(2)) \\ &= \mathrm{cons}(3, \mathrm{cons}(2, f(1))) \\ &= \mathrm{cons}(3, \mathrm{cons}(2, \mathrm{cons}(1, f(0)))) \\ &= \mathrm{cons}(3, \mathrm{cons}(2, \mathrm{cons}(1, \langle 0 \rangle))) \\ &= \langle 3, 2, 1, 0 \rangle \end{aligned}$$

ここまでの定義では，$n = 0$ の場合と $n \geq 1$ の場合に分けたが，その変形・拡張として $n = 0, 1, \cdots, k$ の場合と $n > k$ の場合に分ける等が考えられる．この場合，$f(n)$ の定義において $m < n$ となる $f(m)$ の値を使ってもよい．

例 100 Fibonacci 数列 $\mathrm{fib}(n)$ は以下のように帰納的に定義される．

$$\mathrm{fib}(n) = \begin{cases} 1 & (n = 1, 2 \text{ の時}) \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & (n > 2 \text{ の時}) \end{cases}$$

例 101 リストを操作する部分関数 $\mathrm{head}; List_A \to A$, $\mathrm{tail}; List_A \to List_A$.

$$\mathrm{head}(x) = \begin{cases} \text{未定義} & (x = \langle\rangle \text{ の時}) \\ y & (x = \mathrm{cons}(y, L) \text{ の時}) \end{cases}$$

$$\mathrm{tail}(x) = \begin{cases} \text{未定義} & (x = \langle\rangle \text{ の時}) \\ L & (x = \mathrm{cons}(y, L) \text{ の時}) \end{cases}$$

例 102 リストの長さ $\mathrm{length} : List_A \to \mathcal{N}$.

$$\mathrm{length}(x) = \begin{cases} 0 & (x = \langle\rangle \text{ の時}) \\ 1 + \mathrm{length}(L) & (x = \mathrm{cons}(y, L) \text{ の時}) \end{cases}$$

この定義の 2 行目は，$1 + \mathrm{length}(\mathrm{tail}(x))$ と書いても同じである．

例 103 リストの連結 (concatenation) $\oplus : List_A \times List_A \to List_A$.

$$x \oplus y = \begin{cases} y & (x = \langle\rangle \text{ の時}) \\ \mathrm{cons}(z, L \oplus y) & (x = \mathrm{cons}(z, L) \text{ の時}) \end{cases}$$

例 **104** 2 分木の節の数を数える関数 $\mathrm{nodes} : \mathrm{BinTree}_A \to \mathcal{N}$.

$$\mathrm{nodes}(x) = \begin{cases} 0 & (x = \langle\rangle \text{ の時}) \\ 1 + \mathrm{nodes}(L) + \mathrm{nodes}(R) & (x = \mathrm{Tree}(L, y, R) \text{ の時}) \end{cases}$$

例 **105** 2 つの文字列を結合する関数 $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$. ここでは，$\Sigma^*$ は「文字列の定義 1」により定義されているとする．

$$s \cdot t = \begin{cases} t & (s = \Lambda \text{の時}) \\ x\ (u \cdot t) & (s = xu \text{ の時}) \end{cases}$$

この定義で，$s, t, u \in \Sigma^*$ であり $x \in \Sigma$ である．「文字列の定義 1」では，$x \in \Sigma$ と $u \in \Sigma^*$ から $s = xu$ となる文字列 s を構成したので，それに対する関数も上記の形を取る．もし，「文字列の定義 2」を採用した場合は，以下のように定義すべきである．

$$s \cdot t = \begin{cases} s & (t = \Lambda \text{の時}) \\ (s \cdot u)\ x & (t = ux \text{ の時}) \end{cases}$$

6.3 帰納法による証明

帰納的に定義された集合 A に対して，A のすべての要素に対してある性質 P が成立することを証明するための方法が帰納法 (induction) である．

6.3.1 数学的帰納法

自然数に対する帰納法が数学的帰納法である．すべての自然数 x に対して, $P(x)$ が成り立つことを示すために，以下の 2 つのことを示せばよい．

- (base case) $P(0)$ が成り立つことを示す.
- (induction step) どんな $x \in \mathcal{N}$ に対しても，$P(x)$ が成り立つことを仮定[4]して，$P(x+1)$ が成り立つことを示す.

例 **106** 定理 「0 から n までの数の平方 (2 乗) の和は $\frac{n(n+1)(2n+1)}{6}$ である」
証明: $P(n)$ を上記定理中の「 ...」という命題とする．

- $P(0)$ は「$0^2 = \frac{0 \cdot 1 \cdot 1}{6}$」となるので正しい．
- $P(n)$ が正しいと仮定する．0 から $n+1$ までの数の平方の和は

  $$\frac{n(n+1)(2n+1)}{6} + (n+1)^2 \quad = \quad \frac{(n+1)(n+2)(2n+3)}{6}$$

  となるので，$P(n+1)$ も正しい．
- 以上より，どんな $n \in \mathcal{N}$ に対しても $P(n)$ は正しい．

基本的な数学的帰納法は，$\forall n \in \mathcal{N}.P(n)$ を示すためのものであったが，様々な形に拡張して利用されることが多い．

[4]この仮定のことを帰納法の仮定 (induction hypothesis) という．

- n が動く範囲が 0 から始まらない時:

  $n \geq k$ なる任意の自然数に対して $P(n)$ が正しいことを示すためには，base case として $P(k)$ を示し，induction step として $n > k$ なる任意の自然数 n に対して，$P(n)$ を仮定して $P(n+1)$ が成立することを示せばよい．

- n が動く範囲が飛び飛びの時:

  任意の正の奇数に対して $P(n)$ が正しいことを示すためには，base case を $n = 1$ とし，induction step では，$P(2k+1)$ を仮定して $P(2k+3)$ を示せばよい．

- 帰納法の仮定を強めたい時:

  induction step では，$P(n-1)$ だけでなく $0 \leq k < n$ なる任意の k に対して $P(k)$ が成り立つことを仮定して $P(n)$ を証明してもよい．

例 **107** 関数 fib と任意の正の整数 $n > 0$ に対して，

$$fib(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

が成り立つことを証明する．

与えられた式の右辺を $g(n)$ とおく．すなわち，

$$g(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

である．

正の整数 n に関する数学的帰納法を使って $\forall n.(n > 0 \Rightarrow fib(n) = g(n))$ を示す.

- $n = 1$ のとき

$$g(1) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\right) = 1 = fib(1)$$

- $n = 2$ のとき

$$g(2) = \frac{1}{\sqrt{5}}\left(\left(\frac{6+2\sqrt{5}}{4}\right) - \left(\frac{6-2\sqrt{5}}{4}\right)\right) = 1 = fib(2)$$

- $n > 2$ のとき

  $k < n$ となる任意の正整数 k に対して $fib(k) = g(k)$ が成り立つと仮定する. $n-2$ と $n-1$ は正整数だから $k = n-2$ と $k = n-1$ に対してもこの式が成立する．このとき，

$$\begin{aligned}
fib(n) &= fib(n-2) + fib(n-1) \\
&= g(n-2) + g(n-1) \\
&= \frac{1}{\sqrt{5}}\left(\left(\frac{3+\sqrt{5}}{2}\right)\left(\frac{1+\sqrt{5}}{2}\right)^{n-2} - \left(\frac{3-\sqrt{5}}{2}\right)\left(\frac{1-\sqrt{5}}{2}\right)^{n-2}\right) \\
&= \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right) \\
&= g(n)
\end{aligned}$$

6.3.2 リストに関する帰納法

リストに対する帰納法は，以下の形を取る．すべての $L \in List_A$ に対して, $P(L)$ が成り立つことを示すには以下の 2 つを証明すればよい.

- $P(\langle\rangle)$ が成り立つを示す.
- 任意の $L \in List_A$ と任意の $a \in A$ に対して，$P(L)$ が成り立つことを仮定して $P(a::L)$ が成り立つことを示す.

例 **108** 任意の $x, y \in List_A$ に対して以下の等式が成立することを証明する．

$$\mathrm{length}(x \oplus y) = \mathrm{length}(x) + \mathrm{length}(y)$$

(証明) リスト x に関する帰納法を使う．つまり，全ての $x \in List_A$ に対して「$\forall y \in List_A.\mathrm{length}(x \oplus y) = \mathrm{length}(x) + \mathrm{length}(y)$ 」を証明する[5]．この「 ...」を命題 $P(x)$ とする．

- (base case) $x = \langle\rangle$ の時. 任意の $y \in List_A$ を取る．

  $$\mathrm{length}(\langle\rangle \oplus y) = \mathrm{length}(y) = \mathrm{length}(\langle\rangle) + \mathrm{length}(y)$$

  となって，命題 $P(x)$ は成立する．

- (induction step) $x = a :: z$ の時. 任意の $y \in List_A$ を取る．

  $$\begin{aligned} (\text{左辺})\ \mathrm{length}((a :: z) \oplus y) &= \mathrm{length}(a :: (z \oplus y))\ (\oplus \text{ の定義による}) \\ &= \mathrm{length}(z \oplus y) + 1\ (\mathrm{length} \text{ の定義による}) \\ &= \mathrm{length}(z) + \mathrm{length}(y) + 1\ (\text{帰納法の仮定，すなわち } P(z) \text{ による}) \end{aligned}$$

  $$(\text{右辺})\ \mathrm{length}(a :: z) + \mathrm{length}(y) = \mathrm{length}(z) + 1 + \mathrm{length}(y)\ (\mathrm{length} \text{ の定義による})$$

  となって，命題 $P(a :: z)$ は成立する．

  以上より，全ての $x \in List_A$ に対して $P(x)$ は成立する．

帰納法による証明は，似て非なる式がいくつも出てくるので，どの定義の形にあてはめて変形を行っているかを明示的に書くようにするとよい．たとえば，帰納法で証明したい命題と，帰納法の仮定は，命題の形そのものは全く同じで，対象となるデータが少し違うだけであり，間違えやすい[6]．上記の証明では，等式変形のたびに，その根拠を右端に記述した．このような習慣をつけることにより，間違いを減らすことができ，また，あとで証明の正しさをチェックしやすくなる．

[5] このように，証明したい式に x と y という 2 つのリストが現れているときは，x に関する帰納法を使う場合と y に関する帰納法を使う場合がある．どちらを使うと，うまく行くかは問題によるので人間が考える必要がある．

[6] よくある間違いは，証明したい論理式を仮定してしまうというものである．

6.3.3　2 分木に関する帰納法 (†)

すべての $x \in \mathrm{BinTree}_A$ に対して, $P(x)$ が成り立つことを示すには以下の 2 つを言えばよい．

- $P(\circ)$ が成り立つを示す.
- 任意の $x, y \in \mathrm{BinTree}_A$ と，任意の $a \in A$ に対して，$P(x)$, $P(y)$ が成り立つことを仮定して，$P(\mathrm{Tree}(x, a, y))$ が成り立つことを示す.

ここまでに述べた自然数，リスト，木に対する帰納法はすべて，帰納的定義の構造に沿ったものであるため，構造帰納法 (structural induction) と呼ばれる．

関連図書

離散数学に関する書籍

[1] J. L. Hein, “Discrete Structures, Logic, and Computability” Second Edition, Jones and Bartlett Publishers International, 2002. かなり分厚い「アメリカらしい」教科書であり，離散構造以外の分野も含んでいる．本講義資料の内容は，この書籍の前半 1/3 程度を縮めたものに相当する．

[2] R. Haggarty, “Discrete Mathematics for Computing”, Addision Wesley, 2002. 上記 [1] よりだいぶ薄くて要領よく書かれているので読みやすい．

[3] P. Grossman, “Discrete Mathematics for Computing”, Second Edition, Macmillan, 2002. 上記 [2] と同様に薄くて読みやすい．

[4] 守屋悦朗 ,「コンピュータサイエンスのための離散数学」, サイエンス社, 1992. 本講義資料と同じレベルから，やや進んだレベルまで広くカバーしている．

[5] 小野寛晰 ,「情報代数」, 共立出版情報数学講座 2, 1994. 表題は「情報代数」であるが，前半は離散構造に関する良い入門書となっている．後半は，本講義資料では全く取り扱わなかった代数系の話を扱っている．

[6] 徳永豪 ,「工学基礎– 離散数学とその応用」, 数理工学社，2003．比較的最近出版された本であり，本講義資料で扱わなかった話題が多数載っている．数学的な話が好きな人には大変興味深いだろう．

グラフ，論理など離散数学の個別の話題に関する書籍

[7] 根上生也 ,「離散構造」, 共立出版情報数学講座 1, 1993. 表題は「離散構造」という一般的なものであるが，グラフ理論に特化した内容である．

[8] 小野寛晰 ,「情報科学における論理」, 日本評論社, 1994. 日本を代表する論理学者が，情報科学を学ぶ学生向けにわかりやすく書いた論理の本である．