目次

第 I 部

論理と集合

第 1 章

一階論理の形式化

まず，数理論理学で扱われる形式である一階述語論理と呼ばれるデータ構造を，帰納的定義により定義する．その意味論の導入方法を，モデルを介して接続する．意味論と構文論とは，形式体系と数学の理論体系との対立・相互関係のことを特にこう呼ぶ．２つの密接な結び付きは健全性と完全性に端的に集約される．健全性とは証明可能な論理式が常に正しい $\to$ ことを主張し，完全性が正しい論理式が証明可能である $\leftarrow$ ことを主張する．

すると，一階の論理式には，意味論的な振る舞いを全く同一にするという意味で標準形が見つかる．次の形をした論理式は Skolem 標準形と呼ばれ，その双対概念は Herbrand 標準形と呼ばれる．

$$\forall \vec{x} A(\vec{x}; \vec{a})$$

ここから数理論理学が始まる．

1.1 記法

1. $:\Leftrightarrow$ は，左辺を右辺で定義すると言うメタ言語内での記法である．
2. 同様にして，$\Rightarrow$ や $\Leftrightarrow$ はメタ論理での「ならば」記号として使い，$\to$ や $\leftrightarrow$ を一階述語論理の論理記号として採用する．
3. 配置集合は x^I とも ${}^I x$ とも書く．
4. $\langle x_0, \cdots, x_n \rangle$ とは，自然数の有限列 $(x_0, \cdots, x_n) \in {}^{<\omega}\mathbb{N}$ を，Cantor 写像 J^∞ により自然数に対応させたもの $J^\infty(x_0, \cdots, x_n) = \langle x_0, \cdots, x_n \rangle$ である．でも両方とも自然数の列として混用することも多い．
5. 変数や定数の組を $\vec{x} := (x_1, \cdots, x_n)$ と書き，$\forall \vec{x}$ を記号列 $\forall x_1 \forall x_2 \cdots \forall x_n$ の略記とする．

1.2　集合の扱い，帰納的定義，Cantor の対関数

1.2.1　集合と濃度

形式体系は記号列を扱う．その濃度の議論の基礎を見る．$\mathbb{N}$ は構成子 $(0, s)$ から帰納的に定まる．その上の有限列 $\mathbb{N}$-list も定まる．従って，可逆な写像 $\mathbb{N}^2 \to \mathbb{N}$ は一意に $\mathbb{N}$-list 上に一意に定まる．このような議論で，結論 $\mathbb{N} \simeq {}^{<\omega}\mathbb{N}$ を得る．この同型射は計算可能であり，「数え方が存在する」ことを意味する．その数え方の例（これしか存在しないのだろうか？）J^{∞} を以下のように構成できる．

記法 1.2.1. ${}^{<\omega}X := \cup_{n\in\mathbb{N}} X^n$ を，集合 X の元の有限列全体からなる集合とする．

定義 1.2.2 (Cantor's pairing function)**.** 次の関数 $J : \mathbb{N}^2 \to \mathbb{N}$ を Cantor の対関数という．

$$J(n, m) = \frac{(n+m)(n+m+1)}{2} + m = \left(\sum_{i \leqslant n+m} i \right) + m$$

注 1.2.3. 自然数の組 (n, m) を二数の和 $n + m$ によって分類する．$n + m = k$ の時，そのような組は $k + 1$ 個存在する．これを最初から番号を付けていき，$n + m = k$ のものは全て付け終わったとする．すでに番号は $\sum_{i \leqslant k+1} i$ まで付け終わったいる．従って，$n + m = k + 1$ を満たす組 $k + 2$ 個は，m の値に依って $\sum_{i \leqslant k+1} i + m$ と付ければ良い．この帰納的定義に定義した関数を明示的に書き下すと，随分と綺麗な形になる．

逆関数は，この $n + m = k$ の値を復元する関数 $t : \mathbb{N} \to \mathbb{N}$ を定めれば，簡単に構築できる．

命題 1.2.4.

$t(n) = \max \left\{ t \leqslant n \;\middle|\; \sum_{i \leqslant t} i \leqslant n \right\}$ とする．次の J_1, J_2 は J の逆関数である．

$$J_2(n) = n - \sum_{i \leqslant t(n)} i \qquad\qquad J_1(n) = t(n) - J_2(n)$$

注 1.2.5. J を繰り返すことで，$J^n : \mathbb{N}^n \to \mathbb{N}$ が次のように定まる．

$$\begin{array}{ccc} J^n : \mathbb{N}^n & \longrightarrow & \mathbb{N} \\ \downarrow & & \downarrow \\ (x_0, x_1, \cdots, x_{n-2}, x_{n-1}) & \longmapsto & J(x_0, J(x_1, J(\cdots, J(x_{n-2}, x_{n-1})\cdots))) \end{array}$$

この逆は次のように定まる．

$$J_i^n(x) = \begin{cases} J_1(J_2^{(i)}(x)) & i < n-1 \text{ の時} \\ J_2^{(n-1)}(x) & i = n-1 \text{ の時} \end{cases}$$

命題 1.2.6. X が可算集合ならば，その有限列全体の集合 ${}^{<\omega}X$ も可算である．

注 1.2.7. ${}^{<\omega}\mathbb{N}$ の元を，Cantor の対関数 J を用いて畳み込んでいく関数を帰納的に定義すれば，その閉包 ${}^{<\omega}\mathbb{N}$ 上に可逆な関数 $J^\infty : {}^{<\omega}\mathbb{N} \to \mathbb{N}$ が定まる．

［証明］. 次のように構成した写像 $J^\infty : {}^{<\omega}\mathbb{N} \to \mathbb{N}$ は可逆である．列の長さ n に依って挙動を変えるように，次のように定義すれば良い．

$$J^\infty(x_0, \cdots, x_{n-1}) \begin{cases} J(0,0) & n = 0 \text{ のとき} \\ J(0, 1 + x_0) & n = 1 \text{ のとき} \\ J(n-1, J^n(x_0, \cdots, x_{n-1})) & n > 1 \text{ のとき} \end{cases}$$

逆関数は，$x = J^\infty(x_0, \cdots, x_{n-1})$ としたとき，まず列の長さの情報を復元する．

$$\mathit{length}(x) = \begin{cases} 0 & x = J(0,0) = 0 \text{ のとき} \\ J_1(x) + 1 & \mathit{otherwise} \end{cases}$$

これにより，順次 $i = 0, 1, \cdots, n-1$ について，次のように復元できる．

$$x_i = \begin{cases} J_2(x) - 1 & \mathit{length}(x) = 1 \text{ のとき} \\ J_i^{\mathit{length}(x)}(J_2(x)) & \mathit{otherwise} \end{cases}$$

以上の議論は正確には，J^∞ の再帰的定義をアルゴリズム的に与えているのみである．計算機上に実装してみたい． □

例 1.2.8. $J^\infty(1,0) = J(1, J^2(1,0)) = J(1, J(1,0)) = J(1,1) = 2$ すごい！計算可能だ．この順番で折りたたんでいくからか！順番って大事だな，順番って関数の本質だな．「長さ $n = 1$ のものだけ先に数えてしまおう」とか言う方針では数え終わらないのだが，問題は数え終わらなさそのもので，数え方が存在するかどうかが可算性のポイントなのだな．そして，${}^{<\omega}\mathbb{N}$ にはこのような数え方 J^∞ が存在した．

1.2.2　二項関係

形式的な議論をするにあたって，集合の要素に言及する言葉としての，二項述語をいくつか整備しておくと便利である．
同値関係が一番代表的な二項関係と言える，これは非常に普遍的な言語となる．また，対称的な関係からの同値関係の一般的な構成法として，「反射的推移閉包」が存在する．反射推移閉包は射の特徴である．このうち対称的なものを同型（同値関係）というのである．
順序関係とは，「反対称性と推移性」を持つ二項関係のことである．このままでは共役な 2 つの関係（反射的閉包と非反射的核）が考えられるが，特に数学では反射的なものを順序とする．

定義 1.2.9 (二項関係の属性 : reflexive, symmetric)**.**

1. R が A 上反射的であるとは，次が成り立つことをいう : $\forall x \in A,\ R(x,x)$.
2. R が A 上非反射的であるとは，次が成り立つことをいう : $\forall x \in A,\ \neg R(x,x)$.
3. R が A 上対称的であるとは，次が成り立つことをいう : $\forall x,y \in A,\ R(x,y) \to R(y,x)$.
4. R が A 上反対称的であるとは，次が成り立つことをいう : $\forall x,y \in A,\ R(x,y) \wedge R(y,x) \to x = y$.
5. R が A 上非対称的であるとは，次が成り立つことをいう : $\forall x,y \in A,\ \neg(R(x,y) \wedge R(y,x))$.

同値関係

定義 1.2.10 (同値関係の構成 : reflexive and transitive closure)**.** 関係 R に対し，その**推移閉包** R^+ とは，次のように定義される関係である．

1. $R(x,y) \Rightarrow R^+(x,y)$.
2. $R^+(x,y) \wedge R^+(y,z) \Rightarrow R^+(x,z)$.
3. 上の 1,2 によって得られるもののみが R^+ の要素である．

関係 R の**反射的推移閉包**を，$R^* := R^+ \cup \Delta_X$ とする．

順序関係

特に順序関係について，さらにもう少し言葉を用意する．

順序関係とは，反射性と推移性と反対称性に特徴付けられる二項関係であるが，各順序関係は付随する非反射的な関係を持つ．実は，命題 1.2.13.4 が成り立つ．これより，推移性と反対称性の 2 条件を満たせば順序関係と読んでしまっても差し支えない．

記法 1.2.11. 関係 R に対して，その反射的閉包を $R_{=}(x,y) :\Leftrightarrow R(x,y) \vee x = y$ と定める．同様にして，非反射的核を $R_{\neq} (x,y) :\Leftrightarrow R(x,y) \wedge x \neq y$ とする．

例 1.2.12. 自然数 $\mathbb{N}$ 上の関係 $\subset$ は順序関係を定め，$\in = \subset_{\neq}$ がそれに付随する非反射的核である．

命題 1.2.13.

1. 関係 R が反射的ならば，$(R_{\neq})_{=} = R$ が成り立つ．
2. 関係 R が非反射的ならば，$(R_{=})_{\neq} = R$ が成り立つ．
3. R が反順序関係ならば，$R_{\neq}$ は非対称的かつ推移的な関係である．
4. R が非対称的かつ推移的な関係ならば，$R_{=}$ は半順序関係になる．

注 1.2.14. これが成り立つのが少し不思議かもしれないが，非反射性と言った時は，条件 $\forall x \in A,\ R(x,x)$ の否定ではなくて，双対である．

定義 1.2.15 (順序の属性 : maximum, upper bound, supremum, maximal)**.** 順序集合 $(A, \leqslant)$ について，$X \subset A, a \in A$ とする．

1. a が X の**最大元**であるとは，次が成り立つことをいう：$a \in X \wedge \forall x \in X,\ x \leqslant a$.
2. a が X の**上界**であるとは，次が成り立つことをいう：$\forall x \in X,\ x \leqslant a$.
3. a が X の**上限**であるとは，X の最小の上界であることをいう．
4. a が X の**極大元**であるとは，次が成り立つことをいう：$a \in X \wedge \forall x \in X,\ a \not< x$.

$\not< = \neg <$ が $\geqslant$ と等しくなるのは，これらが全順序を定める時のみである．また，双対概念についても，同様に定める．

注 1.2.16.

1. 実数の連続性で，常に上限が存在するという主張は，上界全体の集合 $\{x \in \mathbb{R} \mid \forall a \in X,\ a \leqslant x\}$ が下に閉じていることを利用しているに過ぎない．つまり，上限という

概念装置の定義が巧妙であるだけだ．反射的閉包をここで用いる（？）．うーん，数は順序関係の極みみたいなデータ構造だが，まだその局所的考察が足りない．

2. 極大元：「俺より強い奴は居ない」「全員と戦って全勝したわけじゃないが，とりあえず負けたことはない」

定義 1.2.17 (chain, antichain)**.** 半順序集合 (A, R) の部分集合 $X \subset A$ について，X が A 上線型順序（全順序）である時，これを **A 上の鎖**という．X の任意の二元が比較不能である時：$\forall x, y \in X,\ \neg(R(x,y) \vee R(y,x))$，これを **$A$ 上の反鎖**という．

最後に同値関係について考える．関係 R や関数 f と両立する／整合的である (compatible) とは，これらが定める商集合上の同値関係であることとして理解できる．

1.2.3　帰納的定義

帰納的定義はデータ構造の定義に使われる形式手法である．また帰納的定義自体を表すのに，BNF 記法というメタ言語がある．自然数などの数，多項式などの対象，行列など，扱いたい情報の平方としてのデータ構造は直観的に明らかなものが多いが，形式的に厳密に定義するのにこの形式的手法を用いる．すると，その上での関数も，この帰納的な定義から再帰的に定義される．これは数学基礎論独特の論理の進め方でもあり，プログラミングをすることも本質的にはこの行為と対応する．

定義 1.2.18 (inductive definition)**.** 集合 A 上での帰納的定義とは，その部分集合 $X_0 \subset A$ と有限個の関数 $f : A^n \to A$ の組に対して，その閉包 $X = \cup_{m\in\mathbb{N}} X_m$ $(X_m := f(X_{m-1})\ (m = 1, 2, \cdots))$ を A 上に定義する手法である．この時，X_0 の元と各関数 (f_i) を構成子という．

注 1.2.19. 閉包である点が重要である．後者関数 $s : x \mapsto x + 1$ に対して，組 $(0, s)$ による帰納的定義は自然数 $\mathbb{N}$ を定義するが，条件 $n \in X \Leftrightarrow n = 0 \vee \exists m \in X,\ n = m + 1$ 自体は $\mathbb{Z}$ も満たす．しかしこれらから生成される訳ではない．

注 1.2.20 (多項式)**.** 整数係数多項式全体の集合 $\mathbb{Z}[X]$ は次のように定義される．

1. 整数 $a \in \mathbb{Z}$ は $a \in \mathbb{Z}[X]$ で，変数 $x \in X$ も $x \in \mathbb{Z}[X]$.
2. $p, q \in \mathbb{Z}[X]$ ならば $p + q, p \cdot q \in \mathbb{Z}[X]$.
3. 以上によって $\mathbb{Z}[X]$ に含まれると分かるもののみが $\mathbb{Z}[X]$ の元である．

$X = \{x_1, \cdots, x_n\}$ の時，$p \in \mathbb{Z}[X]$ は $p(x_1, \cdots, x_n)$ などと書く．

　すると，このデータ構造 $\mathbb{Z}[X]$ に対する「代入」などの操作が，この帰納的定義から帰納的に定義できることが帰納的に証明できる．これが数学者には直観的に明らかなものを形式化する，形式的手法である．

1.2.4　Zorn の補題と無限集合の扱い

　無限集合の扱い方は奇妙になることが多く，その際には Zorn の補題がどうやら重要な位置を占める．その理由として，選択公理と ZF 上同値になることも一つの要因であろう．

定義 1.2.21 (inductive set)**.** 順序集合 $(A, <)$ について，任意の鎖 $C \subset A$ が上限 $\sup C$ を A 上に有する時，A は**帰納的**であるという．

補題 1.2.22 (Zorn's lemma)**.** 空でない帰納的（半）順序集合は極大元を有する．

注 1.2.23.

1. 束でも，論理体系についてのコンパクト性定理でも，いろんなデータ構造の上でこのような形をした主張を見てきた．Zorn の補題も，大域的な性質を，全称命題を使ってではあるが有限の範囲の中で確認可能にする定理の 1 つであり，しかもこれを順序関係という一番抽象的な述語について行っている．この抽象レベルだと，選択公理と同値になるのだ．
2. 鎖とは，帰納的定義をしたときに $X_m := f(X_{m-1})\ (m = 1, 2, \cdots)$ が包含関係について作る列である．これが極大元を持つとは，帰納的定義が成功することを意味する．そう考えるとなぜこれが成り立つのかの方が不思議になる．反例が作れないのか．ああ，ということは逆に言えば，集合が包含関係について作る順序集合は全て帰納的になるのか．

1.2.5　有限木の扱い

木も束も，形式体系では普遍的に現れるデータ構造で，一番おとなしい半順序集合のクラスである．特に組み合わせ論的な対象として現れる．その中で，有限木 $(T, \leqslant)$ は，ある自然数の有限列の集合が定める半順序集合 $(s(T), \subset)$ でコード出来（と同型で），逆に始切片関係と呼ばれる順序関係について閉じている（推移的である）ような「自然数の有限列の集合」は木とみなせる．これは全ての有限木自身が最終的には自然数 1 つでコード出来ると言う点で数学基礎論的にも重要な事実である．
従って，木自体も再帰的な方法で定義されるデータ構造であるが，木全体の集合 $\mathsf{FinTree} \subset \mathsf{P}({}^{<\omega}\mathbb{N})$ も再帰的な方法で定義されるデータ構造である．

定義 1.2.24 (finite tree)**.** 有限な半順序集合 $(T, \leqslant)$ であって，**根 (root)** と呼ばれる最小元 $r \in T$ を持ち，どんな $x \in T$ についてもそれ以下の元 $(\{y \in T \mid y < x\}, \leqslant)$ が全順序になっている半順序集合 $(T, \leqslant, r)$ を言う．木の元を**節 (node)** と言う．

$x < y$ であって，$\forall z \in T,\ \neg(x < z < y)$ である時，y は x の**子 (son, child)** であると言う．$\mathrm{son}_T(x) \subset T$ で x の子からなる T の元の集合を表す．$\mathrm{son}_T(x) = \varnothing$ を満たす $x \in T$ を**葉 (leaf)** と言う．

注 1.2.25. 木は $r : \mathit{tree}$ と $\mathrm{son} : \mathit{tree}^n \to \mathsf{P}(\mathit{tree})$ からなる構成子の組 (r, son) から生成されるデータ構造として定義できる．

集合 ${}^{<\omega}\mathbb{N}$ 上に自然に構造を入れて，有限木の受け皿とする．

定義 1.2.26 (concatenation)**.** ${}^{<\omega}\mathbb{N}$ 上の二項演算 $*$ を，$(x_0, \cdots, x_n) * (y_0, \cdots, y_m) = (x_0, \cdots, x_n, y_0, \cdots, y_m)$ として定める．多分再帰的方法によっても定められるが，数学の文脈ではあまり見ない．同様にして $\varnothing \in {}^{<\omega}\mathbb{N}$ は見るが，$[] \in {}^{<\omega}\mathbb{N}$ は見ない．

定義 1.2.27 (initial segment)**.** ${}^{<\omega}\mathbb{N}$ 上の順序 $\subset$ を $s \subset (x_0, \cdots, x_n) :\Leftrightarrow \exists m \leqslant n,\ s = (x_0, \cdots, x_m)$ と定める．関係 $s \subset t$ が成り立つ時，s は t の**始切片**であると言う．

命題 1.2.28 (有限木の自然数の有限列による表現)**.** $(T, \leqslant)$ を木とする．次のように帰納的に定めた写像 $s : T \to {}^{<\omega}\mathbb{N}$ は可逆である．

1. $s(r) = \varnothing$

2. $b \in T$ について，$s(b) = (x_0, \cdots, x_{n-1})$ であり，$son_T(b) = \{a_1, \cdots, a_m\}$ であったとする．この時，$s(a_i) := s(b) * (i)$ $(i = 1, 2, \cdots, m)$ と定める．

注 1.2.29. 従って，空列を含み，順序 $\subset$ について閉じている（推移的）であるような ${}^{<\omega}\mathbb{N}$ の有限部分集合は，有限木と同型である．

定義 1.2.30 (有限木の自然数によるコード)**.** 写像 $t : \mathrm{LimTree} \to \mathbb{N}$ 次のように帰納的に定義する．ただし $\mathrm{LimTree} \subset P({}^{<\omega}\mathbb{N})$ は有限木全体の集合とした．有限木 $T \subset {}^{<\omega}\mathbb{N}$ に対して，$t(T) \in \mathbb{N}$ を次のように定めた $t(T) = t(r)$ とする．

1. $son_T(b) = \{a_1, \cdots, a_m\}$ $(m \geqslant 0)$ として，$t(b) = J^\infty(b, J^\infty(t(a_1), \cdots, t(a_m)))$ と定める．子がない場合は $J^\infty(b)$ である．
2. 有限木なので，最終的に $t(r)$ が定まる．

注 1.2.31. これは木について拡張できないのか？

注 1.2.32. 木全体の集合 FinTree も，自然数の列の集合がなすデータ構造として次のように帰納的に定義できる．

1. $\mathbb{N}$ の元 1 つからなる一元集合は木である．
2. $T_1, \cdots, T_k$ は木とする．これらの根 $r(T_1), \cdots, r(T_k)$ を含む最短の列を $r(T_1), \cdots, r(T_k) \subset r$ とし，$son_T(r) = \{r(T_1), \cdots, r(T_k)\}$ として作った T は木である．

$r = []$ としたいから，あまり綺麗な定義ではないな．

1.3　一階述語論理

1.3.1　言語

一階述語論理での論理式とは，記号の列である．論理記号は，一階述語論理と言った時にすでに定まっており，これは論理としての模型である．一方非論理記号は，議論の対象に依って様々である．殆どの数学分野は一階の量化で十分表現できる．

定義 1.3.1 (language)**.** 次の記号を用意する．**論理記号**または**論理結合子**は次からなる．

1. （個体変数）$x_1, \cdots, x_n, \cdots$
2. 結合記号 $\wedge, \vee, \neg, \to, \leftrightarrow$
3. 量化記号 $\forall, \exists$
4. 括弧 $(), []$
5. 等号 $=$

論理記号以外に，**非論理記号**からなる集合を L と書き，**言語**と言う．数学の分野や用途によって様々であるが，例えば次のようなものがよく登場する．

1. 述語記号 $P/n, Q/n, R/n, \cdots$
2. 関数記号 $f/n, g/n, h/n, \cdots$
3. 定数記号 $a, b, c, \cdots$（arity 0 の関数記号または述語記号と見なせるが，大抵記号として区別される）．

注 1.3.2. L は空でも，どんなに大きい集合であっても構わない．また，変数記号は一般に可算無限個用意する．

例 1.3.3 (群論)**.** 群論の言語として $L_G = \{+/2, -/2, -/1, 0\}$ を定める．L_G-文 3 つからなる集合 T_G を次のように定め，これを群の公理と呼ぶ．

1. $\forall x \forall y \forall z,\ x + (y + z) = (z + y) + z$,
2. $\forall x,\ x + 0 = x$,
3. $\forall x,\ x - x = -x + x = 0$.

数学ではよく量化子は省略する．また混同の危険がない際，議論領域上の関係に $=^G$ などと添字もつけない場合が普通である．これを満たす L_G 構造 $G = (G, +, -, 0)$ を群という．$G = (\mathbb{Z}, +, -, 0)$ などが具体例である．即ち，G は T のモデルである：$G \vDash T$．対象領域の濃度を位数と呼ぶ．

アーベル群とは，公理系 $T'_G := T_G \cup \{\forall x \forall y,\ x + y = y + x\}$ を充すモデルのことをいう．非自明なアーベル群とは，$T''_G := T'_G \cup \{\exists x,\ x \neq 0\}$ を充すモデルのことをいう．

ねじれのないアーベル群とは，$T'''_G := T''_G \cup \{\forall n \geqslant 1 \forall x,\ x \neq 0 \to nx \neq 0\}$ のモデルのことである．ここで，新たに追加した論理式は，n は領域 G を走るわけではないので，正確には公理図式 $\forall x,\ x \neq 0 \to nx \neq 0$ $(n = 2, 3, \cdots)$ であるから，公理系 T'''_G は無限集合である．このことを，ねじれがないアーベル群は 1 階論理で公理化可能だが，有限公理化可能ではない，という．実は次が成り立つ．

命題 1.3.4 (Compactness)**.** 有限個の L_G-文が全てのねじれがないアーベル群で正しければ，それはあるねじれがあるアーベル群でも正しい．

注 1.3.5. これはコンパクト性定理の例になっている．こんなメタな言及が出来るのは，一階述語論理に内在した性質から言えるから，ということなのだろうか．

例 1.3.6 (体)**.**

命題 1.3.7 (Compactness)**.** ある 1 階の閉論理式が，全ての標数 0 の体で成り立つならば，それは十分大きい素数 p について，標数 p の体でも成立する．

例 1.3.8 (代数的閉体)**.**

例 1.3.9 (実数)**.**

命題 1.3.10 (Compactness)**.** $\mathbb{R} = (\mathbb{R}; +, \cdot, <, r)_{r\in\mathbb{R}}$ の拡大体 ${}^*\mathbb{R}$ で，アルキメデス的ではなく，しかも $\mathbb{R}$ と同じ一階論理の閉論理式を充すものが存在する．

注 1.3.11. ${}^*\mathbb{R}$ はアルキメデス的ではないので，無限大 $c > n$ $(n = 0, 1, 2, \cdots)$ に当たる定数が存在する．この逆元 c^{-1} は無限小に当たる．これが，無限小解析，または A. Robinson による超準解析の出発点となっている．

さていよいよ本格的に逆数学的な存在のお出ましである．

例 1.3.12 (実閉体)**.**

1.3.2　構文論

記法 1.3.13.

1. 言語 L を 1 つ定め，変項を $x, y, z, \cdots$，言語 L に含まれる定数を $a, b, c, \cdots$，述語を $P, Q, R, \cdots$ と書くこととする．
2. Prolog での記法に倣い，述語の arity $n \in \mathbb{N}$ を P/n というように述語記号に/を挟んで添えて書く．
3. 記号の有限列として一致するという同値関係を $\equiv$ で表す．
4. 記号の自由出現に対する置換を $s[a := t]$ または $s(t)$ に対して $s(a)$ などと書く．後者は特に変数代入の文脈などで用いる．

定義 1.3.14 (expression)**.** $L \cup \{\wedge, \vee, \neg, \to, =, \exists, \forall, x, y, z, \cdots\}$ の有限列を，**L-記号列**という．

注 1.3.15. 記号列には長さという特徴量がある．今後何かを証明するにあたって，この長さについての帰納法で示すことが多い．これは expression が構成子 $L' := L \cup \{\wedge, \vee, \neg, \to, =, \exists, \forall, x, y, z, \cdots\}$ と $cons : L \times L^n \to L^{n+1}$ $(n = 0, 1, 2, \cdots)$ から帰納的に定義されるからである．その上の性質が定まっているかの証明は全て此処に戻ることになる．

この上に，様々な表現を帰納的に定義していく．L が可算である時，記号列全体の集合も可算であることに注意する（命題 1.2.6）．まず，もの (object) の受け皿としての最小単位が項である．x, x^n や値 $f(x)$，多項式 $a_n x^n + a_1 x + a_0$ なども項である．

定義 1.3.16 (term)**.** **L-項**または **L-式**全体の集合 Tm_L を次の構成子により帰納的に定める．

1. $x, y, z, \cdots, a, b, c, \cdots \in Tm_L$.
2. $f : Tm_L^n \to Tm_L \quad (f^n \in L)$.

変数 $x, y, z, \cdots$ を含まない L-項を **L-閉項 (L-closed term)** ／または **L-基礎項 (L-ground term)** と言う．

定義 1.3.17 (literal)**.** 次のいずれかの形をした論理式を，**原子論理式 (atomic formulta)** と言う．

$$t_1 = t_2, \qquad R(t_1, \cdots, t_n), \qquad (t_i \in Tm_L, R/n \in L).$$

原子論理式とその否定形を **literal** と言う．

定義 1.3.18 (formula)**.** **L-論理式**とは，次のように帰納的に定義される記号列のことである．

1. literal は論理式である．
2. 論理式 φ, ψ について，$(\varphi \vee \psi), (\varphi \wedge \psi), (\varphi \to \psi)$ は論理式である．
3. 論理式 φ と変数 x について，$(\exists x \varphi), (\forall x \varphi)$ は論理式である．

L-論理式全体の集合を Fml_L と書くこととする．

注 1.3.19. 論理式 $\varphi[\nu_1, \cdots, \nu_n]$ と書いた時，自由出現する変数が高々 $\nu_1, \cdots, \nu_n$ として有限個準備されているのみで，全て出現するとは限らない．

論理式の標準形

主に否定標準形と冠頭標準形と Skolem 標準形（とその双対概念）の 3 種（4 種）の標準形がある．これらは全て，古典論理の範囲では「標準形」である．

標準形の議論はどこか代数学的である．そもそも論理結合子が $\circ$ の代数に見える．

定義 1.3.20 (negation normal form)**.**

1. literal から，4 種の論理結合子 $\wedge, \vee, \exists, \forall$ のみによって生成される論理式を**否定標準形**の論理式という．
2. 量化記号なしの論理式 θ について，$\exists x_1 \cdots \exists x_n \theta (n \geqslant 0)$ の形の論理式を $\exists$-論理式，$\forall x_1 \cdots \forall x_n \theta (n \geqslant 0)$ の形の論理式を $\forall$-論理式という．

補題 1.3.21 (negation normal form)**.** 任意の論理式 φ に対して，それと古典論理の範囲で同値な否定標準形の論理式が作れる．

［**証明**］**.** 長さ n の論理式 φ について考える．1. 論理式が $\varphi \to \psi$ の形をしていた場合，$\vDash_{NK} (\varphi \to \psi) \leftrightarrow (\neg\varphi \vee \psi)$ により $\to$ を消去する．2. De Morgan の法則と二重否定の除去則により，記号 $\neg$ はどんどん中に入れると，literal と $\wedge, \vee, \exists, \forall$ のみからなる論理式を得る．

$$\begin{aligned}
\vDash_{NK} \neg(\varphi \wedge \psi) &\leftrightarrow (\neg\varphi \vee \neg\psi) \quad (\to \text{が NK}) \\
\vDash_{NJ} \neg(\varphi \vee \psi) &\leftrightarrow (\neg\varphi \wedge \neg\psi) \\
\vDash_{NJ} \neg\exists x\varphi &\leftrightarrow \forall x\neg\varphi \\
\vDash_{NK} \neg\forall x\varphi &\leftrightarrow \exists x\neg\varphi \quad (\to \text{が NK}) \\
\vDash_{NK} (\neg\neg\varphi) &\leftrightarrow \varphi
\end{aligned}$$

このアルゴリズムは φ の長さ n に依らず有効である． □

定義 1.3.22 (prenex normal form)**.** 次の形の論理式を**冠頭標準形**の論理式という．

$$Q_1 x_1 \cdots Q_n x_n \theta \quad (n \geqslant 0, Q_i \in \{\exists, \forall\}, \theta\text{には量化記号なし})$$

この時 θ をこの論理式の母式 (matrix) という．

補題 1.3.23 (prenex normal form)**.** 任意の論理式 φ に対して，それと古典論理の範囲で同値な冠頭標準形の論理式が作れる．

［証明］. 先ず，長さ n の論理式 φ について，否定標準形 φ' を補題 1.3.21 の算譜から得る．この論理式 φ' の長さについての帰納法で，次の変数退避と量化子のくくりだしのアルゴリズムで冠頭標準形を得る．

$$\begin{aligned} &\vDash Qx\varphi \leftrightarrow Qy(\varphi[x := y]) && Q \in \{\exists, \forall\} \\ &\vDash (\varphi \circ Qy\psi) \leftrightarrow Qy(\varphi \circ \psi) && \circ \in \{\wedge, \vee\} \end{aligned}$$

□

定義 1.3.24 (Skolem normal form)**.** 冠頭標準形の論理式 φ について，φ 中の各存在量化記号 $\exists y_l$ $(l \in \mathbb{N})$ について，それより前に全称量化記号が $n_l (\geqslant 0)$ 個 $\forall x_1^l, \cdots, \forall x_{n_l}^l$ とあるとすれば，n_l-変数の新たな関数記号 f_l を導入して，φ の母式中の変数 y_l を $f_l(x_1^l, \cdots, x_{n_l}^l)$ で置き換えることで，存在量化記号 $\exists y_l$ を消去することができる．これは拡張された言語 $L \cup \{f_i\}$ での $\forall$-論理式であり，これを **Skolem（連言）標準形**という．

次が成り立つという意味で，これは標準形である．

命題 1.3.25. 冠頭標準形の論理式 φ のモデルに対して，適切に Skolem 関数記号 f_i に解釈を追加することでその Skolem 標準形 φ^S のモデルが得られ，φ^S のモデルから Skolem 関数への解釈を削れば φ のモデルが得られる．

確かに，φ での各存在量化子で縛られた変数 y_l に対して，y_l が存在することと f_l が構成可能であることに等価である．

結局 Skolem 標準形が Skolem 関数 f/n の存在保証について少し反則的であることを除いて，全部直感的なアルゴリズムで標準形を得る．でもその論理的な同値性は二重否定則を認めない限り成り立たないという事実はどう考えれば良いのかわからない．

1.3.3　意味論

形式体系にとっては明後日の方向から定義した対象「構造」を介して，論理式に意味を付与し，没交渉だった状態から，我々の理解と対話可能にする．果てには証明可能性などの概念もこの寄り道から定義する．

定義 1.3.26 (structure)**. L-構造**とは，議論領域 $M \neq \varnothing$ と解釈写像 $F : L \to M$ との対

$\mathcal{M} := (M, F)$ であって，言語 L に対して次を満たすもののことである．

1. $R/n \in L$ に対して，$F(R) =: R^{\mathcal{M}}$ は M 上の n-項関係である：$R^{\mathcal{M}} \subset M^n$．
2. $f/n \in L$ に対して，$F(f) =: f^{\mathcal{M}}$ は M 上の n-項関数である：$f^{\mathcal{M}} : M^n \to M$．
3. 特に定数 $c \in L$ に対しては，$F(c) =: c^{\mathcal{M}} \in M$ である．

構造 $\mathcal{M}$ は通常 $\mathcal{M} = (M; R^{\mathcal{M}}, \cdots, f^{\mathcal{M}}, \cdots, c^{\mathcal{M}}, \cdots)$ と表し，$|\mathcal{M}| := M$ を構造 $\mathcal{M}$ の対象領域もしくは領域 (universe) という．

注 1.3.27. 数学の慣習では，右肩の添字を省略し，構造 $\mathcal{M}$ と領域 M を混用する．例えば $G = (G, +, -, 0)$ など．

構造中心に論理式をみる場合，これに対応して言語を拡張する操作を定義しておきたい．言語 L と L-構造 $\mathcal{M} = (M, F)$ が与えられた時，これに適して言語 L を拡張した言語を $L(\mathcal{M}) := L \cup \{c_a \mid a \in M\}$ と定める．この c_a を $a \in M$ の名前 (name) という．解釈写像 $F : L \to M$ は自然に $F : L(\mathcal{M}) \to M$ に拡張される．

定義 1.3.28 (value of terms)**.** L を言語，$\mathcal{M} = (M, F)$ を L-構造，t を $L(\mathcal{M})$-閉式とする．解釈写像 F が定める式 t の**値** $t^{\mathcal{M}}$ を次のように帰納的に定める．

1. $t/0$ ならば，$t^{\mathcal{M}} := c^{\mathcal{M}}$．
2. $\exists f \in L,\ t/n \equiv f(t_1, \cdots, t_n)$ ならば，$t^{\mathcal{M}} := f^{\mathcal{M}}(t_1^{\mathcal{M}}, \cdots, t_n^{\mathcal{M}})$．

定義 1.3.29 (satisfication relation)**.** L を言語，$\mathcal{M} = (M, F)$ を L-構造，φ を $L(\mathcal{M})$-文とする．L-構造と $L(\mathcal{M})$-閉論理式の**充足関係** $\vDash$ を，次のように帰納的に定義する．なお，7,8 の定義で左辺ではメタ論理を一階の言語で表現した．

1. $\mathcal{M} \vDash (t_1 = t_2) :\Leftrightarrow t_1^{\mathcal{M}} = t_2^{\mathcal{M}}$．
2. $\mathcal{M} \vDash R(t_1, \cdots, t_n)\ (R \in L) :\Leftrightarrow (t_1^{\mathcal{M}}, \cdots, t_n^{\mathcal{M}}) \in R^{\mathcal{M}}$．
3. $\underline{\mathcal{M} \vDash \neg\varphi :\Leftrightarrow \mathcal{M} \nvDash \varphi} :\Leftrightarrow \mathcal{M} \vDash \varphi$ でない．
4. $\mathcal{M} \vDash \varphi \vee \psi :\Leftrightarrow \mathcal{M} \vDash \varphi \vee \mathcal{M} \vDash \psi$．
5. $\mathcal{M} \vDash \varphi \wedge \psi :\Leftrightarrow \mathcal{M} \vDash \varphi \wedge \mathcal{M} \vDash \psi$．
6. $\mathcal{M} \vDash \varphi \to \psi :\Leftrightarrow \mathcal{M} \vDash \varphi \to \mathcal{M} \vDash \psi \Leftrightarrow \neg(\mathcal{M} \vDash \varphi) \vee \mathcal{M} \vDash \psi$．
7. $\mathcal{M} \vDash \exists v(v),\ \varphi :\Leftrightarrow \exists a \in M,\ \mathcal{M} \vDash \varphi(c_a)$．
8. $\mathcal{M} \vDash \forall v(v),\ \varphi :\Leftrightarrow \forall a \in M,\ \mathcal{M} \vDash \varphi(c_a)$．

注 1.3.30. 初読の際は仰々しく見えるだけで内容が入ってこなかったが，今では直観的

観念を形式的議論の俎上に載せる帰納的定義の威力を痛感する．メタ論理 $:\Leftrightarrow$ の右辺は日本語の代わりにすでに使い慣れた一階論理で表現したが，こうみると関係 $\vDash$ は非常に自然に定義される．

定義 1.3.31 (theory)**.** **L-公理系**とは，L-閉論理式の集合をいう．

モデルは公理系について定義される．

定義 1.3.32 (model)**.** T を L-公理系とする．

1. $\mathcal{M}$ が T のモデルである：$\mathcal{M} \vDash \mathsf{T}$ とは，$\forall \varphi \in \mathsf{T}$, $\mathcal{M} \vDash \varphi$ が成り立つことをいう．

次に，遂に論理式の間の関係を定める．

定義 1.3.33 (logical consequence, tautology)**.** T を L-公理系とする．L-閉論理式 φ が T の**論理的帰結**である，または**定理**である：$\mathsf{T} \vDash \varphi$ とは，T の任意のモデル $\mathcal{M}$ が φ を充す：$\mathcal{M} \vDash \varphi$ ことをいう．

特に $\mathsf{T} = \varnothing$ の時，$\vDash \varphi$ と書き，全てのモデルで成り立つということであるから，**論理的に正しいまたは恒真 (tautology) であるという．**

$\vDash (\varphi \leftrightarrow \psi)$ が成り立つ時，φ, ψ は（**論理的に）同値である**という．

注 1.3.34. 閉じていない論理式 φ については，全ての自由変数 (parameter) を全称量化した全称閉包を $\varphi^{\forall}$ として，$\mathsf{T} \vDash \varphi :\Leftrightarrow \mathsf{T} \vDash \varphi^{\forall}$ と定める．

次に，より弱い関係を定める．

定義 1.3.35 (satisfiability)**.** T を L-公理系，φ を L-閉論理式とする．T のあるモデル $\mathcal{M}$ で $\mathcal{M} \vDash \varphi$ が成り立つ時，φ **は T で充足可能**であるという．

最後に公理系同士の関係を定める．

定義 1.3.36 (extension)**.** L-公理系 T と L'-公理系 T' について，次の 2 条件を満たす時，公理系 T' は T の**拡大**であるという．

1. $\mathsf{L} \subset \mathsf{L}'$.
2. 全ての L-論理式 φ について，$\mathsf{T} \vDash \varphi \Rightarrow \mathsf{T}' \vDash \varphi$.

条件 2 の逆 $\mathsf{T} \vDash \varphi \Leftarrow \mathsf{T}' \vDash \varphi$ も全ての L-論理式 φ について成り立つ場合，これを特に**保存拡大**という．

1.4　コンパクト性定理

一階論理のコンパクト性の問題を命題論理のものに帰着させる方法は筆者の知っている範囲で 2 つある．L. Henkin 21-06 による方法と Herbrand の定理による方法である．

1.4.1　命題論理：論理結合子代数

命題論理は論理モデルであるというより，4 つの論理結合子 $\wedge, \vee, \to, \neg$ の代数的性質に注目するたくさんの内部構造を抽象化したモデルに思える．実際，命題論理式 $A(x_1, \cdots, x_n)$ は，真理関数 $f_A : 2^n \to 2$ に一対一対応する．
意味論については，論理式を「充す」ものとして，一階論理のモデル $\mathcal{M}$ は命題論理の付値 μ に対応する．

先ず，命題論理式は命題変数の集合 I を指定するごとに生成されるデータ構造である．

定義 1.4.1 (prime formula / propositional variable)**.** 空でない集合 $I = \{p, q, r, \cdots\}$ の元を**素論理式**または**命題変数**と呼び，その上に関数 $\neg : I \to I, \wedge, \vee, \to : I^2 \to I$ により帰納的に定義される記号列を I **上の命題論理式**という．その集合を Fm_I と書くこととする．

その意味論は**付値 (truth assignment)** $\nu : I \to 2$ を 1 つ取るごとに定まる．これを真理値表と呼ぶ．帰納的定義に沿って，付値は $\overline{\nu} : Fm_I \to 2$ に一意的に拡張される．

定義 1.4.2 (tautology)**.** 命題論理式 A が**恒真**であるとは，$\forall \nu \in \hom(I, 2)$, $\nu(A) = 1$ であることをいう．

これが一階論理にて $\models A$ が成り立つことにあたり，同様にして「論理的に同値」「充足可能」の概念が定義できる．論理式を「充す」ものとして，一階論理のモデル $\mathcal{M}$ は命題論理の付値 μ に対応する．

注 1.4.3 (truth table)**.** 命題論理式 A が与えられたとき，その恒真性または充足可能性の判定は有限回の機械的な操作で可能である．A は有限長なので，有限の数の命題変数しか含まず，それについての局所的な付値の取り方（論理式 A の真理値表という）について調べれば良いので集合 I の濃度は関係がない．

1.4.2　命題論理のコンパクト性定理

定理 1.4.4 (命題論理のコンパクト性定理)**.** T を命題論理の論理式の集合とする．次の 2 条件は同値である．

1. T は充足可能である．
2. T の任意の（有限）部分集合は充足可能である．

条件 2 が成り立つことを**有限充足可能**と呼ぶこととする．

［**証明**］**.** 1.⇒2. は明らかである．2.⇒1. について考える．命題 1.4.8 より，有限充足可能な I-命題論理式の集合 T は，（有限充足可能なまま）極大な S に拡張できる．命題 1.4.6 より，極大な集合 S には $S = S_\mu$ を充す付値 μ が存在する．この μ について，$T \subset S$ も充足されるから，T は充足可能である． □

定義 1.4.5 (maximal)**.** I-命題論理式の集合 S が**極大**であるとは，次の 2 条件を満たすこととする．

1. S は有限充足可能である：$\mu_I \vDash S$.
2. 任意の I-論理式 $A \in Fm_I$ について，$(A \in S) \vee (\neg A \in S)$ である．

命題 1.4.6 (極大性の特徴付け)**.** 付値 $\mu : I \to 2$ によって充たされるような I-論理式の集合を $S_\mu := \{A \in Fm_I \mid \mu(A) = 1\}$ と置く．I-命題論理式の集合 S について，次の 2 条件は同値である．

1. S は極大である．
2. $S = S_\mu$ を満たす付値 $\mu : I \to 2$ が存在する．

［**証明**］**.** 2.⇒1. は，$A, \neg A$ のいずれかは μ により 1 に対応づけられるので，S_μ は極大である．S_μ は μ により充足される論理式の集合であるから，勿論有限充足可能である．

1.⇒2. を考える．$S \subset Fm_I$ の特性関数を $\mu := \chi_S$ とする．すると，S の極大性から $\mu : I \to 2$ が定まっている．つまり，4 つの論理結合子について次が成り立っていることを，A の長さについての帰納法により確かめれば良い．

$$B \in S \Leftrightarrow \neg B \notin S$$
$$A \vee B \in S \Leftrightarrow A \in S \text{ または } B \in S$$
$$A \wedge B \in S \Leftrightarrow A \in S \text{ かつ } B \in S$$

$$A \to B \in S \Leftrightarrow A \notin S \text{ または } B \in S$$

例えば 1 つ目の条件は，$\Rightarrow$ は $\{B, \neg B\} \subset S$ ならば S の（有限）充足性に反し，$\Leftarrow$ は $B \notin S \wedge \neg B \notin S$ ならば S の極大性に反することから従う．□

補題 1.4.7. S を有限充足可能な命題論理式の集合とする．命題論理式 A について $S \cup \{A\}$ が有限充足不可能ならば，$S \cup \{\neg A\}$ は有限充足可能である（元々 $\neg A \in S$ だった場合も含め）．

［証明］．有限な部分集合 $S_0 \cup \{\neg A\} \subset S \cup \{\neg A\}$ を任意に取る．いま，$S \cup \{A\}$ は有限充足不可能だから，有限部分集合 $S_1 \cup \{A\} \subset S \cup \{A\}$ が存在して，充足不可能である．これについて，$S_0 \cup S_1 \subset S$ を充す付値 μ を取ると，$S_1 \cup \{A\}$ は充足不可能であるから $\mu(A) = 0$ である．従って，$\mu(\neg A) = 1$．よって，この μ によって $S_0 \cup \{\neg A\}$ は充たされる．よって，$S \cup \{\neg A\}$ は有限充足可能．□

命題 1.4.8 (命題論理のコンパクト性定理 2)**.** 有限充足可能な I-命題論理式の集合 T は，（有限充足可能なまま）極大な S に拡張できる．（I が可算な場合は一意的な拡張になり，I が非可算である場合はそうとは限らない）．

［証明］．先ず，素論理式の集合 I を可算として証明する．I は可算なので I-命題論理式全体の集合 Fm_I も可算だから，附番 $\mathbb{N} \to Fm_I$ が取れる．これを 1 つ定め $A_0, A_1, A_2, \cdots$ とする．有限充足可能な I-論理式の集合の包含関係による列 $T_0 \subset T_1 \subset T_2 \subset \cdots$ を次のように帰納的に定義する．

$$T_0 := T$$

$$T_{n+1} := \begin{cases} T_n \cup \{A_n\} & \text{これが充足可能な時,} \\ T_n \cup \{\neg A_n\} & otherwise. \end{cases}$$

よって，$S := \cup_{n \in \mathbb{N}} T_n$ も有限充足可能である．またこの S は極大である．任意の論理式 $A \in Fm_I$ について $n \in \mathbb{N}$ が存在して $A \equiv A_n$ であるから，S の定義より $A \in S \vee \neg A \in S$ である．

I を一般の集合とする．$T \subset S$ を充す有限充足可能な I-論理式の集合 S 全体の集合を $Fm_T \subset Fm_I$ とする．Fm_T に包含関係により順序を導入すると，これは全ての全順序部分集合 $S_0 \subset S_1 \subset S_2 \subset \cdots$ は極大 $S_\infty := \cup_{n \in \mathbb{N}} S_n$ を Fm_T 内に持つので，この順序は帰納的になる．従って，Fm_T の中に（包含関係の意味で）極大元 S が取れる．これは I-論理式の集合として極大であることを示す．S は有限充足可能だから $A \in S \Rightarrow \neg A \notin S$ である．では $A \notin S$ として $\neg A \in S$ を導く．$S \subsetneq S \cup \{A\}$ が有限充足可能ならば S の包含

関係についての極大性に矛盾するので，$S \cup \{A\}$ は有限充足不可能である．よって補題より，$S \cup \{\neg A\}$ は有限充足可能である．S の極大性より，これは $\neg A \in S$ を意味する．　□

命題論理のコンパクト性定理は，命題論理式の集合の属性として，有限充足可能性と充足可能性が同値であることを意味する．これを，有限充足可能な集合 T の全体と充足可能な集合 T の全体とがなす Fm_I 上の部分集合の形が一致する（どちらも極大な集合 $S = S_\mu$ を極大とする包含関係 $\subset$ による帰納的順序集合をなす）ことを示すことによって証明した．そこに至るまでも刺激的な証明たちであった．
先ず，S が極大であることと，S はある付値 μ が真にする命題論理式からなる集合であること（そのような μ が存在すること）とは同値である．これは命題論理式 Fm_I の構成子についての帰納法で示せる．次に，S は一歩ずつ拡張できる．この step の証明も技巧的で，不思議な論理のつながりを感じた．これより，$T_0 \subset T_1 \subset T_2 \subset \cdots$ の形がわかる．I が非可算の場合はこの系列は分岐を含んだ気構造になるかもしれないが，いずれにしろ帰納的であると言う概念で捕まえられるため，Zorn の補題により一緒くたに議論できる．S の一歩ずつの拡張の algorithm がこれ以上進捗を生まなくなる点が極大である．

1.4.3　Henkin 定数

定義 1.4.9 (一階論理の命題論理への翻訳)**.** 一階の L-論理式のうち，原子論理式と量化記号 $\exists, \forall$ から始まる L-論理式とを集合 I に入れる．

定義 1.4.10 (tautology of first-order logic)**.** 素論理式の集合 I にどんな付値を与えても真となる論理式を**一階論理のトートロジー**という．

例 1.4.11 (一階論理では恒真であることと tautology であることにはズレがある)**.** 論理式 $\forall x R(x) \vee \neg \forall x R(x)$ は $p \vee \neg p (p \in I)$ 型の tautology である．

一方で，$c = c$ や $\forall x(R(x) \vee \neg R(x))$ はこれ自体が素論理式であるから tautology とは言わない，これ自身に 0 を対応させる付値 $\mu : I \to 2$ が取れる．同様にして $\neg \exists x S(x) \to \forall x \neg S(x)$ は論理的には正しいが，構文的には $\neg p \to q (p, q \in I)$ の形をしているため，$p \mapsto 0, q \mapsto 0$ の付値について偽と解釈出来るから（実際は De Morgan 則より $\neg p = q$ なので，本来の一階論理の意味論としては nonsense な解釈である），tautology とは呼ばない．

この様に，tautology というときは付値の存在についての命題論理的な意味論を無理に一階論理に適用した時の文脈で使う概念であり，本来の意味論とは異なる．

一階論理としては恒真であっても，定義 1.4.9 のやり方では命題論理の意味で恒真には必ずしもならないことは見た．かと言って定義 1.4.9 よりも筋の良いやり方は思いつかない．定義 1.4.9 により引き起こされた事象の一部を命題として抜き出すと，次の様になる．

補題 1.4.12. L を言語，$\mathcal{M}$ を L-構造とする．名前付き言語 $L(\mathcal{M})$ の閉素論理式 $I'_{\mathcal{M}}$ に対する付値 $\nu : I'_{\mathcal{M}} \to 2$ で，どんな $L(\mathcal{M})$-閉論理式 φ に対しても $\mathcal{M} \vDash \varphi \Leftrightarrow \overline{\nu}(\varphi) = 1$ となるものが一意的に存在する．

従って，モデルを持つ L-閉論理式の集合 T は，命題論理の意味でも充足可能である．

［証明］．この付値 ν は，$\mathcal{M} \vDash \varphi$ を満たす構造 $\mathcal{M}$ が定める付値 $\nu_{\mathcal{M}}$ に等しく，これ以外の構成はない．

$$\mathcal{M} \vDash \varphi \Rightarrow \nu_{\mathcal{M}}(\varphi) = 1$$
$$\mathcal{M} \nvDash \varphi \Rightarrow \nu_{\mathcal{M}}(\varphi) = 0$$

□

注 1.4.13 (逆の反例)**.** 定義 1.4.9 による翻訳で，命題論理として充足可能だったとしても，一階論理としてモデルを持つとは限らない．例えば

$$\{\forall x(R(x) \to S(x)), \forall x R(x), \neg\forall x S(x)\}$$

は命題論理への翻訳では $\{p, q, \neg r\}$ $(p, q, r \in I)$ だから明らかに充足可能だが，一階論理としては矛盾している．

ここで，言語 L の Henkin 拡張 (witnessing expansion)$L(C)$ を，定数の集合 C に対して $L(C) = L \cup C$ として作る．

定義 1.4.14 (witnessing expansion)**.** 言語 L に対して，次の様に相互再帰的に定義される言語 L の拡張を $L(C)$ とする．

1. $C_0 = \varnothing$ とする．
2. $L_n := L \cup C_n$ とする．
3. $C_{n_1} := C_n \cup \{\text{literal } \varphi(x) \text{ に対して } \exists x\varphi(x) \text{ という形の } L_n\text{閉論理式に対応した定数 } c_{\exists x\varphi(x)}\}$ とする．
4. 最後に $C = \cup_{n\in\mathbb{N}} C_n$, $L(C) = L \cup C$ とする．

3. では，すでに以前の L_m $(m < n)$-論理式として導入されている $c_{\exists x\varphi(x)}$ はダブルカウントされないことに注意．また，閉論理式ごとに定数を作っているので，$c_{\exists x\varphi(x)}, c_{\exists y\varphi(y)}(y \not\equiv x), c_{\exists x\varphi(x)\wedge\varphi(x)}$ はみな違う．

定義 1.4.15 (Henkin axiom)**.** 各定数 $c_{\exists x\varphi(x)} \in C$ は **Henkin 定数 (witnessing constant)** と呼ばれ，次の **Henkin 公理**を満たすものとする．

$$\exists x\varphi(x) \to \varphi(c_{\exists x\varphi(x)})$$
$$\varphi(c_{\exists x\neg\varphi(x)}) \to \forall x\varphi(x)$$

即ち，ある構造で $\exists x\varphi(x)$ が正しければ，その witness a が存在し，これが φ を真にする．これに対応して言語を拡張し，名前 $c_{\exists x\varphi(x)}$ を附ける．一方で $\neg\forall x\varphi(x) \Leftrightarrow \exists x\neg\varphi(x)$ が成り立てば，その反例に名前 $c_{\exists x\neg\varphi(x)}$ という名前をつける．

定義 1.4.16 (quantifier axiom)**.** 次の，$L(C)$-閉項 t についての公理図式を**量化公理**と呼ぶ．

$$\varphi(t) \to \exists x\varphi(x)$$
$$\forall x\varphi(x) \to \varphi(t)$$

Henkin 公理と量化公理とからなる $L(C)$-閉論理式の集合を T_{Henkin} とする．

公理系については拡大 (extension) という概念を定義してあるが，構造についても関係を定義する．

定義 1.4.17 (expansion, reduct)**.** 言語 L, L' について，記号の集合として $L \subset L'$ とする．$\mathcal{M}' = (M; F')$ を L'-構造とすると，L-構造 $\mathcal{M} := (M; F' \restriction L)$ を $\mathcal{M}'$ の L への**縮小**といい，逆に見たものを**拡張**という．

さて，ここまで整備した．するとまずは次が成り立つ．証明の仕方は勿論，帰納的に Henkin 拡張の仕組みを作った時から決まっている．

補題 1.4.18. $\mathcal{M} = (M; \cdots)$ を L-構造，$L(C)$ を L の Henkin 拡張とする．C に属する定数に適切な M への解釈を拡張すると，$\mathcal{M}$ の拡張として T_{Henkin} のモデル $\mathcal{M}'$ を得る．

［**証明**］**.** 量化公理は常に正しいから，Henkin 公理を考える．それも特に 1 つ目を考えれば，2 つ目は 1 つ目により $\exists x\neg\varphi(x) \to \neg\varphi(c_{\exists x\neg\varphi(x)})$ は $\neg\forall x\varphi(x) \to \neg\varphi(c_{\exists x\neg\varphi(x)})$ に同値で，この対偶が 2 つ目である．

閉論理式

$$\exists x\varphi(x) \to \varphi(c_{\exists x\varphi(x)})$$

を正しくする様な解釈 $F' : C \to M$ を，Henkin 定数の帰納的定義に沿って構成する．

$\mathcal{M}_0 := \mathcal{M}$ とする．$L_n = L \cup C$-構造 $\mathcal{M}_n = (M, F_n)$ から $\mathcal{M}_{n+1} = (M, F_{n+1})$ を次の様に定義する．$c_{\exists x\varphi(x)} \in C_{n+1} \setminus C_n$ に対して，$\mathcal{M}_n \vDash \exists x\varphi(x)$ ならばこの式を満たす $a \in M$ を任意に一つとり，$F_{n+1}(c_{\exists x\varphi(x)}) = a$ と定める．そうでないなら，任意の M の元を $F(c_{\exists x\varphi(x)})$ とする．この定義により得た $\mathcal{M}' = \cup_{n\in\mathbb{N}}\mathcal{M}_n$ は $L(C)$-構造で，Henkin 公理を満たす． □

定義 1.4.19 (canonical structure)**.** $L(C)$-構造 $\mathcal{M} = (M; F)$ が，$M = \{c^{\mathcal{M}} \mid c \in C\}$ となっている時，即ち $F \restriction C : C \to M$ が全射である時，これを $L(C)$-**標準構造**という．

注 1.4.20. 領域 M の元たちが論理的に区別可能であるなら，普通に F を作ればこうなるのか．

定義 1.4.21 (equality axiom)**.** **等号公理**は次のいずれかの論理式を指す．ただし，$t, s, t_1, \cdots$ は $L(C)$-閉項で，$R/n, f/n \in L$ である．

$$\begin{gathered}
t = t \\
t = s \to s = t \\
t_1 = t_2 \wedge t_2 = t_3 \to t_1 = t_2 \\
t_1 = s_1 \wedge \cdots \wedge t_n = s_n \to R(t_1, \cdots, t_n) \to R(s_1, \cdots, s_n) \\
t_1 = s_1 \wedge \cdots \wedge t_n = s_n \to f(t_1, \cdots, t_n) = f(s_1, \cdots, s_n)
\end{gathered}$$

この 5 つの閉論理式の集合を $\mathbf{Eq}$ と書く．これは，初めの 3 つは$=$が同値関係であることを，残りの 2 つは合同関係であることを保証することになる．

補題 1.4.22 (Henkin 拡張による，一階論理の充足可能性の，命題論理の充足可能性への帰着)**.** L を言語，$L(C)$ を L の Henkin 拡張とする．L-閉論理式の集合 T について，次の 3 条件は同値である．

1. T はモデルを持つ．
2. T のモデルとなる $L(C)$-標準構造 $\mathcal{M}$ が存在する．
3. $T \cup T_{\text{Henkin}} \cup \mathbf{Eq}$ を命題論理式の集合と見た時に充足可能である．

これを用いて，一階論理のコンパクト性が証明できる．

定理 1.4.23 (Compactness theorem)**.** T を一階論理の閉論理式の集合とする．次の 2 条件は同値である．

1. 任意の有限部分 $T_0 \subset T$ はモデルを持つ．
2. T はモデルを持つ．

［証明］**.** 1.⇒2. を示す．任意の $T_0 \subset T$ がモデルを持つならば，補題 1.4.22 より，$T_0 \cup T_{\text{Henkin}} \cup Eq$ は命題論理式の集合として充足可能である．従って，命題論理のコンパクト性より，$T \cup T_{\text{Henkin}} \cup Eq$ も充足可能である．よって，再び補題 1.4.22 より，T は充足可能である． □

系 1.4.24. $T \cup \{\psi\}$ について，$T \models \psi$ とわかるなら，この時すでにある有限部分 $T_0 \subset T$ について $T_0 \models \psi$ である．

Henkin 拡張

Henkin 拡張によって，一階述語論理のコンパクト性は命題論理のコンパクト性から導かれる．見事な接続を補題 1.4.22 で見せられた．これは一体どう言った現象を捉えたのか．

1.5　完全性定理

証明論に話が移る：「論理法則」や「証明」は数学的に明晰に定義できるだろうか？最初に提出された方針は，「φ には T から論理法則のみによって従う（＝モデルの取り方に依らない)」は「仮定 T からの推論のみを用いた φ の証明がある」と同値として定式化する枠組みであった．これが完全性定理の実際である，ということである．ある意味で，一階論理のモデルを介した意味論は，「解釈を超えたところの消息」を「論理法則」として捉える見方である．

1.5.1　証明体系 H

記法 1.5.1. L を言語とする．論理記号は簡単のためならば $\to$，矛盾命題 $\bot$，存在量化記号 $\exists$ のみとする（この 3 つで complete である！）．

$\varphi \to \psi \to \theta$ は右結合 $\varphi \to (\psi \to \theta)$ の略記である．その他の論理記号は，次の省略記

法として定義する．

$$\begin{aligned}
\neg\psi &:\equiv \psi \to \bot \\
\varphi \vee \psi &:\equiv \neg\varphi \to \psi \\
\varphi \wedge \psi &:\equiv \neg(\varphi \to \neg\psi) \\
\varphi \leftrightarrow \psi &:\equiv (\varphi \to \psi) \wedge (\varphi \leftarrow \psi) \\
\forall x\varphi &:\equiv \neg\exists x\neg\varphi
\end{aligned}$$

定義 1.5.2 (free occurrence, free term)．φ を論理式，t を項，x を変数とする．

1. x が φ に**自由出現する**とは，φ の長さについての帰納法で定義する．$\exists x(\cdots)$ の $(\cdots)$ 内の記号列の部分のことを $\exists x$ の **scope** という．

(a) x は $\bot$ に自由に現れていない．
(b) φ が原子論理式 $R(t_1, \cdots, t_n)$ または $t_1 = \cdots = t_n$ の時には，x が φ に自由に現れているのは，x が式 $t_1, \cdots, t_n$ のいずれかに現れている場合のみである．
(c) $\varphi \equiv \psi \to \theta$ の時は，x が ψ に自由に現れているか，または，x が θ に自由に現れているかの，少なくとも一方が成立するときのみである．
(d) $\varphi \equiv \exists x\psi$ には，x は φ に自由に現れていない．
(e) $x \neq y$ について $\varphi \equiv \exists y\psi$ の時，x が ψ に自由に出現する場合のみである．

2. t が φ において x に関して自由である (t **is free for** x **in** φ) とは，φ 内の x のどの自由出現も，t に現れるいかなる変数 y の scope の中には含まれないことをいう．即ち，x に t を代入する時に，t にどんな変数が含まれていても束縛されないことを保証する．これを φ の長さについての帰納法で次の様に定義する．

(a) x が φ に自由出現していない時．
(b) $\varphi \equiv \psi \to \theta$ の時は，t が ψ, θ の両方において x に関して自由である時．
(c) $x \neq y$ について $\varphi \equiv \exists y\psi$ の時，t が ψ において x に関して自由であり，かつ y が t に現れていない時，t は φ において x に関して自由である．

3. 式 s 中の全ての x の自由出現に対して t を代入した結果 $s[x := t]$ を次の様に s の長さに関する帰納法で定義する．

(a) s が定数の時，$s[x := t] \equiv s$.
(b) $s \equiv x$ の時，$s[x := t] \equiv t$.
(c) $s \equiv x \neq y$ の時，$s[x := t] \equiv s$.
(d) $s \equiv f(s_1, \cdots, s_n)$ の時，$s[x := t]$ は $f((s_1[x := t]), \cdots, (s_n[x := t]))$

4. 論理式 φ 中の全ての x の自由出現に t を代入した結果 $\varphi[x := t]$ を φ の長さに関する帰納法で次の様に定義する．

(a) $\varphi = \bot$ の時，$\varphi[x := t] = \bot$ である．
(b) $\varphi \equiv R(t_1, \cdots, t_n)$ の時，$\varphi[x := t] \equiv R((t_1[x := t]), \cdots, (t_n[x := t]))$ である．
(c) $\varphi \equiv \psi \to \theta$ の時，$\varphi[x := t] \equiv (\psi[x := t] \to (\theta[x := t])$ である．
(d) $\varphi \equiv \exists x\psi$ の時，$\varphi[x := t] \equiv \varphi$ である．
(e) $x \not\equiv y$ について $\varphi \equiv \exists y\psi$ の時，$\varphi[x := t] \equiv \exists y(\psi[x := t])$ である．

注 1.5.3. こうして，代入操作を定義したが，$\varphi[x := t]$ が安全に行えるのが，t が φ において x に関して自由であるときのみである．例えば，$\varphi \equiv \forall y(y = x)$, $y \equiv t$ の時，$\varphi[x := t]$ は恒真命題になってしまう．

証明体系 H をその公理と推論規則を与えることで定義する．

定義 1.5.4 (proof calculus / proof system)**.** 証明体系 H を，次の公理と推論規則の組とする．ただし，φ, ψ は任意の L-論理式であり，以下は公理図式／推論図式である．

H の公理

1.（古典）命題論理の公理：$\to$ I の empty discharge，$\to$ E，$\bot$E の absurdity rule，$\neg$E の二重否定除去則．

$$(K)\varphi \to \psi \to \varphi.$$
$$(S)(\varphi \to \psi \to \theta) \to (\varphi \to \psi) \to \varphi \to \theta.$$
$$(\bot)\bot \to \varphi.$$
$$(\neg\neg)\neg\neg\varphi \to \varphi.$$

2. 等号公理（定義 1.4.21 の 5 つ）
3. 量化公理（定義 1.4.16 の 2 つ）2 つの間に論理的な従属関係があるので実質，次の 1 つである．任意の L-項 t について，t は φ において x に関して自由である時，

$$\varphi(t) \to \exists x\varphi(x)$$

H の推論規則

1. Modus Ponens

$$\frac{\varphi \to \psi \quad \varphi}{\psi}\ (MP)$$

2. 全称化 (Generalization) : ∃I にあたる．変数 ν は $\exists y\psi(y)$ と φ に自由に出現していない記号として，

$$\frac{\psi(\nu) \to \varphi}{\exists y\psi(y) \to \varphi}\ (\exists)$$

変数 ν は自由に出現する必要があるというのは，$\psi(y)$ を満たすということ以外では文脈ψ, φ に影響を受けてはならない ν をとって示す必要がある，ということである．

定義 1.5.5 (H-proof)**.** 閉論理式の集合 T について，H における T からの論理式 φ の証明とは，$\varphi \equiv \psi_n$ を満たす論理式の有限列 $(\psi_1, \cdots, \psi_n)$ であって，各 $\psi_i (i = 1, 2, \cdots)$ は次のいずれかを満たすものをいう．

1. ψ_j $(0 < j < i)$ から H の推論規則を施して得られたものである．
2. $\psi_i \in T$ である．
3. ψ は H の公理である．

定義 1.5.6 (provable)**.** φ が T から証明可能である（$T \vdash \varphi$）とは，φ の T からの証明 $n \to Fml_L$ が存在することをいう．

従って，証明を構築することが主要な仕事になる．この形式的証明 $(\varphi_1, \cdots, \varphi_n)$ は，その木構造を尊重して，証明木の形で記述される．

定義 1.5.7 (proof tree)**.** 証明体系 H の**証明木**とは，次の様に帰納的に定義される．

1. H の公理は，その公理自身の証明木である．
2. H の推論規則の前提となる論理式の証明木を，その推論規則の上に載せれば，その推論規則の結論の証明木になる．

定理 1.5.8 (Soundness theorem)**.**

$$T \vdash \varphi \Rightarrow T \vDash \varphi$$

定理 1.5.9 (Completeness theorem, K. Gödel)**.**

$$T \vdash \varphi \Leftrightarrow T \vDash \varphi$$

述

語 $\vdash$ は証明体系について，述語 $\models$ は論理の意味論にて定義される記号である．証明体系 (proof calculus) の設計が成功していることは，健全性定理 $T \vdash \varphi \Rightarrow T \models \varphi$ の成立を以て確かめられる．その逆も成り立つとき，即ちモデルの言葉で定義される論理的帰結が，全て証明可能になる様な「完全」な証明体系を作れていることは，完全性定理の成立を以て確かめられる．
それにしてもこんな証明体系 H が完全になるとは思わなかった．論理記号は $\to$ と $\bot$ と $\exists$ だけで完全なのか．

1.5.2　命題論理の完全性定理

完全性定理もコンパクト性定理と同様，命題論理での成立を示してから述語論理にリーチする．

定義 1.5.10 (証明体系 H の命題論理部分)**.** 証明体系 H_0 を，論理記号 $\to, \bot$ からなり，公理 $(K), (S), (\bot), (\neg\neg)$ と推論規則 (MP) からなる H の部分証明体系とする．命題論理式の有限集合 Γ からの命題論理式 φ の証明が H_0 で存在することを $\Gamma \vdash_0 \varphi$ と表す．

注 1.5.11. $\Gamma \vdash_0 \varphi$ であるとき，$\Gamma \cup \{\varphi\}$ に現れる命題変数に述語論理の閉論理式を代入した結果 Γ_1, φ_1 について，$\Gamma_1 \vdash \varphi_1$ が成り立つ．命題論理とはそういうものである．

しかし特に $\vdash_0 \varphi$ のときは，閉じているとは限らない一般の論理式を代入しても $\vdash \varphi_1$ となる．

補題 1.5.12.

$$\vdash_0 \varphi \to \varphi$$

定理 1.5.13 (Deduction theorem (propositional))**.** 命題論理式の有限集合 Γ と論理式 φ, ψ について，

$$\Gamma \cup \{\varphi\} \vdash_0 \psi \Rightarrow \Gamma \vdash_0 \varphi \to \psi$$

注 1.5.14. これは型変数の計算でも見た．この計算性を hack するとまた別の証明体系が作れそうだな．

系 1.5.15 (Deduction theorem (predicate))**.** 閉論理式の有限集合 Γ と論理式 φ, ψ について，

$$\Gamma \cup \{\varphi\} \vdash \psi \Rightarrow \Gamma \vdash \varphi \to \psi$$

補題 1.5.16.

1. $\neg\varphi \vdash_0 \varphi \to \psi$
2. $\psi \vdash_0 \varphi \to \psi$
3. $\varphi, \neg\psi \vdash_0 \neg(\varphi \to \psi)$
4. $\varphi \to \psi \vdash_0 \neg\psi \to \neg\varphi$
5. $\varphi \to \psi, \neg\varphi \to \psi \vdash_0 \psi$

補題 1.5.17. 命題論理式の有限集合 Γ について，

$$\Gamma \cup \{\neg\varphi\} \vdash_0 \psi かつ \Gamma \cup \{\varphi\} \vdash_0 \psi ならば \Gamma \vdash_0 \psi$$

補題 1.5.18. 命題論理式 φ は $p_1, \cdots, p_n$ 以外の命題変数を含まないとして，次が成り立つ．

$$\nu(\varphi) = 1 \Rightarrow p_1^\nu, \cdots, p_n^\nu \vdash_0 \varphi$$
$$\nu(\varphi) = 0 \Rightarrow p_1^\nu, \cdots, p_n^\nu \vdash_0 \neg\varphi$$

定理 1.5.19 (completeness theorem (propositional))**.** 命題論理式 φ が tautology ならば，証明体系 H_0 で証明可能である．

定義 1.5.20 (consistent)**.** Γ を命題論理式の集合とする．$\Gamma \nvdash_0 \bot$ が成立するとき，Γ は**命題論理で無矛盾**であるという．

系 1.5.21. 命題論理式の集合 Γ が無矛盾ならば，充足可能である．

1.5.3　述語論理の完全性定理

1.6　定義による拡張

1.7　始切片，終延長と有界論理式

有界論理式というデータ構造を，論理式の上に定義する．始切片，終延長という構造を構造の方に入れることによって．

記法 1.7.1. $<$ を 2 変数関数記号とする．

有界論理式を定義する．同時に，有界論理式よりも定義を緩め，非有界な量化の仕方について 2 つのパターン Σ_n, Π_n を定義する．

定義 1.7.2 (bounded / restricted formula)**.**　1. $<$ に関する**有界量化記号 (bounded / restricted quantifier)** とは，変数 x を含まない式 t について，$\exists x < t\ \varphi :\Leftrightarrow \exists x[x < t \wedge \varphi]$ または $\forall x < t\ \varphi :\Leftrightarrow \forall x[x < t \to \varphi]$ の形の量化のことをいう．

2. 論理式 φ が $<$ に関する**有界論理式**もしくは **Δ_0-論理式**であるとは，φ 中の量化記号が全て有界であることをいう．

3. 論理式 φ が $<$ に関する **Σ_n-論理式**であるとは，非有界存在量化記号のブロック $\exists \vec{x_1}$（空で良い）から始まり，次に全称量化記号のブロック $\forall \vec{x_2}$ と入れ違いに量化記号のブロックが n 個続いた後に，Δ_0-論理式である母式が続く形の論理式のことをいう：

$$\varphi \equiv \exists \vec{x}_1 \forall \vec{x}_2 \cdots Q \vec{x}_n \theta \quad \theta \in \Delta_0,\ Q = \begin{cases} \exists & n \text{ is odd,} \\ \forall & \text{otherwise.} \end{cases}$$

4. 同様にして，**Π_n-論理式**を定義する：

$$\varphi \equiv \forall \vec{x}_1 \exists \vec{x}_2 \cdots Q \vec{x}_n \theta \quad \theta \in \Delta_0,\ Q = \begin{cases} \forall & n \text{ is odd,} \\ \exists & \text{otherwise.} \end{cases}$$

こうして定義された有界論理式／ Δ_0-論理式の意味が変わらない範囲の構造同士の関係を考える．「意味が変わらない」という関係を「絶対的な論理式」という概念で捉える．すると，Δ_0-論理式は絶対的である．これよりも緩い論理式として，Δ_1-論理式は簡単な特徴付けを満たせば絶対的であることが言える．そんなデータ構造が論理式の上に定義できた．

定義 1.7.3 (initial segment, end extension)**.** $\mathcal{M}, \mathcal{N}$ を関係記号 $<$ を含むある言語に対する構造とする．$\mathcal{M}$ が $\mathcal{N}$ の**始切片**，または $\mathcal{N}$ が $\mathcal{M}$ の**終延長**であるとは，次が成り立つことをいう．

1. $\mathcal{M}$ は $\mathcal{N}$ の部分構造である．
2. $\forall a \in |\mathcal{N}|\ \forall b \in |\mathcal{M}|\ [\ a <^{\mathcal{N}} b \to a \in |\mathcal{M}|\ ]$

この関係を，関係記号 $<$ が明らかな時，単に $\mathcal{M} \subset_e \mathcal{N}$ と表す．

定義 1.7.4 (absolute, Δ_1)**.** 公理系 T の言語に 2 変数関数記号 $<$ が含まれるとする．

1. 論理式 $\varphi(\vec{v})$ が $<$ と T **に関して絶対的**であるとは，任意の T のモデル $\mathcal{M}, \mathcal{N} \vDash T$ で $\mathcal{M} \subset_e \mathcal{N}$ を満たすものについて，$\forall \vec{a} \subset |\mathcal{M}|\ \{\mathcal{M} \vDash \varphi[\vec{a}] \Leftrightarrow \mathcal{N} \vDash \varphi[\vec{a}]\}$ を満たすことをいう．$T = \varnothing$ の時は，単に論理式 φ が絶対的であるという．

2. 論理式 φ が T に関して Δ_1-論理式であるとは，それが T 上である Σ_1-論理式ともある Π_1-論理式とも同値となることをいう：$T \vDash \varphi \leftrightarrow \varphi_\exists \leftrightarrow \varphi_\forall$

補題 1.7.5. 次の 2 条件が成り立つ．

1. $<$ に関する Δ_0 論理式は，$<$ に関して絶対的である．
2. $<$ と公理系 T に関する Δ_1 論理式は，$<$ と T に関して絶対的である．

［証明］．Δ_0 論理式の長さに関する帰納法より． □

1.7.1　自然数の公理系群 PA

（ベースとなる証明体系が NK か，LK か，古典ラッセルヒルベルト系 RHK かに依らず）Peano の公理を充す構造 (Peano 構造) は同型を除いて一意に定まる．これを自然数という．ただし，"野生の"数学的帰納法の公理は一階論理の論理式ではない．

$$\forall X \subset \mathbb{N}\,(0 \in X \wedge \forall x \in X[\, x+1 \in X\,] \to \forall x \in \mathbb{N}[\, x \in X\,])$$

これはモデルの議論領域の任意の部分集合について述べているためである．解決法としては，問題になる部分集合を一階論理で定義できる集合，即ち論理式に限ることで，PA に組み込む．すると，自然数についての一階理論 Peano Arithmetic を得る．

定義 1.7.6 (PA の言語)**.** 言語を $L(PA) := \{0/0, 1/0, +/2, \cdot/2, E/2, < /2\}$ とする．$E(x,y) = x^y$ とする．

注 1.7.7. $E, <$ は他の記号のみに関する数学的帰納法を含む公理から定義可能であるから，独立ではない．

定義 1.7.8 (PA の公理)**.**

1. 関数記号の公理
2. 順序の公理
3. 数学的帰納法の公理図式：任意の論理式 $\varphi(x)$ について，$\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(x+1)) \to \forall x \varphi(x)$.

1.,2. のみからなる公理系を PA^- とする．

定義 1.7.9 (numeral)**.** 自然数 n を標準的に表す閉式を数字といい $\overline{n} :\equiv ((\cdots((0+1)+1)\cdots)+1)$

命題 1.7.10 (value)**.** 各閉式 t はある自然数 n を表す．これを式 t の**値**と言う．

$$\exists! n\,[\,\mathrm{PA}^{-} \vDash t = \overline{n}\,]$$

定義 1.7.11 (standard model)**.** 自然数を領域とし，L(PA) の記号を標準的に解釈したモデルを標準モデルという．

これ以外のモデルを**超準モデル (nonstandard model)** という．

命題 1.7.12. PA には超準モデルが存在する．

［**証明**］**.** 定数記号 c と，それに付随して無限個の公理

$$\overline{n} < c\ (n = 0, 1, 2, \cdots)$$

を PA に加える．この公理系の任意の有限部分について，標準モデルで c を，その公理系の有限部分で言及している数字 $\overline{n}$ たちに対して十分大きく解釈すればモデルを持つ．従ってコンパクト性定理より，どんな数字 $\overline{n}$ よりも大きい「無限大」c をもった PA のモデルは存在する． □

その他の構造|,-はこれらの原始的記号の組み合わせの略記となる．これらをあたかも関数記号とみなして使用しても保存拡大になる（1.6 節）．

1.7.2　PA-NK

定義 1.7.13 (Peano の公理)**.** PA-NK は，NK の推論規則に数学的帰納法の公理図式と，Peano の公理系を加えた証明体系である．

1. $\forall x\ (\neg Sx = 0)$
2. $\forall x \forall y\ (Sx = Sy \to x = y)$
3. $\forall x\ (x + 0 = x)$，$\forall x \forall y\ (x + Sy = S(x + y))$
4. $\forall\ (x \cdot 0 = 0)$，$\forall x \forall y\ (x \cdot Sy = (x \cdot y) + x)$

1.7.3　帰納法

元の数学的帰納法の公理の代わりに，累積的帰納法が採用されることもある．その根拠を示す．

$$\frac{\Phi(0) \quad \forall x(\Phi(x) \to \Phi(Sx))}{\forall x \Phi(x)} SMI$$

定義 1.7.14 (Scheme of Accumulative Induction)**.**

$$\frac{\forall x(\forall y(y < x \to \Phi(y)) \to \Phi(x))}{\forall x \Phi(x)} SAI$$

ただし，$x < y :\Leftrightarrow \exists z(x + Sz = y)$ とした．

命題 1.7.15 (累積的帰納法)**.**

$$PA \vdash_{NJ} SAI$$

数学的帰納法は本当に不思議なものだが，帰納的定義に付随するデータ構造の性質の証明と捉えるのが一番見通しが良かった．自然数の性質は基本的には数学的帰納法で届く範囲のものであるので，数学的帰納法による証明を持つ．一方で，自然数を形式的な定義（＝帰納的な定義）と捉えるのではなく，「数の概念」として特徴付けから考えることも出来る．その複雑な自由度をどう理解するか．

1.7.4　Heyting 算術 HA

ペアノ算術の直観主義論理版と言うべき「戻ってこれない」世界と考えられる一階理論に，ハイティング算術がある．最小数原理などを除いて，殆どがそのままペアノ算術の定理でもある．

第 2 章

不完全性定理

不完全性定理は，再帰的に公理化されるよくある感じの数学の公理系は必ず超準モデルを持つ，つまり「有限性」概念が理論の中と外で食い違うことがあることを示しました．これは，私たちが「有限」とは何かわかってるつもりですが，実は大変難しい概念であることを示します．証明図とは「有限の長さの文字列」の一種なので，有限性概念こそ無矛盾性の根幹にある訳です．Hilbert が「有限の立場」と言って算術上での無矛盾性証明に拘った理由がそれです．[†1]

[†1] 矢田部俊介先生

第 II 部

計算理論

授業「計算の理論」を基調として，新井敏康先生の本をまとめた．
まず，数学的に厳密な構成を持つ自然数を基調として，数学基礎論の文脈から計算理論を立てる（再帰理論）．実際のプログラムも有限な記号群 (alphabet) しか使わないからそれは自然数でコードできる．従って，プログラム，入出力，計算過程は全て自然数だと思って理論を立てる．この発想は最初にこの手法を用いた人物の名前から，「ゲーデル数化」とも呼ばれる．
次に，数学的に厳密な定式化が出来る決定問題を中心として，アルゴリズム論を見る（計算複雑性理論）．
最後に，計算可能性と論理と証明を見る．

1932 年に Herbrand、および 1934 年に Gödel が、原始帰納的関数と呼ばれる自然数上の関数の明示的構成法を拡張して帰納的関数（もしくは一般帰納的関数）と呼ばれる関数の構成法を作り上げた。1933 年から 1935 年ごろには、チャーチやクリーネがやはり関数の構成的な定義法であるラムダ記法を用いて定義可能な関数のクラスを定めた。さらに、1935 年から 1936 年にかけてポストとチューリングは、チューリングマシンの概念を用いてこの理念的計算機械で実行可能なプログラムのクラスを定めた。こうしてほぼ同時期に出現したさまざまな計算できる数論的関数のクラスは、実は互いに同じものであることが証明された。これにより、それまで非形式的に「実質的に計算できる関数」(effectively computable function) と呼び慣わされていたこのクラスをもってして「計算できる関数」とみなそうという主張がなされることになった。これがチャーチ＝チューリングのテーゼと呼ばれている主張である。この意味で計算できる関数はチューリング計算可能な関数、あるいは単に計算可能関数と呼ばれるようになった。この主張自体はチャーチ、チューリング論文を参照して 1943 年にクリーネによってなされた。[†2]

[†2] https://ja.wikipedia.org/wiki/チャーチ＝チューリングのテーゼ

第 3 章

Recursion Theory

「自然数から自然数への関数が計算可能であるとはどういう意味か？」「計算不能関数は、その計算不能性のレベルに基づいて階層分けできるか？」が主な問題意識である．以降，集合といえば自然数全体の集合を，関数といえば自然数の部分関数，すなわち $\mathbb{N}^n$ の部分集合を定義域とした関数を考える．

TM(Turing Machine), RAM(Random Access Machine), PRAM(Parallel RAM) などの計算モデルは，多項式時間でエミュレート可能？

3.1 μ 再帰的関数

部分関数 $\mathbb{N}^n \rightharpoonup \mathbb{N}$ のうち，特別なクラスを定義する．なお，部分関数について $f(x) \simeq g(x)$ とは，定義されない場合も含めて取る値が等しいことを意味する (strong equality).

定義 3.1.1 (recursion function (Kleene, 1952)). $x = (x_0, \cdots, x_n), y, z \in \mathbb{N}$ とする．次の 5 通りの関数定義法を定め，それらの関数構成法について閉じている最小の関数のクラスとして，A,B,C の関数のクラスを定める．

1(合成). $G(y_1, \cdots, y_n), H_1(x), \cdots, H_n(x)$ がいずれも計算可能ならば，$F(x) \simeq G(H_1(x), \cdots, H_n(x))$ も計算可能である．計算過程を繋げて続けて計算すれば良い．

2(再帰的定義 (primitive recursion)). $G(x), H(y, x, z)$ が計算可能であるとき，

$$F(0, x) \simeq G(x)$$

$$F(y + 1, x) \simeq H(y, x, F'y, x)$$

と定義された F も計算可能である．G, H に従って計算すれば良い．

3(同時に行われる再帰的な定義 (simultaneous recursion)). $G_i(x), H_i(y, x, z_0, z_1)$ $(i = 0, 1)$ が計算可能とする.

$$F_i(0, x) \simeq G_i(x)$$

$$F_i(y + 1, x) \simeq H_i(y, x, F_0(y, x), F_1(y, x))$$

と定義された F_0, F_1 は計算可能である. 同様に, 順番に G, H に従って計算すれば良い. 定義に要する複雑な手順に対する自由度に対して言及している項目である.

4(μ 作用素). (特性関数が) 計算可能な関係 $R(x, y)$ を用いて, $F(x) \simeq \mu y.R(x, y)$ 即ち

$$F(x) \simeq y :\Leftrightarrow R(x, y) \wedge \forall z < y\ \neg R(x, z)$$

によって定義された F も計算可能である. 即ち, mu 作用素とは, $k + 2$ 変数自然数値関数 $f(y, x_0, \cdots, x_k)$ から $k + 1$ 変数自然数値関数 $\mu y.f(y, x_0, \cdots, x_k)$ に写す作用素で, $\mu y.f$ は $f(y, x_0, \cdots, x_k)$ を満たす最小の y を, そのような y が存在しなければ定義不能性とした関数である. 0 から順番に, 条件 $R(x, y)$ が成立するときの y を探していけば良い. これを最小化作用素 μ による定義という. これは, 関係 $R(x, y)$ に依って, ある x についてこれを満たす y が存在せず, 関数 $F(x)$ の値が定義されないことがある.

5(場合分けによる定義). $R_1, \cdots, R_n$ を計算可能関係で, どの x についても n 個のうちどれか 1 つの条件のみが成立するとする. $F_1, \cdots, F_n$ を計算可能関数として,

$$F(x) \simeq \begin{cases} F_1(x) & R_1(x) \text{ のとき} \\ \vdots & \\ F_n(x) & R_n(x) \text{ のとき} \end{cases}$$

と定義された F も計算可能である. 毎度, R_1 から順に検証して, R_i で hit した際に F_i で値を定めれば良い.

A(initial function). 射影 $\mathrm{pr}_i^k(x_1, \cdots, x_k)$ $(1 \leqslant k, 1 \leqslant i \leqslant k)$, 後者関数 $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$, 0 変数関数 $\mathrm{zero}() = 0$ 即ち $i_\varnothing : \varnothing \to \varnothing$ の 3 つを初期関数と呼ぶ.

B((general) recursive function, μ-recursive function). 初期関数から, 合成・再帰的定義・最小化作用素による定義の 3 種を有限回施して得られる部分関数の閉包を再帰的 (部分) 関数という.

C(primitive recursive function). 初期関数から, 合成・再帰的定義によって定義される関数を原始再帰的関数という. これは部分関数ではなく, 自然数全体で定義される. 原始再帰関数全体の集合を PR とも書く.

これは「計算可能な関数」の一番直観的なクラスとして立てられた.

命題 3.1.2. 2 項演算 $+, \cdot$，2 項関係 $x = y, x \leqslant y, x < y, x|y, x$ is a prime number は全て原始再帰的である．

［証明］．足し算や掛け算は，後者関数 $+1$ を用いて，$x+0 = x, x+(y+1) = (x+y)+1$ とすれば良い．□

命題 3.1.3. 原始再帰的な関数・関係は，

1. 命題論理の結合子
2. 場合分けによる定義
3. 累積足し算

［証明］．関数 $\chi_R(x)$ が計算可能なら，$1 - \chi_R(x) =: \chi_{\neg R}$ も計算可能．関数

$$\mathrm{isnotzero}(x) = \begin{cases} 0 & x = 0 \\ 1 & otherwise \end{cases}$$

が計算可能であるとき，$(\chi_{R\vee Q}(x) := \chi_R(x), \chi_Q(x))$ も，定義 3.1.1.2 より計算可能である．□

数論が扱う関数の多くや、実数を値とする関数の近似は原始再帰的であり、加法、除法、階乗、指数、n 番目の素数を求める関数などがある (Brainerd and Landweber, 1974 年)．実際，原始再帰的でない関数は見つけるのも難しい．

命題 3.1.4 (Ackermann function)**.** Ackermann 関数 $\mathbb{N}^2 \to \mathbb{N}$ を次のように定める．これは，原始再帰関数ではないが，μ 再帰的な全域関数である．

3.2　Church-Turing's thesis

ここで，次の 2 点について考える．

命題 3.2.1 (関係 comp)**.** 関係 $\mathrm{comp}(e, x, y)$ を，「列 $y = (y_0, \cdots, y_k)$ は，項数 n のプログラム e に，入力 $x = (x_0, \cdots, x_n)$ を与えた環境での，k ステップまでの計算過程のコードである」とする．これは計算可能である．

まだ計算可能性の定義はしていないが，模倣して試して y に一致するかを確認すれば良い．

命題 3.2.2 (Kleene の T-predicate)**.** 各 n について，関係 $T_n(e, x, y)$ を，「項数 n のプ

ログラム e について $\mathrm{Comp}(e, x, y)$ かつ $y = (y_0, \cdots, y_k)$ の最後 y_k の計算状態は計算の終了状態である」とする．これは計算可能である．

なお，このとき y を計算数 (computation number) という．

命題 3.2.3. 計算数 y について，計算結果を $U(y)$ とする．これも計算可能である．

定義 3.2.4 (Church-Turing's thesis)**.** 計算可能な関数とは，再帰的関数 (μ-recursive function)（定義 3.1.1.B）のことである．

注 3.2.5. この定義により，計算が可能な関数とは，その計算を実行できるような有限のアルゴリズムが存在するような関数，よっておおよそコンピュータで実行できる関数と同じだと考えられる．このように，この定義は「計算可能な関数」に対する数学的定義を与えることに加えて，日常的・実際的な意味も持ち，「計算とは何か」という問題に対する答えや果てには「理論構築の方向性の是非」などという文脈上でも議論がなされるため，単に定義とは言わずに含みを持たせて「提唱」「定立」などと言われる．

現在，等価な関数族を計算可能と満たすような自然な計算の理論が複数見つかっているので（命題 3.5.4），このパラダイムは数学的に安定したものだという経験事実が蓄積しており，幅広く受け入れられていると言える．

3.3　レジスター機械による計算

レジスター機械による計算に関する Kleene の T-述語が原始再帰的に構成できることを見る．

定義 3.3.1 (レジスタ機械)**.** レジスタ機械は，（レジスター，プログラム格納庫 (program holder)，カウンター）の 3 つ組である．
1. レジスタ $R_0, R_1, \cdots$ は可算無限個あり，計算の各時点で 1 つのレジスタは自然数 1 つを格納する．
2. プログラム格納庫にはプログラムが一つ格納される．プログラムとは，「命令」の空でない有限列であり，命令は番号 $(0), (1), \cdots, (N-1)$ で番号付けされており，このとき N をプログラムの長さという．
3. カウンターには，計算の各時点で括弧付きの数字 (k) が 1 つ格納されている．この番号に対応する命令が次に実行される．

定義 3.3.2 (命令)**.** 命令は次の 3 種類である．
1(INCREASE R_i)．レジスタ R_i に格納されている数に 1 を加えて，カウンターを increment する．
2(DECREASE R_i)．$R_i = 0$ のとき，何もせずにカウンターを increment する．そうでない場合，$R_i = R_i - 1$ とし，カウンターを increment する．
3(GO TO (n))．カウンタの番号を (n) とする．

計算は次のように進行する．
1. プログラム格納庫にプログラムを，レジスタに任意の自然数を格納し，カウンターを (0) にする．
2. カウンタの番号 (k) $(k \leqslant N-1)$ の命令を実行する．$k \geqslant N$ の場合は計算を停止する．

定義 3.3.3 (プログラムが計算する部分関数)**.**

3.4　Turing Machine

計算は transition であるという離散的モデル．

定義 3.4.1 (Turing Machine). 次を満たす 7 組 $M = (\Sigma, I, b; Q, q_0, q_h; \delta)$ をチューリング機械という.
1. Σ は alphabet と呼ばれる記号の有限集合で，その元を tape symbol という. $\varnothing \neq I \subset \Sigma$ で，I の元を input symbol という. $b \in \Sigma \setminus I$ を blank symbol という.
2. Q も有限集合で，その元を state という. $q_0 \in Q$ を initial state，$q_h \in Q$ を halting/accepting state という.
3. 部分関数 $\delta : \Sigma \times (Q \setminus \{q_h\}) \nrightarrow \Sigma \times Q \times \{L, R\}$ を transition function という.
A. state register を持った有限オートマトン $(I \cup \{b\}, Q, q_0, \delta, \{q_h\})$ を，本体と呼ぶ. 本体には各 cell を 1 つ制御 (scan) 出来る head が付いていて，可算無限の長さを持つ tape 上の文字を書き換えることが出来る. tape は可算無限個の cell に別れていて，初め，有限個の記号 $\Sigma \setminus \{b\}$ と可算無限個の b が並んでいるとする. $\delta(q, a) = (q', a', D)$ の時，本体の状態を q' に遷移させ，tape の着目位置に a' を書き込み，head を D 方向に 1 つずらすことを意味する. 従って，本体は，有限な tabel of instructions を持つ事になる.
B. 記号の集合 Σ に対して，それを用いた有限列全体の集合を $\Sigma^* = {}^{<\omega}\Sigma$ と定める. 無限列 $s \in {}^{\omega}(\Sigma \cup Q)$ であって，Q の元はちょうど 1 回，b 以外の記号は全て高々有限個しか出現しないものを，M の状況という.
C. 3 つ組 (q, α, i) で，$q \in Q, \alpha = a_1, \cdots, a_l \in \Sigma, 0 \leqslant i \leqslant l+1$ を満たすものを，M の時点表示 (instantaneous description) という.
D. 停止状態に到るまでの δ の適用回数を，実行時間または time complexity と呼ぶ. scan された cell の個数を space complexity という.

記法 3.4.2. 今回の alphabetΣ を Γ とし，I を Σ とする流儀もある. 特定の終了状態 q_h 1 つを指定する代わりに，部分集合 $H \subset Q$ を指定することもある. 現状のコンピュータでは，$I = \{0, 1\}, b = 0, \Sigma = I \cup \{b\} = \{0, 1\}$ である.

例 3.4.3 (3-state busy beaver).

例 3.4.4 (偶奇判定アルゴリズム). $Q = \{q_0, q_1 = q_Y, q_2 = q_N, \cdots, q_r\}, F = \{q_Y, q_N\}, \Sigma = \{0, 1\}, \Gamma = \Sigma \cup \{b\}$ とし，δ を次のように定める.

定義 3.4.5 (部分関数への翻訳埋め込み)**.** Turing 機械 M と input size と呼ばれる $\mathrm{arity} n \in \mathbb{N}$ とに対して，次の I^* 上 n 変数部分関数 f_n^M が定まる．

1. 入力 $\alpha_1, \alpha_2, \cdots, \alpha_n \in I^*$ に対し，それぞれの有限列入力を blank symbol を区切り記号として

$$\cdots \overset{\downarrow}{b} b \alpha_1 b \alpha_2 b \cdots b \alpha_n b b \cdots$$

という状況を整える．

2. 遷移関数を繰り返し適用する．

3. 停止状態 $q_h \in F$ を検出したら計算を止める．

4. 計算が停止し，blank symbol を間に挟まないひとつながりの記号列 $\beta = (\beta_1, \cdots, \beta_m)$ が高々 1 つ tape 上にあり，この β の左端で止まっているものとする．

$$\cdots b b \overset{\downarrow}{\beta_1} \beta_2 \cdots \beta_m b b \cdots$$

この時，$f_n^M(\alpha_1, \cdots, \alpha_n) = \beta$ と定める．

5. M と $\alpha_1, \cdots, \alpha_n$ に対して，このように f_n^M が定まることを，**文字列 $\alpha_1, \cdots, \alpha_n$ を受理する**という．

A. この f_n^M の値域が真理値 $TV = \{\mathrm{TRUE}, \mathrm{FALSE}\}$ などの 2 点集合であるような場合，その問題を判定問題 (decision problem) と呼ぶ．それ以外の問題を関数問題 (function problem) という．

3.5　複雑性クラス

定義 3.5.1 (判定問題の複雑性クラス 1 : P, PSPACE, EXP)**.** 判定問題 Π について，次の用語を定める．問題 Π の入力 $x = x_1 x_2 \cdots x_n$ に対して，$|x| := n$ を input size という．2 つの関数 t, s を次のように定める．

$$t(n) = \max\{t(x) \mid \mathrm{input} x \in \Sigma^n \mathrm{for} \Pi\} \qquad s(n) = \max\{s(x) \mid \mathrm{input} x \in \Sigma^n \mathrm{for} \Pi\}$$

1. P を，Turing 機械で $O(f(n))$（ただし f は多項式）の時間計算資源を使って解くことが出来る問題全体の集合 $P = \mathrm{PTIME}$ と定め，以降同様に $\mathrm{EXP} := \mathrm{EXPTIME}$ と定める．

$$P(\subseteq \mathrm{NP}) \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

となる．

注 3.5.2.

1. この max 関数を使った複雑性の定式化の流儀を，worse case analysis という．

2. PTIME までを tractable という．実行時間の定義上，これに含まれるものは自明に PSPACE である．一方，PSPACE に真に含まれる問題以降を intractable という．実は次が成り立つ．

この定義の 2. の主張を分解すると，次のことが分かっている．

命題 3.5.3. 次が成り立つ．

1. PSPACE である問題は，必ず EXP である．
2. NP である問題は，必ず PSPACE である．

3(Savitch). PSPACE=NPSPACE である．

命題 3.5.4. 次の条件は同値である．

1. 関数 f が計算可能である（定義 3.2.4 より，再帰的関数である）．
2. 関数 f が Turing 機械で有限時間内に実行可能である．
3. 関数 f が λ 計算で定義可能である．

3.6 Non-deterministic Turing Machine と更なる複雑性クラス NP

定義 3.6.1 (非決定性チューリングマシン)**.** δ を多価関数とした 7 組 $M = (\Sigma, I, b; Q, q_0, q_h; \delta)$ を Turing 機械という．

定義 3.6.2 (判定問題の複雑性クラス２：NP)**.** 非決定性 Turing 機械で多項式時間で解くことのできる決定問題のクラスを NP(Non-deterministic polynomial time) と書く．

命題 3.6.3 (NP の特徴付け)**.** 決定問題について，次の２つの条件は同値である．

1. NTM で多項式時間で解くことができる．
2. その問題が多項式時間で解くことができることの，特定の証拠が発見されている場合，そのことが本当に正しいかどうかを確かめるという判定問題（検算）が，多項式時間で解決できる．

例 3.6.4 (Hamilton 経路問題)**.** これは「与えられたグラフについて、全ての頂点を一度だけ通る閉路が存在するか」という問題であり、この問題に対して yes/no を判定するの

は容易ではない (多項式時間で解けるアルゴリズムは知られていない)。しかし yes となる証拠、すなわち実際の閉路が与えられたならば、「本当に全ての点を一度ずつ通っているか」という検証は多項式時間で可能である。したがって、ハミルトン閉路問題は NP に属する。

3.6.1　P≠NP

NTM は基本的に DTM と等価であり，分岐は高々有限だから実際 DTM 上でシュミレート可能である．しかしその際の実行時間の増大の振る舞いについては未解決問題が残る．おそらく NTM の方が速いだろうと思われる．

例 **3.6.5** (Primality Testing).

3.6.2　Oracle Machine

定義 3.6.6 (Oracle machine). 特定の決定性問題を 1 ステップで解くことのできる付加構造を持った Turing 機械を神託機械と呼ぶ.

記法 3.6.7. クラス A のアルゴリズムにクラス B の決定問題を解く oracle を組み合わせることで解ける複雑性クラスを A^B と書く.

$NP \subset P^{NP}$ ではあるが逆が成り立つかはわからない.

第 4 章

計算量とアルゴリズム

計算複雑性理論について深掘りをする．河村彰星先生による講義を参考にして導入していく．

4.1 計算の枠組み

「この問題を解く系統だった方法は存在しない」みたいな主張を確認するためには，頑健で祈りのこもった肥沃な基盤を立てる必要がある．

定義 4.1.1 (問題)**.** 或る記号列の集合 Σ による有限列 ${}^{<\omega}\Sigma$ を定義域かつ値域とした部分関数（をメタ言語で指定したもの）のことを「問題」という．

例 4.1.2. ２つの整数を入力として受け取り，その最大公約数を返す問題は，正確に言えば，

「十進法によって正の整数 a を表す有限列と十進法によって正の整数 b を表す有限列とをコンマで区切って括弧で括ったもの」を入力として受け取ったとき，「a と b との最大公約数 $\gcd(a,b)$ を十進法で表す有限列」を出力する問題．なお，入力される文字列がこの形でなかった場合の結果はなんでも認める．

となる．従って，「275 と 500 の最大公約数は？」は強いていうなら入力 $(275, 500)$ を指すものであって，問題ではない．

例 4.1.3 (Post's Correspondence Problem)**.** 有限列が上下 1 組書かれた札（を表す文字列）が有限種類・各種類可算無限個与えられる．これらを有限枚並べて，上下の文字列を

一致させることが出来るか（を表す文字列 $\{\bigcirc, \times\}$ を返すことが出来るか）？

定義 4.1.4 (算法)**.** 問題の処理の手順を一意に定めたもの．例えば，特定のプログラミング言語によるプログラムと，それを処理し実行する計算機の組など．計算理論では，抽象的な規定としての算法と，それの実装としての Turing 機械とを同一視する．

注 4.1.5 (Church-Turing's thesis)**.** Turing 機械の挙動は極めて機械的で単純であるから，これは計算と呼べるであろうが，直観的に計算として考えたい全ての挙動が Turing 機械として還元できるかは，もはや計算という物理的な事象との対話も必要とするメタ的な問題として残る．これをチャーチ・チューリングの定立という．例えば [2] の頃では，Computer という計算職に就く人間の動きの模倣性が議論の中心であった．しかし，Turing 機械自体が，テープの本数，テープの形，遷移規則の細かな調整（例えば $D = \{L, R, \mathrm{stay}\}$ などと変えること）などに対して安定な概念であることと，全く独立に形成された計算可能性の定義（一般再帰関数，ラムダ計算）と同値になることが判明したことにより，現在では広く受け入れられている．

定義 4.1.6 (算譜)**.** Turing 機械を指定することは，有限個の遷移規則 $\delta(q, a) = (r, b, d)$ を指定することと等価である．従って結局，算法，算譜，機械は全て同一視する．

注 4.1.7. プログラムと言っても，非常に低レイヤーなアセンブリ言語くらいを考えておくと良い．またこれより，算譜を介して，究極的には，全ての機械は自然数を用いてコードできる．

4.2　万能性と計算不能性

ここまで全て Turing (1936) の仕事であるとは考え難いな……．

定理 4.2.1 (UTM の存在)**.** 次の問題を解く機械 U が存在する．[2]

問題 $U(M, x) = M(x)$

1(入力). 機械 M と入力 x を表す二進数表記による自然数からなる組 (M, x) であって，M に x を入力すると停止するもの．

2(出力). 出力 $M(x)$

実際の構成をするには，Turing 機械の自然数によるコードの規則を定めないと始まらないが，方向性としては，M の算譜に応じて遷移規則を変えられる遷移状態 Q を持ち，x

にそれを施すという，ある種プログラム内蔵的な Turing 機械を構成すれば良い．

注 4.2.2 (UTM が Neumann Architecture の興り)**.** multi-tape UTM は，模倣するところの TM で $T(n)$ の時間がかかったとしたら，$5kT(n)^2$ だけで解ける．

定理 4.2.3 (停止性判定機械の非存在)**.** 次の問題を解く機械は存在しない．

> 停止性判定問題
> 1(入力)．　機械 M と入力 x を表す二進数表記による自然数からなる組 (M,x)の全て．
> 2(出力)．M が停止する場合は，出力 $M(x)$．停止しない場合は記号 $\times$．

［証明］**.** 停止性判定問題を解く機械 M_0 の存在を認めて，矛盾を導く．このとき，任意の機械 M と入力 x に対して，その停止性についての情報を得られるのだから，それに挙動を追加しただけの次のような 2 つの機械 M_1, M_2 を構成できる．

> 入力 (M,x) に対して，$M(x)$ が停止するならば停止せず（$\times$ を出力し），$M(x)$ が停止しないならば停止する（$\bigcirc$ を出力する）機械 M_1．
> 入力 x に対して，x をコードされた機械と見たときの機械 x について，$x(x)$ が停止するならば停止せず（$\times$ を出力し），$x(x)$ が停止しないならば停止する（$\bigcirc$ を出力する）機械 M_2．

このとき，$M_1(x,x) \simeq M_2(x)$ が成り立つ．即ち，M_2 は関数 $M_1 : \mathbb{N}^2 \to \{\bigcirc, \times\}$ の定義域を $\Delta \subset \mathbb{N}^2$ に制限し，1 変数にしたものに他ならない．しかしこのとき，機械 $M_2(x)$ は自然数の中にコードされて居らず，そのようなものを作り出してしまったことになる．よって矛盾．　□

注 4.2.4. 今回構成した非存在機械 M_2 を非停止性認識機械と呼ぶ．

4.3　PCP とヒルベルトの決定問題と帰着

4.3.1　PCP と PCPw1 が Turing 等価

PCP：有限列が上下 1 組書かれた札（を表す文字列）が有限種類・各種類可算無限個与えられる．これらを有限枚並べて，上下の文字列を一致させることが出来るかを表す文字列 $\{\bigcirc, \times\}$ を返せ．
PCPw1：有限列が上下 1 組書かれた札と区別されたそのうちの 1 枚（を表す文字列）が有限種類・各種類可算無限個与えられる．これらを，区別された 1 枚を先頭として有限枚並べて，上下の文字列を一致させることが出来るかを表す文字列 $\{\bigcirc, \times\}$ を返せ．

命題 4.3.1. PCP は PCPw1 に帰着する．

［証明］．PCP 問題はカードの枚数 n について，そのそれぞれを最初に使うカードとして指定した場合の n 回の PCPw1 問題に等価である． □

命題 4.3.2. PCPw1 は PCP に帰着する．

［証明］．n 枚の札と 1 枚の指定札からなる PCPw1 問題を考える．計 $n+1$ 枚の札 $\frac{\xi}{\eta} = (\xi, \eta)$（但し ξ, η は記号の有限列）について，ξ 内の任意の記号 x の出現を全て $!x$ で置換し，η 内の任意の記号 y の出現を全て $y!$ で置換した $n+1$ 枚のカードを作成する．次に，指定札について，同じような置換を施した上で η の先頭に！を追加したカードを 1 枚作成する．これは，唯一上下について両方とも！が先頭にくる札であり，一致させるにはこの札を先頭に用いるしかない．最後に，上のカードに記号！が 1 つだけ足りない点を除いて記号列が全て上下一致した場合に，$\bigcirc$ 判定が出来るように，札 $\frac{!?}{?}$ を作成する．こうして作成した計 $n+3$ 枚のカードについての PCP は，所与の PCPw1 に等価になる． □

4.3.2　非停止性の認識問題は PCPw1 に帰着する

HALT：機械 M と入力 x を表す 2 進数表記による自然数の組 (M, x) の全てを入力として取り，M が停止する場合は出力 $M(x)$ を，停止しない場合は記号 $\times$ を出力する．
HALT'：機械 M と入力 x を表す 2 進数表記による自然数の組 (M, x) の全てを入力として取り，M が停止する場合は出力 $M(x)$ を出力してテープ上の文字を全て消去し，ヘッドをテープ左端に揃えてから終了状態に至り，停止しない場合は記号 $\times$ を出力する．

すると，2 つの問題は互いに帰着し，等価である．

命題 4.3.3. 停止前に追加の行動を要求する停止性判定問題 HALT' は，最初のカードの指定つきポストの文字列合わせ問題 PCPw1 に帰着する．

［証明］．まず，Turing 機械の挙動を文字列にコードする方法を準備する．簡単のため，勝手な Turing 機械 $M = (\Sigma, I, b; Q, q_0, q_h; \delta)$ について，そのアルファベットは $\Sigma = \{0, 1\}$ とし，$Q = \{q_0, q_1, \cdots, q_k, q_h\}$ とする．すると各状態は，tape（のうち十分大きくとった有限領域，即ち入力 x と同じ長さ）上に存在する cell 内の記号の列（例えば $010110\cdots010$）の中に，head が指し示す cell に書き込まれている記号の直前に，head の状態 q_i を挿入した文字列（例えば初期状態から 2 回右に遷移した場合 $01q_i0110\cdots010$ など）を用いて表せる．すると，この各状態を表す記号列を，separate symbol % などを用いて区切って繋げることで，或る Turing 機械の挙動全体をコードすることが出来る．

入力 (M, x) に応じて，次のような手順でカードを作る．まず，上部分は空欄のカード $\frac{\quad}{q_0x\%}$ を作成し，これを最初に使うべきカードとして指定する．次に，separate カード $\frac{\%}{\%}$ を追加する．続いて，機械 M のアルファベットに応じて，その元 (tape symbol) 1 つを上下同じ文字書いたカード，従ってここでは $\Sigma_M = \{0, 1\}$ であるからカード $\frac{0}{0}, \frac{1}{1}$ を用意する．次に，遷移規則 δ の算譜に応じて，例えば $\delta(q_i, a, R) = (q_j, b)$ の場合は右遷移カード $\frac{q_ia}{bq_j}$，$\delta(q_i, a, L) = (q_j, b)$ の場合は左遷移カード $\frac{aq_i}{q_jb}$ を作成する．最後に，受理状態 q_h についてカード $\frac{aq_h}{q_h}$ を全てのアルファベット $a \in \Sigma$ について作成する．

すると，これらの有限枚のカードについての PCPw1 問題は，最初のカードの下の内容

を上が追随するために次のカードから始まり，次の separate カードを迎えるまでは，カードの上の文字列は Turing 機械 M の初期状態を表す文字列を模倣する．その時カードの下の文字列としてあり得るパターンは，Turing 機械 M の算譜 δ に記載されていたもののみである．これが separate　カード $\frac{\%}{\%}$ を挟んで続く．最後のカード $\frac{aq_h}{q_h}$（を含むような separate カードで区切られたブロック）を迎えることが出来る場合は，「全ての文字を消去した結果，head は左端に到達した結果停止する」ことに対応するから，◯ を出力，もしそのような並べ方がない場合は Turing 機械が停止しないことがわかるので × を出力すれば良い．　□

4.3.3　ヒルベルトの決定問題は，非停止性の認識問題に帰着する [2]

4.4　計算量の枠組み

> As soon as an Analytic Engine (解析機関) exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time?
>
> — Charles Babbage (1864)

4.5　多項式時間帰着 (polynomial-time reduction)

定義 4.5.1 (polynomial-time reduction)**.**

定義 4.5.2 (判定問題の複雑性クラス３：NP 困難，NP 完全)**.** 任意のクラス NP の問題から

第 5 章

計算と論理

計算と論理に関する諸概念は計算機科学の基盤となって居る．しかも，計算機科学に於ける長年の研究によって，両者の間の様々な関係が発見され，論理と計算はもはや一体化して居ると言っても過言ではない．

計算を言語的に実現する一連の手法,「演繹体系」と「ラムダ計算」とその間の関係を考える．ラムダ計算は，Turing 機械が計算者 (computor) の動きを模倣して作られた計算モデルであるのと同様に，現在授業で教えられる殆どの「手続き」もラムダ計算の variation として理解できる．その結果，propositions as types というパラダイムが見えてくる．自然演繹体系とラムダ計算の型システム，推論体系とコンビネーター論理の型システムである．

定義 5.0.1 (科学の，計算と数学)**.**

0. Homo sapiens の創造行為の内に，理論体系の創造という行為がある．そのうち，特に科学と呼ばれる営みは，形式体系と意味論的な知識体系との 2 本の世界樹からなることに特徴付けられる．前者のメディア側を中心とした学問を形式科学，後者の対象側を中心とした学問をその対象に依って自然・社会科学（経験科学）という．

1.（形式）論理とは，古くから数学に現れる演繹の過程を，記号に対する操作として定式化した形式体系である．形式体系を研究する学問を形式科学という．自然言語は形式の一例である.[†1]

2. 構文論と意味論が，形式体系と人間の精神活動との 2 本の世界樹であり，2 つの密接

[†1] jp.wikipedia.org より，論理学、数学、システム理論に加え、計算機科学、情報理論、ミクロ経済学、統計学、言語学などといった分野の理論ベースの細分野（たとえば計算機科学であれば理論計算機科学）がこれに含まれる。

な結び付きは健全性と完全性に端的に集約される．健全性とは証明可能な論理式が常に正しい $\rightarrow$ ことを主張し，完全性が正しい論理式が証明可能である $\leftarrow$ ことを主張する．

5.1 Lambda Calculus

計算を記号列の簡約として端的に実装する計算モデルであり，この数学模型の上に Lisp の世界観やコンパイラなどの計算機言語が乗る．物理的な計算機を，人間が一番操作しやすい実体としての計算に翻訳するシステムであるところの計算機言語の基本原理となって居る．取り沙汰されないだけで，数学者の行う計算は大体これであるくらいの一般的で純粋なモデルとなっており，実際 Curry-Howard 対応が成立するような綺麗な圏を作る．

ラムダ式とは，1 つの名前のない関数を指し示し，しばしば高階になる．これは一般再帰関数に一致し，Church-Turing thesis は，あらゆる計算可能な対象はラムダ式としての表現を持つことを主張する．

5.1.1 ラムダ計算

定義 5.1.1 (Lambda 式)**.** Lambda 式とは，x をある議論領域を走る変数，M をラムダ式についてのメタ変数として，次の BNF 記法による条件を満たすと分かるような式（形式言語）である．

$$M ::= x \mid \lambda x.M \mid M_1 M_2$$

注 5.1.2.

1. 「ラムダ式とは、変数または、$\lambda x.M$ という形の式（で M の部分はラムダ式)、または M_1, M_2 という形の式（で M_1, M_2 の部分はラムダ式）のいずれかである」と読む．
2. ラムダ式という対象言語を定義するために，BNF 記法という言語や自然言語を用いて定義した．後者をメタ言語という．従って，M と x は全く出自が違うことに注意．
3. OCaml では，λ は fun，ピリオド. は->となる．

例 5.1.3. 1. $\lambda y.yy$ はラムダ式である．y は変数でありラムダ式である．yy は自身 y に y を適応するラムダ式である．これに対して y についてのラムダ抽象が $\lambda y.yy$ である．

定義 5.1.4 (括弧の除去則)**.** 関数適用の場合の左結合と，ラムダ抽象の範囲の全域性とは暗黙の約束とする．

1(関数適用). $(xy)z = xyz$

2(ラムダ抽象). $\lambda x.(xy) = \lambda x.xy$

注 5.1.5. 前者は和や積を計算するにあたっての暗黙の規則に一致するが，これらは特に結合的な演算の例である．

定義 5.1.6 (計算規則)**.**

1(関数適用式に対する β-簡約). $\lambda x.M$ という形のラムダ式の N への作用 $(\lambda x.M)N$ は，ラムダ式 M 内の x の出現を，全てラムダ式 N に置換したもの $[N/x]M$ とする．但し，$[N/x]$ はラムダ式に対する置換演算子のようなものである．

2(ラムダ抽象式に対する η-変換). x の出現しないラムダ式 M について，$\lambda x.Mx \xrightarrow{\eta} M$ と簡約出来る．これは任意のラムダ式 N に対して適用した時の振る舞いが（β-簡約形が），$\lambda x.Mx$ と M とで等価だからである．この逆操作を η-展開 (expansion) という．

注 5.1.7. η 簡約は，「全ては関数」というラムダ計算の世界で重要な意味論的言い換えになっている．ラムダ式は，関数適用をしないと意味を持たず，関数適用（即ち β 簡約）に対する振る舞いがラムダ式を意味論的に特徴付ける．この関係（β 簡約形の一致）を β 等号といい，$\lambda x.Mx =_\beta M$ と表す．

定義 5.1.8 (代入の形式的定義)**.** 代入 $[N/x]M$ とは，次の操作である．

M が変数の時

1. $[N/x]x = N$
2. $[N/x]y = y$

M が関数適用式である時

3. $[N/x]M_1M_2 = ([N/x]M_1)([N/x]M_2)$

M がラムダ抽象式である時

4. $[N/x](\lambda x.M) = (\lambda x.M)$
5. $[N/x](\lambda y.M) = \lambda y.[N/x]M$（但し，y が N 内に自由出現しない時.）
6. $[N/x](\lambda y.M) = \lambda z.[N/x][z/y]M$（但し，y が N 内で自由出現する時，z を新たな変数として.）

注 5.1.9. 4 は，式 $\lambda x.M$ 内での，M で出現する x は，λ で束縛されてる x であり，M 内に出現する他の変数との区別の中でしか意味を持たない．これを束縛変数という．従って，自由出現である $[N/x]$ の x としては hit しない．

6 も同様であり，M 内に束縛出現する y と，N 内に自由出現する y とは違う文字であるから，この場合は束縛されている M に出現する方を別の文字に退避する．

この時の束縛変数の退避の部分は，β 等号で結ばれるようなラムダ式を導く．この構成を特に，α 変換といい，$\lambda x.M =_\alpha \lambda y.[y/x]M$ と表す．

例 5.1.10 (文字が衝突する際の処理が煩雑なので，衝突する前に α 変換を施しておくと楽だと気づいた．)**.** 1. $(\lambda x.\lambda y.x)yx$

$$\begin{aligned}(\lambda x.\lambda y.x)yx &\xrightarrow{\alpha} (\lambda x.\lambda z.x)yx\\ &\xrightarrow{\beta} (\lambda z.y)x\\ &\xrightarrow{\beta} y\end{aligned}$$

2. $(\lambda b.\lambda x.\lambda y.bxy)(\lambda x.\lambda y.y)xy$

$$\begin{aligned}(\lambda b.\lambda x.\lambda y.bxy)(\lambda x.\lambda y.y)xy &\xrightarrow{\beta} (\lambda b.\lambda x.\lambda y.bxy)(\lambda y.y)y\\ &\xrightarrow{\alpha} (\lambda b.\lambda x.\lambda y.bxy)(\lambda z.z)y\\ &\xrightarrow{\beta} (\lambda b.\lambda x.\lambda y.bxy)y\\ &\xrightarrow{\alpha} (\lambda b.\lambda x.\lambda z.bxz)y\\ &\xrightarrow{\beta} \lambda x.\lambda z.yxz\\ &\xrightarrow{\eta} \lambda x.yx\\ &\xrightarrow{\eta} y\end{aligned}$$

3. $(\lambda f.\lambda x.f(f(x)))(\lambda f.\lambda x.f(f(x)))$

$$\begin{aligned}(\lambda f.\lambda x.f(f(x)))(\lambda f.\lambda x.f(f(x))) &\xrightarrow{\alpha} (\lambda f.\lambda y.f(f(y)))(\lambda f.\lambda x.f(f(x)))\\ &\xrightarrow{\beta} \lambda y.(\lambda f.\lambda x.f(f(x)))((\lambda f.\lambda x.f(f(x)))(y))\\ &\xrightarrow{\beta} \lambda y.(\lambda f.\lambda x.f(f(x)))(\lambda x.y(yx))\\ &\xrightarrow{\alpha} \lambda y.(\lambda f.\lambda z.f(f(z)))(\lambda x.y(yx))\\ &\xrightarrow{\beta} \lambda y.(\lambda z.(\lambda x.y(yx))((\lambda x.y(yx))(z)))\\ &\xrightarrow{\beta} \lambda y.(\lambda z.(\lambda x.y(yx))(y(yz)))\\ &\xrightarrow{\beta} \lambda y.(\lambda z.(y(y(y(yz)))))\end{aligned}$$

これは実は 2^2 という自然数演算を表し，答えとして自然数の 4 を得ている計算を表す変換である．次の節で，ラムダ計算による数学的対象の表現を調べる．

5.1.2　Church-Rosser 性と評価戦略

まずラムダ項の簡約の定義は，様々な言葉でかける．文脈を使う方法，推論規則即ち帰納的な定義など．

定義 5.1.11 (Context)**.** λ 項の文脈とは，次により定義されるものである．

$$C[] ::= [] \mid (M\ C[]) \mid (C[]\ M) \mid \lambda x.C[]$$

記号 [] を穴 (hole) と呼び，λ 項の文脈のことを穴あき項ともいう．

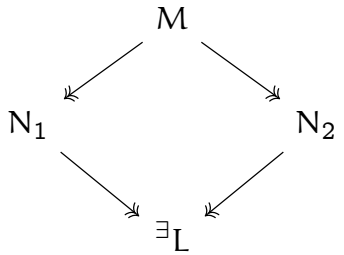
関係の定義

β 簡約が停止するようなラムダ項には一意的な表示（正規形）が存在する．

定義 5.1.12 (反射推移閉包：ラムダ項の射)**.** 二項関係 $\to^*$ または $\twoheadrightarrow$ は，簡約の反射推移閉包で，「左辺は 0step 以上の簡約によって右辺に変換する」，という関係を表す．

定義 5.1.13 (β 等価)**.** 二項関係 $=_\beta$ を，$\to^*$ を含む最小の同値関係とする．

定理 5.1.14 (Church-Rosser theorem)**.** 「これ以上 β 簡約できない形」を β 正規形と呼ぶ．β 正規形を持つラムダ項のその最終的な簡約形は，β 簡約を施す順序に依らない．



注 5.1.15. 1. 証明は基本的には，$M \to^* N_1 \not\to, M \to^* N_2 \not\to$ という関係が成り立った時，同型 $N_1 =_\beta N_2$ を構成できることによる．

なお，β 簡約が停止しないようなラムダ項などについては言及していない．

2. メタ的には，この定理は簡約によって定義される計算概念に，矛盾がないという物理的な安定性を示唆して居る．

［証明］．　□

簡約戦略

定理 5.1.16 (最左簡約)**.** β 簡約が停止するようなラムダ項 M について，その正規形は常に一番左の β 簡約基 (β-redex) から評価することで得られる．

注 5.1.17 (名前呼びと値呼びと必要呼び)**.** 最左簡約は理論的には頑健だが，Scheme など ML 系や C を初めとした一般の言語は効率の問題だろうかこれを採用していない．**値呼び Applicative order** (Programming 言語での用語) と言って，引数を先に計算する戦略の方が，重複した必要のない計算をしない．一方最左簡約は**名前呼び**と呼ばれ，これをそのまま採用しているプログラミング言語はなく，これを動的に選択した**必要呼び**はあり，例えば Haskell はこれを採用している．

5.2　Church encoding：ラムダ計算による数学的対象の表現

ラムダ計算は Turing 機械と同等の表現能力を持つ形式体系である．この一文がどういう意味を持つのかがよくわかっていない．いずれにしろ，Turing 機械を模倣することができる計算モデルである．このことを，まずは Church encoding と呼ばれる，ラムダ計算によるデータと操作の定義法から確認する．

定義 5.2.1 (formal system)**.** 公理と定理とその間の論理計算 (logical system) からなる体系のこと．これに解釈写像 F を与えて構造 M を考えるのが次のステップであろうか，いや，そこには進化的要因しかなくて，集合論数学と完全に相対的であろうか．

注 5.2.2. The term formalism is sometimes a rough synonym for formal system, but it also refers to a given style of notation, for example, Paul Dirac's bra - ket notation.[†2]

記法 5.2.3. 二項関係 $\to^*$ は，簡約の反射推移閉包で，「左辺は 0step 以上の簡約によって右辺に変換する」，という関係を表す．

5.2.1　種々のデータ構造の表現

集合と論理を実装する．

[†2] en.wikipedia.org での Fromal system の entry.

5.2.1.1　Bool 値，即ち命題論理

まずは論理の実装が肝要である．

定義 5.2.4 (Boolean values)**.**

$$\mathsf{T} := \lambda \mathrm{t}.\lambda \mathrm{f}.\mathrm{t},\ \mathsf{F} := \lambda \mathrm{t}.\lambda \mathrm{f}.\mathrm{f}$$

解釈 1. True とは，t を受け取ってそれの定値関数を返す関数，False とは，どんな t を受け取ろうとも，恒等関数を返す関数とする．

解釈 2. 組 (t, f) をもらってきて，第一引数 t を返すか第二引数 f を返すか．

これは明らかに，関数適応 MN の形のラムダ式を与えた時の挙動を意識して作られている．実際，if 文は次のように実装される．

定義 5.2.5 (if sentence)**.** if 文

```
if e_1 then e_2 else e_3
```

をラムダ式 $e_1e_2e_3$ と置く．

すると，

```
if T then e_2 else e_3
```

は $(\lambda \mathrm{t}.\lambda \mathrm{f}.\mathrm{t})e_2e_3 \xrightarrow{\beta} (\lambda \mathrm{f}.e_2)e_3 \xrightarrow{\beta} e_2$ を得る．同様に，$(\lambda \mathrm{t}.\lambda \mathrm{f}.\mathrm{f})e_2e_3 \xrightarrow{\beta} (\lambda \mathrm{f}.\mathrm{f})e_3 \xrightarrow{\beta} e_3$

定義 5.2.6 (Boolean operator)**.**

1. 一項演算 $\neg$ を，$\mathrm{not} = \lambda \mathrm{b}.\mathrm{bFT}$ とする．
2. 二項演算 $\wedge$ を，$\mathrm{and} = \lambda \mathrm{b}_1.\lambda \mathrm{b}_2.\mathrm{b}_1(\mathrm{b}_2\mathrm{TF})\mathrm{F}$ とする．

例 5.2.7 (Scheme)**.**

```
> (define mytrue (lambda (t) (lambda (f) t)))
> (define myfalse (lambda (t) (lambda (f) f)))
> (define myif (lambda (e1 e2 e3) ((e1 e2) e3)))
> (myif mytrue 1 2)
1
>(myif myfalse 1 2)
2
> (define mynot (lambda (b) ((b myfalse) mytrue)))
> (myif (mynot mytrue) 1 2)
```

```
10      2
11      > (define myand (lambda (b1 b2) ((b1 ((b2 mytrue) myfalse))
            myfalse)))
12      > (myif (myand mytrue myfalse) 1 2)
13      2
```

1. Scheme の構文として，組を引数として与えることができるので，何度も lambda を書くことを省略した．

2. if 文の定義では，$e_1e_2e_3$ を η 展開したものを使ったが，この時 $(e_1\ e_2)\ e_3$ という括弧が必要であった．本来ラムダ式としてはなくて良い．

5.2.1.2　組とその射

組という対象は特にあからさまに，普遍性への下心が丸見えに自然に構成できる．組という対象自身も，それを特徴付ける射も，いずれもラムダ式である．集合論の場合も同じことが成り立つ（全てが集合）のではあるが．

定義 5.2.8 (tuple)**.** 組 (M, N) とは，関数 f を受け取って，MN に作用させる関数とする．

$$(M, N) := \lambda f.fMN$$

これを定める射影は，$\mathrm{pr}_1(P) = P(\lambda x.\lambda y.x), \mathrm{pr}_2(P) = P(\lambda x.\lambda y.y)$ とする．

注 5.2.9. OCaml では $\mathrm{pr}_1, \mathrm{pr}_2$ は fst と snd になっている．

また，確かに

$$\begin{aligned}\mathrm{pr}_1(M, N) &= (\lambda f.fMN)(\lambda x.\lambda y.x)\\ &\xrightarrow{\beta} (\lambda x.\lambda y.x)MN\\ &\xrightarrow{\beta} (\lambda y.M)N\\ &\xrightarrow{\beta} M\end{aligned}$$

である．

例 5.2.10 (Scheme)**.**

```
1  > (define mkpair (lambda (M N) (lambda (f) (f M N))))
2  > (define p (mkpair 1 2))
3  > p
```

```
#<procedure #f>
> (define pr1 (lambda (p) (p (lambda (x y) x))))
> (define pr2 (lambda (p) (p (lambda (x y) y))))
> (pr1 p)
1
> (pr2 p)
2
```

5.2.1.3　Church numerals：自然数とその上の演算

Church encoding では何故か冪演算が一番簡単な表記になる．Church numeral n とは，後者関数（第一引数）の n 回適用であるから，ある z について，等式 $s^n z \xleftarrow{\eta} (\lambda z.s^n z)z \xleftarrow{\beta} (\lambda s.\lambda z.s^n z)sz = nsz$ が成り立つ．従って Church numerals を関数適用で繋げると「後者関数を m 回適用するという（後者）関数を n 回適用する」という意味論となり，確かに冪演算が一番簡単な表記になる（定理 5.2.20：関数適応は冪に対応する）．

> "The Church numeral 3 means simply to do anything three times. It is an ostensive demonstration of what is meant by "three times"."[†3]

定義 5.2.11 (Church numerals)**.** 0 とは関数に対して恒等関数を返す関数で（False と等価！），$n \leqslant 1$ とは，関数に対して，「それを n 回適用するという関数」を返す関数のことである．

$$
\begin{aligned}
0 &= \lambda s.\lambda z.z \\
1 &= \lambda s.\lambda z.sz \\
2 &= \lambda s.\lambda z.ssz \\
&\vdots = \quad \vdots \\
n &= \lambda s.\lambda z.s(s\cdots(sz)\cdots) =: \lambda s.\lambda z.s^n z
\end{aligned}
$$

定義 5.2.12 (後者関数)**.** $n \in \mathbb{N}$ を表すラムダ式を $n+1$ を表すラムダ式に返すような関数であるから，$\mathrm{Succ} := \lambda n.(\lambda s.\lambda z.s(s^n z))$ としたいが，これは厳密なラムダ式ではない．$s^n z \xleftarrow{\eta} (\lambda z.s^n z)z \xleftarrow{\beta} (\lambda s.\lambda z.s^n z)sz = nsz$ より，

$$\mathrm{Succ} := \lambda n.(\lambda s.\lambda z.s(nsz))$$

[†3] en.wikipedia.org の Church encoding から．

と定める.

注 5.2.13. いま，$ns =_\beta \lambda z.s^n z$ であり，右辺は Church numeral ではない.

定義 5.2.14 (加法)**.**

$$\begin{aligned}\mathrm{Plus} &= \lambda m.\lambda n.\lambda s.\lambda z.s^{n+m}z \\ &= \lambda m.\lambda n.\lambda s.\lambda z.s^m(s^n z) \quad \text{or} \quad \lambda m.\lambda n.\lambda s.\lambda z.s^n(s^m z) \\ &= \lambda m.\lambda n.\lambda s.\lambda z.s^m(nsz) \quad \text{or} \quad \lambda m.\lambda n.\lambda s.\lambda z.s^n(msz) \\ &= \lambda m.\lambda n.\lambda s.\lambda z.ms(nsz) \quad \text{or} \quad \lambda m.\lambda n.\lambda s.\lambda z.ns(msz)\end{aligned}$$

命題 5.2.15. $\mathrm{Succ} =_\beta (\mathrm{Plus}\ 1)$

［証明］.

$$\begin{aligned}\mathrm{Plus}\ 1 &= \lambda m.\lambda n.\lambda s.\lambda z.(\lambda s'.\lambda z'.s'z')s(nsz) \\ &= \lambda m.\lambda n.\lambda s.\lambda z.(\lambda z'.sz')(nsz) \\ &= \lambda m.\lambda n.\lambda s.\lambda z.s(nsz) = \mathrm{Succ}\end{aligned}$$

□

注 5.2.16 (謎)**.** また一方で，加法演算 $n + m$ とは，「n に対して後者関数 Succ を m 回適用したもの」と捉えなおせる．これによる表現 $\lambda m.\mathrm{Succ}^n\ m$ も extentional に等価なラムダ式であるはずだが，β 変換で一致することが示せない．乗法のようには行かないのだ.

定義 5.2.17 (乗法)**.** 加法の際と同様，n と m は可換である.

$$\begin{aligned}\mathrm{Mult} &= \lambda m.\lambda n.(\mathrm{Plus}\ m)^n 0 \\ &= \lambda m.\lambda n.n(\mathrm{Plus}\ m)0 \\ &= \lambda m.\lambda n.n(\lambda n.\lambda s.\lambda z.ms(nsz))\lambda s.\lambda z.z\end{aligned}$$

命題 5.2.18 (乗法の別表現)**.**

$$\mathrm{Mult} = \lambda m.\lambda n.\lambda s.n(ms)$$

定義 5.2.19 (冪乗)**.** 加法の際と同様，n と m は可換である.

$$\begin{aligned}\mathrm{Exp} &= \lambda m.\lambda n.n(\mathrm{Mult}\ m)1 \\ &= \lambda m.\lambda n.n(\lambda n.n(\lambda n.\lambda s.\lambda z.ms(nsz))\lambda s.\lambda z.z)\lambda s.\lambda z.sz\end{aligned}$$

定理 5.2.20 (冪乗の別表現：関数適応は冪に対応する)**.**

$$\mathrm{Exp} = \lambda m.\lambda n.nm$$

ついに他の引数 s,z は全て消え去った．ではこの上の演算はどう表現するのか？

命題 5.2.21. Church encoding の世界では，$0^0 = 1$ が成り立つ.

［証明］**.** 部分的に z にのみ η 展開を施す操作に注意して

$$\begin{aligned} 0^0 &= \mathrm{Exp}\ 0\ 0 \\ &= (\lambda s.\lambda z.z)(\lambda s.\lambda z.z) \\ &\xrightarrow{\beta} \lambda z.z \\ &\xrightarrow{\eta} \lambda z.\lambda s.zs \end{aligned}$$

を得る．あとは，s, z の記号を入れ替えれば良い. □

例 **5.2.22** (Scheme)**.**

```
1  > (define zero ((lambda (s) (lambda (z) z))))
2  ERROR on line 1: not enough args
3      #<procedure #f>
4      0
5  > (define zero (lambda (s) (lambda (z) z)))
6  > (define one (lambda (s) (lambda (z) (s z))))
7  > (define two (lambda (s) (lambda (z) ((s s) z))))
8  > (define three (lambda (s) (lambda (z) (((s s) s) z))))
9  > (define four (lambda (s) (lambda (z) ((((s s) s) s) z))))
10 > (define five (lambda (s) (lambda (z) (((((s s) s) s) s) z))))
11 > (define toint (lambda (n) ((n (lambda (s) (+ s 1))) 0)))
12 > (toint five)
13 ERROR on line 8: invalid type, expected Number: #<procedure #f>
14 > (toint one)
15 1
16 > (define two (lambda (s) (lambda (z) (s (s z)))))
17 > (define three (lambda (s) (lambda (z) (s (s (s z))))))
18 > (define four (lambda (s) (lambda (z) (s (s (s (s z)))))))
19 > (define five (lambda (s) (lambda (z) (s (s (s (s (s z))))))))
20 > (toint five)
21 5
```

```
> (define add (lambda (m) (lambda (n) (lambda (s) (lambda (z) ((m s)
    ((n s) z)))))))
> (add five five)
ERROR in final-resumer: too many args
    #<procedure add>
    2
> ((add five) five)
#<procedure #f>
> (toint ((add five) five))
10
> (toint ((add three) four))
7
> (define mult (lambda (m) (lambda (n) (lambda (s) (n (m s))))))
> (toint ((mult five) five))
25
> (define exp (lambda (m) (lambda (n) (n m))))
> (toint ((exp three) three))
27
> (toint ((exp five) five))
3125
> (toint ((exp four) four))
256
```

1. 関数 toint が 1 つの解釈関手となって居る．s は successor function，z は 0 である．

最初ものすごく美しい系列が Succ, Plus, Mult, Exp にあるかと思われたが，そこまででもなかった．
以上，後者関数 Succ から始まり，Plus n m は Succ n を m 回適用したもの，Mult n m は Plus m

定義 5.2.23 (0 判定真理値関数)**.**

$$\mathrm{isZero} := \lambda n.n(\lambda b.F)T$$

これは，Church encoding された自然数 n を，後者関数を λb.F，0 を T として解釈せよ，という関数である．これは後者関数を一回でも 0 に対して適応するような数（非零数）に対して False を返す関数となって居る．

定義 5.2.24 (前者関数)**.**

$$\mathrm{Pred}(n) = \begin{cases} 0 & (n = 0) \\ n - 1 & (\text{otherwise}) \end{cases}$$

は，$\mathrm{Next}(x, y) = (x + 1, x)$ という，情報保管用の引数を備えた実質上一変数の関数を用いて，n を再現し，第一引数が n に到達したときの第二引数を参照することで，次のように定める．

$$\begin{aligned} \mathrm{Pred} &= \lambda n.\mathrm{pr}_2(\mathrm{Next}^n(0,0)) \\ &= \lambda n.\mathrm{pr}_2(n(\lambda x.\lambda y.(x+1, x))(0,0)) \end{aligned}$$

定義 5.2.25 (減算)**.**

$$\mathrm{Minus} = \lambda m.\lambda n.n\mathrm{Pred}m$$

5.2.2　不動点演算子を用いた再帰関数の定義

非常に鮮やかな Lambda calculus 特有の技法で，（原始的な）再帰の概念が実装される．これで Turing 完全性が確認される．

命題 5.2.26. ラムダ式 F に対して，式 $M_F := (\lambda x.F(xx))(\lambda x.F(xx))$ と置くと，これは F の不動点である．即ち，次が成り立つ．

$$M_F F =_\beta M_F$$

［証明］**.**

$$\begin{aligned} M_F &= (\lambda x.F(xx))(\lambda x.F(xx)) \\ &\xrightarrow{\beta} F((\lambda x.F(xx))(\lambda x.F(xx))) \\ &= F(M_F) \end{aligned}$$

より，$M_F \to^* F(M_F)$ □

定義 5.2.27 (fixed point combinator)**.** 任意のラムダ式に対して，その不動点を返す関数 $Y := \lambda f.M_f = \lambda f.((\lambda x.f(xx))(\lambda x.f(xx)))$ を，不動点演算子という．

例 5.2.28 (再帰関数の表現：階乗)**.** まず，再帰関数の定義式を，等式の形で得る．

$$f = \lambda n.(\mathrm{if}\ n = 0\ \mathrm{then}\ 1\ \mathrm{else}\ n * f(n-1))$$

この f を何度も適用して，変化しなくなった安定点が解である．ここで，この等式を満たす関数 f に対して，次の関数 factgen を定義すると，この不動点とは，上の等式を満たす関数に他ならない．

$$\mathrm{factgen} = \lambda f.\lambda n.(\mathrm{if}\ n = 0\ \mathrm{then}\ 1\ \mathrm{else}\ n * f(n-1))$$

これを用いて，階乗を計算する関数 fact は，$\mathrm{fact} = Y\ \mathrm{factgen}$ を得る．実際，$F = \mathrm{factgen}$ とすると，

$$\begin{aligned}\mathrm{fact}\ 3 &= YF3\\ &\to^* F(YF)3 = \mathrm{factgen}(\mathrm{fact})3\\ &\to^* \mathrm{if}\ 3 = 0\ \mathrm{then}\ 1\ \mathrm{else}\ 3 * f(3-1)\\ &\to^* 3 * (YF2)\end{aligned}$$

例 **5.2.29** (Scheme).

```
1   > (define Y (lambda (f) ((lambda (x) (f (x x))) (lambda (x) (
        f (x x))))))
2   > (define factgen (lambda (f) (lambda (n) (if (= 0 n) 1 (* n
        (f (- n 1)))))))
3   > (define fact (Y factgen))
4   ERROR: out of stack space
5   > (define Y (lambda (f) ((lambda (x) (f (lambda (z) ((x x) z)
        ))) (lambda (x) (f (lambda (z) ((x x) z))))))))
6   > (define fact (Y factgen))
7   > (fact 3)
8   6
9   > (fact 10)
10  3628800
```

1. Scheme では引数を先に評価しようとするので，$Y = \lambda f.((\lambda x.f(xx))(\lambda x.f(xx)))$ の xx 部を η 展開しておかないと，延々と評価が続いていずれは stack overflow が起こる．η 展開するとそこがラムダ抽象されるために，引数が与えられるまで評価されない．

命題 5.2.30. 等式 $fx = e$ を満たす再帰関数 f は $f = Y(\lambda f.\lambda x.e)$ である．

［**証明**］. 等式 $fx = e$ を満たす再帰関数 f は $f = \lambda x.e$ となる（e には x が自由出現して居るかも知れない）．この時，$f = \lambda x.e \xrightarrow{\eta} (\lambda f.\lambda x.e)f$ より，f は関数 $\lambda f.\lambda x.e$ の不動点である． □

5.3 Simply typed lambda calculus

> 型の概念に大きな役割を誰にも見える形で付与したのは型付ラムダ計算の影響が大きかっただろう．ラムダ計算は初め数理論理学の世界で考え出されたものであるが，その初期の段階から，型は重要な地位を与えられていた．ラムダ計算はやがて，計算機科学の中で関数型プログラミングを基礎付ける道具として用いられる様になり，それに遅れて型の概念も計算機科学の中に取り入れられる様になった．（長谷川立）

こうして，ラムダ計算と型理論という 2 つの形式体系が交わり合う体系が得られる．しかし，もはや Y 結合子には型が付かない為，明示的に primitive として再帰を加えない限り，Turing 機械と同等の表現力は必ずしも持たなくなる．

定義 5.3.1 (明示的に型付けされた単純型付きラムダ計算)**.**

$$\begin{aligned}\tau &::= b \mid \tau_1 \to \tau_2\\ b &::= \mathrm{int} \mid \mathrm{bool} \mid \cdots\\ M &::= c^{b_1\to\cdots\to b_k\to b} \mid x \mid \lambda x:\tau.M \mid M_1M_2\end{aligned}$$

注 5.3.2. 1. 型 τ は, 基本型か, その間の関数型かのいずれかである（直積型, 直和型も含める）．ただし，この記法は右結合とする．$\tau_1 \to (\tau_2 \to \tau_3) = \tau_1 \to \tau_2 \to \tau_3$.
2. ラムダ項は，従来のものに変数 x について型注釈を付ける記法を採用し，また新たに定数も primitive にラムダ項として認める．定数にはその右肩に型明示する記法を採用する．これを明示的型付け (explicit typing) という．
3. 定数とは，自然数やその間の演算など，arity 0 の関係記号や関数記号のことを指す．定数の β 簡約は，翻訳関手による規則に従う．$c^{b_1\to\cdots\to b_k\to b}c^{b_1}\dots c^{b_k} = [c](c_{b_1},\cdots,c_{b_k}) : b$ ただし $[c]$ とは翻訳関手による値 $F(c^{b_1\to\cdots\to b_k\to b})$ とした．何かしらの arity k の数学的対象である．

5.3.1 型の理論

定義 5.3.3 (Type judgement)**.** 三項関係 $\vdash$ を, 3-組 (Γ, M, τ) が, 型環境 Γ においてラムダ式 M に型 τ が
という関係（この文の意味はあくまで推論規則によって構文論的／構成的に定義される）
と定める．これを $\Gamma \vdash M : \tau$ と表す．

ただし，型環境 (type environment) ／型割当 (type assignment) γ とは，変数記号（の部分集合）から型への写像 $(x_1 : \tau_1, \cdots, x_n : \tau_n)$ である．Mapping 型というか Hash table というか．

定義 5.3.4 (Well-typed)**.** 型判断の三項関係 $\Gamma \vdash M : \tau$ は，次の帰納的定義に従う．この時，ラムダ項 $M : \tau$ は正しく片付けされて居る (well-typed) といい，これを $\Gamma \vdash M : \tau$ と表す．

1(定数の型). $\Gamma \vdash c^{\tau} : \tau$

2(ImpVar:型環境が変数に定める型). $\dfrac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$

3(ImpLam:ラムダ抽象). $\dfrac{\gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . M : \tau_1 \to \tau_2}$

4(ImpApp:関数適用). $\dfrac{\Gamma \vdash M : \tau_1 \to \tau_2, \Gamma \vdash N : \tau_1}{\Gamma \vdash MN : \tau_2}$

注 5.3.5. このように，型判断を導くのに証明木が用いられ，これを**型付け導出木 (typing derivation tree)** と呼び，各規則を型付け規則 (typing rule) という．

命題 5.3.6 (型付けの一意性)**.** 正しく型付けされた項 M の型は一意的に定まる．

注 5.3.7. 本質的には，各変数について型を一通りに定めたことによる．そうではない別の体系を作れば，これは成り立たなくなる．

定義 5.3.8 (weak normalizability)**.** 正しく型のついた M が弱い意味で正規化可能であるとは，β 正規形に至る β 簡約列が存在することをいう．

注 5.3.9. 即ち，有限圏からの n-図式を持つ，と．従って，簡約戦略に依っては β 正規形に到達しない道があることも許して居る．

定理 5.3.10 (強い正規化定理 strong normalizability)**.** 正しく型のついた M は，強い意味で正規化可能である．即ち．M は無限 β 簡約列を持たない．

注 5.3.11. 即ち，ω-図式を持たない．従って，簡約戦略に依らずに，正しく型がついていれば簡約は停止する．逆に言えば，簡約の停止しない Y 結合子には型がつかない．

定理 5.3.12 (主部簡約定理／型保存 subject reduction theorem)**.** $\Gamma \vdash M : \tau$ かつ $M \to_\beta N$ ならば，$\Gamma \vdash N : \tau$ である．

注 5.3.13. β 簡約に対して型が保存するように，compatible に体系が作れたことがわか

る．射である．圏である．実際，これは殆どの型付き計算モデル，あるいは言語で目指される性質である．

系 5.3.14. $\Gamma \vdash M : \tau$ かつ $M \to^* C[(\lambda x : \sigma.L)N]$ ならば，$\Gamma \vdash N : \sigma$ である．

注 5.3.15. 「すなわち，関数が要求する引数の型と実際の引数の型は一致する」とあるが，よくわからない．

5.3.2　型検査

問 (Type checking). (Γ, M) を入力として，M が型を持つか（$\Gamma \vdash M : \tau$）を判定し，成り立つならばその型 τ を返せ．

には，算譜が存在して，次の算譜 TC が実際のコンパイラで使われて居る．

算譜. 算譜 $TC : (\Gamma, M) \nrightarrow \tau$ を，型付け導出木に従って次のように定める．

1(定数). $TC(\Gamma, c^\tau) = \tau$ とする．

2(Var). $TC(\Gamma, x)$ は if x in dom(Γ) then $\Gamma(x)$ else fail とする．

3(Lam). $TC(\Gamma, \lambda x : \tau_1.M) = \tau_1 \to TC(\Gamma \cup \{x : \tau_1\}, M)$ と再帰的に定める．

4(App). $TC(\Gamma, MN)$ は，let $\tau_0 = TC(\Gamma, M)$ and let $\tau_1 = TC(\Gamma, N)$ if $\exists \tau_2\ \tau_0 = \tau_1 \to \tau_2$ then τ_2 else fail と定める． □

5.3.3　型推論

問 (Type checking). M を入力として，M が型を持ち得るかを判定し，成り立つならばその環境と型の組 (Γ, τ) を返せ．

変数や式に型変数を割り当て，型付け規則をその間の方程式に落とし込み，これを解く．

記法 5.3.16 (型変数)**.** α を型変数と呼び，全ての型を走るメタ変数とする．

算譜. 1.　与えられた M とその自由変数 $x_1, \cdots, x_n$ に対して，制約抽出手続き $Extract(\{x_1 : \alpha_1, \cdots, x_n : \alpha_n\}, M, \alpha)$ を施して，等式集合 $\mathcal{E}$ を得る．

2. $\mathcal{E}$ の単一化 σ に成功した場合，この時の $(\{x_1 : \alpha_1, \cdots, x_n : \alpha_n\}, \sigma)$ を返し，単一化に失敗したならば False を返す． □

5.3.4　型の構成

定義 5.3.17 (直積型 product type)**.**

$$\begin{aligned}\tau &::= b \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2\\ b &::= \mathrm{int} \mid \mathrm{bool} \mid \cdots\\ M &::= c^{b_1\to\cdots\to b_k\to b} \mid x \mid \lambda x:\tau.M \mid M_1M_2 \mid (M_1, M_2) \mid \mathrm{pr}_1(M) \mid \mathrm{pr}_2(M)\end{aligned}$$

定義 5.3.18 (直和型 sum type)**.**

$$\begin{aligned}\tau &::= b \mid \tau_1 \to \tau_2 \mid \tau_1 + \tau_2\\ b &::= \mathrm{int} \mid \mathrm{bool} \mid \cdots\\ M &::= c^{b_1\to\cdots\to b_k\to b} \mid x \mid \lambda x:\tau.M \mid M_1M_2 \mid\\ &\qquad \mathrm{inl}(M) \mid \mathrm{inr}(M) \mid (\text{case } M_0 \text{ of } \mathrm{inl}(x) => M_1 \mid \mathrm{inr}(y) => M_2)\end{aligned}$$

ただし，case 項や inl, inr 項は次のように positive type として構成的に定められる．なお，case 項は次のように match 項で表現されることもある．最後の表現はここだけのものである．

$$\begin{aligned}(\text{case } M_0 \text{ of } \mathrm{inl}(x) => M_1 \mid \mathrm{inr}(x) => M_2) &= (\mathrm{match}(M_0, x.M_1, x.M_2))\\ &= \lambda M_0:\tau_x+\tau_y.\{(\lambda x:\tau_x.M_1)+(\lambda y:\tau_y.M_2)\}M_0\end{aligned}$$

1. $\dfrac{\Gamma \vdash M:\tau_1}{\Gamma \vdash \mathrm{inl}(M):\tau_1+\tau_2}$
2. $\dfrac{\Gamma \vdash M:\tau_2}{\Gamma \vdash \mathrm{inr}(M):\tau_1+\tau_2}$
3. $\dfrac{\Gamma \vdash L:\tau_1+\tau_2,\ \Gamma, x:\tau_1 \vdash M:\tau,\ \Gamma, y:\tau_2 \vdash N:\tau}{\Gamma \vdash \text{case } L \text{ of } \mathrm{inl}(x) => M \mid \mathrm{inr}(y) => N:\tau}$ または $\dfrac{p:A+B,\ x:A \vdash c_A:C,\ y:B \vdash c_B:C}{\mathrm{match}(p, x.c_A, y.c_B):C}$

また，case 項の簡約規則は次の通りである．

1. $\text{case } \mathrm{inl}(M) \text{ of } \mathrm{inl}(x) => M_1 \mid \mathrm{inr}(y) => M_2 \to_\beta [M_1/x]M = (\lambda x.M_1)M$ または $\mathrm{match}(\mathrm{inl}(a), x.c_A, y.c_B) \to_\beta c_A[a/x]$
2. $\text{case } \mathrm{inr}(M) \text{ of } \mathrm{inl}(x) => M_1 \mid \mathrm{inr}(y) => M_2 \to_\beta [M_2/y]M = (\lambda x.M_2)M$ または $\mathrm{match}(\mathrm{inr}(b), x.c_A, y.c_B) \to_\beta c_B[b/y]$

注 5.3.19 (どうしてこんなに記法がわかりにくいんだ．)**.** $\mathrm{inl}(M:\tau_1)$ は，M が左に入った $\tau_1+\tau_2$ の項の一部としての M ということである．このように，直和型の構成は，既存の型 A に対してタグ inl を附す形で構成される．従って，直和型を構成する時，τ_2 の方に自由度があるために，型付けは明らかに一意的ではない．

case, match 文はある意味依存型の実装であり，変数 x, y の型に依存して振る舞いを変えることで，統合した型の上での関数適用を実装して居る．これは 2 つの関数 $A \to C, B \to C$ から和写像 $A + B \to C$ を構成して居るのに他ならない．この和写像の簡約則 case inl(M) of inl(x) => M_1 | inr(y) => M_2 とは，関数適用 $(\lambda x.M_1 + \lambda y.M_2)\,M$ で，M が x の型に族して居る場合は $(\lambda x.M_1)M$ とするということを表して居る．

データとしては, float は int と他との直和と考えたい．混ぜて使いたい．これを variant という．ん？ OCaml コードを見る限り，BNF 記法自体が型の直和なのかもしれない．だから | を混用するのか.

例 **5.3.20.** The positive presentation of sum types can be regarded as a particular sort of inductive type. In Coq syntax:

program 5.3.1　Coq

```
1        Inductive sum (A B:Type) : Type :=
2        | inl : A -> sum A B
3        | inr : B -> sum A B.
```

5.3.5　Curry-Howard 同型

関手 $NJ \to \lambda - Types$ が存在し，NJ に適切な同値関係を入れると，その商の圏については，この関手が可逆になる．

計算の世界	論理の世界
プログラム	証明
計算（簡約）	計算の簡単化
型検査	証明の整合性検査

正規化定理 5.3.10 とサブジェクト・リダクション定理 5.3.12 より，「型 A により型付けされたラムダ項が存在するならば，型 A により型付けられた正規形であるラムダ項が存在する」と言える．正規形は「カットを用いない証明」に対応するので，これの圏同型による翻訳は「全ての証明に対して，それと同じ結論を持つカットを用いない証明が存在する」（カット消去定理）となる．

例 **5.3.21** (証明の表現)**.**

1. 命題 $A \wedge B$ の証明としては，型 A, B から，それぞれの evidence $a : A, b : B$ を用いて，$(a, b) : A \times B$ という witness の構成をすれば良い．また，$A \wedge B$ から別の問題を示そうとしたときは，A, B いずれの主張も用いて良い．これは，直積型 $A \times B$ の型導出規則に対応する．

2. 命題 $A \to B$ の証明としては，自由変項として型 A の変数 (witness) を含むかもしれない (may) 型 B の元 (witness) を表す記号列を構成すれば良い．この命題 $A \to B$ から別の命題を示すときは，型 A の witness を取ってきて，それに関数適用をすれば良い．

3. 命題 $A \to (B \to (A \wedge B))$ の証明は，記号列 $\lambda x : A \lambda y : B.(x, y)$ が well-typed であることが証明になって居る．

4. 空型 (empty type) 0 とは，項を持たない型である．全く思考が進まないので categorical semantics で考えると，これは始対象であるから，0 からは全ての命題が導かれる．一方で 0 への射の存在は非自明である．命題 $\neg A$ の証明は，型 $A \to 0$ を持った項を構成することである．

5. 二重否定除去則とは次のような命題を指す $\neg\neg A \to A$．一般に型 $(A \to 0) \to 0$ と型 A は異なる．ただし，$A \to \neg\neg A$ は成り立つ．証明は $\lambda x : A.\lambda y : \neg A.yx$ である．

6. 排中律 $A \vee \neg A$ の証明は存在しない（従って，これかつ不条理則と同値な二重否定除去則も示せない)．これが直観的にあまりにわからなくて，一晩示そうとしてしまった．出来ないとは，出来ないと分からないことだ．

注 5.3.22. HoTT book で学んだ witness という表現が魔力的な力を発揮して居る．なんだか命題／型 A, B から元を取って居るように思えるが，全く別の概念な気もする．

5.4　その他の型付きラムダ計算の構成

ラムダ計算に対して，更なる型の構成を見る．これは Curry-Howard 対応の見地から見れば，むしろ非常に自然な構成であり，高階へ向かっていく世界樹の 1 本である．単純型付きラムダ計算と直観主義命題論理との対応から，依存型は一階の量化を，多相型は二階の量化を追加する構成に等しい．

5.4.1　多相型 / System F

Jean-Yves Girard (47-) により提唱された．

> In computer science, polymorphism refers to situations either where the same

> name is used to refer to more than one function, or where the same function is used at more than one type. One usually distinguishes these two kinds of polymorphism as ad hoc polymorphism and parametric polymorphism, respectively.[†4]

例 5.4.1. 恒等関数 $\lambda x.x$ は $\alpha \to \alpha$ の他に $(\alpha \to \alpha) \to (\alpha \to \alpha)$ という高階関数としての型も持ち得るので，それによる解釈で $(\lambda x.x)(\lambda x.x)$ は単純型付き λ 計算でも型 $\alpha \to \alpha$ を持つ．

一方，$(\lambda f.ff)(\lambda x.x)$ は $(\lambda x.x)(\lambda x.x)$ に β 簡約されるにも関わらず，引数 f に同じ型をつける必要があるため，上記の手法が使えず，整合的な型を付け得ない．

定義 5.4.2 (System F / polymorphic λ-calculus / second-order typed λ-calculus)**.** 型変数 α も関数の引数としてはラムダ項内に採用する．

$$\begin{aligned} \tau &::= b \mid \alpha \mid \tau_1 \to \tau_2 \mid \forall \alpha.\tau \\ M &::= c^{b_1 \to \cdots \to b_k \to b} \mid x \mid \lambda x : \tau.M \mid M_1 M_2 \\ &\quad \Lambda \alpha.M \mid M[\tau] \end{aligned}$$

型付け規則には次を追加する．

1. $\dfrac{\Gamma \vdash M : A}{\Lambda \alpha.M : \forall \alpha.A}$ である．ただし，型変数 α は M の各自由変数内で自由出現しないものとした．
2. $\dfrac{\Gamma \vdash M : \forall \alpha.A}{\Gamma \vdash MB : A[\alpha := B]}$

簡約規則には次の Λ 簡約を追加し，関係 $\to_{\beta\Lambda} := \to_\beta \cup \to_\Lambda$ を定め，この反射推移閉包を $\to^*_{\beta\Lambda}$ とする．

1. $(\Lambda \alpha.M)A \to_\Lambda M[\alpha := A]$

注 5.4.3. 1. $\Lambda \alpha.M$ は型変数 α についての抽象化であり，型抽象という．$M[\tau]$ は型抽象されたラムダ項 M への，型 τ の適用を表す．これを型適用という．

2. 直積型や直和型は多相型の $\to$ と $\forall$ で表現できるので（一階述語論理の要領？），これは一般には明示的には含めない．

3. 型付け規則の 1 の注意は，sequent 計算の $\forall$ 右の規則 $\dfrac{\Gamma \to \Delta,\ A[a]}{\Gamma \to \Delta,\ \forall x.A[x]}$ に付随する条件と対応する．M に自由出現する項 $\Lambda \alpha.x^\alpha$ も縛ってしまい，型 $\forall \alpha.\alpha$ を持つことになってしまう．

[†4] nLab "polymorphism"

例 5.4.4. 恒等関数 $\lambda x.x$ には，多相型 $\forall\alpha.\alpha \to \alpha$ が与えられる．これは $\alpha \to \alpha$ や $(\alpha \to \alpha) \to (\alpha \to \alpha)$ を表現して居る．

命題 5.4.5 (型推論)**.** 型推論は決定不能になる．

定理 5.4.6 (Subject reduction theorem)**.** 項 M が型 A を持つとする．$M \to M'$ ならば M' も型 A を持つ．

定理 5.4.7 (強い正規化定理 strong normalizability)**.** 項 M が型 A を持つとする．この時，M は，強い意味で正規化可能である．即ち．M は無限 β 簡約列を持たない．

5.4.2　多相型によるデータ型の表現

実は単純型付ラムダ計算では，Church numerals はうまくいかない．この System F で初めて全てに型がつく．

例 5.4.8. 1. チャーチ数は $(A \to A) \to A \to A$ という型を持つので，$nat = \forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha$ という多相型によりまとめられる．従って改めて自然数 n は $\lceil n \rceil = \Lambda\alpha.\lambda s : \alpha \to \alpha.\lambda z : \alpha.s^n z$.

2. $plus = \lambda m : nat.\lambda n : nat.\Lambda\alpha.\lambda s : \alpha \to \alpha.\lambda z : \alpha.m[\alpha]s(n[\alpha]sz)$
3. $mult = \lambda m : nat.\lambda n : nat.\Lambda\alpha.\lambda s : \alpha \to \alpha.m[\alpha](n[\alpha]s)$
4. $exp = \lambda m : nat.\lambda n : nat.\Lambda\alpha.n[\alpha \to \alpha](m[\alpha])$

5.4.3　再帰型

不動点演算子 Y に整合的に型を付けることで，Turing 機械と同等の表現力を回復することを考える．

定義 5.4.9 (再帰型)**.** 型変数も関数の引数としてはラムダ項内に採用する．

$$\tau ::= b \mid \alpha \mid \tau_1 \to \tau_2 \mid \mu\alpha.\tau$$

5.4.4　部分型 subtyping

関係 $\sigma < \tau$ を「型 σ の値は型 τ の値としても使える」と定める．

例 **5.4.10.** $\mathrm{int} < \mathrm{real}, \mathrm{real} \to \mathrm{int} < \mathrm{int} \to \mathrm{real}$

5.4.5　依存型

型が引数に依存できるようにする．別の型で別の型を添字づけて居るだけに見える．

> 直観的には，ラムダ項がデータやプログラムを表現し，型はデータやプログラムの集合を表現して居る．数学では，データをパラメータとして持つような集合がある．例えば，n 次元実ベクトル空間 $\mathbb{R}^n$ は自然数 n をパラメータとしてもつ集合である．
>
> 型付きラムダ計算においても，このように項をパラメータとする型を考えることができる．[7]

例 **5.4.11** (双対な 2 つの依存型：記号と意味が逆転して居るのが気になる)**.** 1(Π-type, dependent function type). 依存型 $\Pi n : \mathrm{Nat}.A^n$ を持つ項 M を，$\overline{3} : \mathrm{Nat}$ に適用した項は $M\overline{3} : A^3$ となる．A^n とは，n 個の直積を表す型である．この依存型は $\Pi n : \mathrm{Nat}.A$ の時，関数型 $\mathrm{Nat} \to A$ に等しくなる．

2(Σ-type, dependent product type). 「ある n が存在していて，A^n という型」を表す依存型 $\Sigma n : \mathrm{Nat}.A$ を持つ項 M は，第一成分として Nat の項と，それに依存して定まる第二成分として型 A^n の項とを得ることができる．それぞれを $\pi_1(M), \pi_2(M)$ と書く．この依存型は $\Sigma n : \mathrm{Nat}.A$ の時，直積型 $\mathrm{Nat} \times A$ に等しくなる．

5.4.6　多相型・依存型に於ける Curry-Howard 対応

多相型は二階の型付きラムダ計算 (second-order typed λ-calculus) ともいうのであった．これは，型変数 α が二階の変数に対応し，$\forall\alpha$ が二階の変数に対する量化 $\forall X$ に対応するからである．ここでも直観主義の二階命題論理に対応する．

型／命題 $\forall\alpha.A$ の型抽象／証明 $\Lambda\alpha.M$ から，任意の型 B に対して，$A[x := B]$ という命題の証明を得ることができる．なんだか 2-圏を感じる．

依存型の $\Pi x : A.P(x)$ は，項の変数による型の中傷を可能にするものということであった．命題を項について中傷するとは，一階の変数 x に対する量化 $\forall x$ に対応する．

5.5　証明支援システム

定理 5.5.1 (証明可能性：直観主義命題論理)**.** 次の 2 条件は同値である.

1. 型 A の閉じた単純型付きラムダ項が存在する.
2. A が直観主義命題論理で証明可能である.

系 5.5.2. 二重否定除去則 (double negation) $\neg\neg\alpha \to \alpha$ は直観主義命題論理では証明できない.

注 5.5.3. 逆は成り立つ．証明は $\lambda x : A.\lambda y : \neg A.yx$ である.

5.6　型理論

5.6.1　Propositions as types

命題と型は対応するので，その型を持った元 (evidence, witness) を構成することが証明に等しい.

> In gegenral, however, we will not construct witnesses explicitly; instead we present the proofs in ordinary mathematical prose, in such a way that they could be translated into an element of a type.

全くわからない段落があるが，なるほど，だからこそ Curry-Howard 同型とは言わないのか.

> However, the type-theoretic perspective on proofs is nevertheless dif- ferent in important ways. The basic principle of the logic of type theory is that a proposition is not merely true or false, but rather can be seen as the collection of all possible witnesses of its truth. Under this conception, proofs are not just the means by which mathematics is communicated, but rather are mathematical objects in their own right, on a par with more familiar objects such as numbers, mappings, groups, and so on. Thus, since types classify the available mathematical objects and govern how they interact, propositions are nothing but special types — namely, types whose elements are proofs.

5.7　計算模型としてのラムダ計算

定義 5.7.1 (computability)**.** 関数 $F : \mathbb{N} \to \mathbb{N}$ が計算可能であるとは，次が成り立つことである．ラムダ式 f が存在して，$\forall x, y \in \mathbb{N}\ F(x) = y \Leftrightarrow fx =_\beta y$ を満たす．

第 6 章

計算複雑性

長谷川立さん 2020 年度の応用数学 XC を中心に学ぶ.

第 7 章

関数プログラミング

計算可能性の定義の 1 つの流儀は，「原始再帰関数による定義」である．従って，計算可能な対象は原始再帰関数による表現を持つ．この表現によって構築された対象を用いて計算を行うための言語を，関数型プログラミング言語という．
本当に，数学基礎論とは，計算可能な対象，計算不可能な対象という観点からの数学の再構成である．データ構造は帰納的定義により定められる．その上の関数は，構成子を初期関数として，「条件節」と言う構文を用いて，論理から場合分けにより定義される．その場合分けの中に自己言及，またはいくつかの関数の集合の中に互いに相互言及の関係がある時，これを再帰的という．再帰性が原始的であることが，$\mathbb{N}$ 上での計算可能性に対応することが知られている．計算自体は簡約である．この時正規形とは，そのデータ構造上の「値」のことである．
リストをデータ構造として正式に定義する．いわばベクトルであるが，$\mathbb{R}^n$ とは限らない，一般化されたベクトルのことをいう．帰納的定義とは，種と関数とを与えて，それら構成子が有限回の適用で作り出せる閉包を取るという形式手法である．この上の原始再帰的関数とは，停止性が約束された関数 (total function) に対応する．また，帰納的定義によって生成したデータ構造の上では，いくつか普遍構成が可能である．組，リストはその代表例である．
リストなどのデータ構造を定義するにあたって，その構成子に注目するというパラダイムは画期的である．わかりやすいだけでなく，理論的にも頑健である．このような方法でリストを扱えるようになったら，リストの上には高階関数が定義できる．（なぜこのタイミングで初めて高階関数が出るのかは，組の上の関数は高階関数とは認識されないことが多いからである）．

$$\begin{array}{ccc} (1,2,3) & \xrightarrow{\mathrm{cons}\circ\mathrm{cons}} & \mathrm{cons}(1,\mathrm{cons}(2,3)) \\ {\scriptstyle \mathrm{f}}\downarrow & & \downarrow{\scriptstyle \mathrm{f}} \\ (\mathrm{f}(1),\mathrm{f}(2),\mathrm{f}(3)) & \xrightarrow{\mathrm{cons}\circ\mathrm{cons}} & [\mathrm{f}(1),\mathrm{f}(2),\mathrm{f}(3)] \end{array}$$

これは可換図式的で，これを利用した高次の関数は Curry 化と呼ばれる．この言葉

7.1　再帰的関数

定義 7.1.1 (再帰的関数定義)**.** 関数 ${}_nC_m$ の再帰的定義は次の様に書く.

program 7.1.1　recursive definition

```
1        if m=0 then 1
2        else if m=n then 1
3        else n-1_C_m + n-1_C_m-1
```

一般に if 文は,「順序付きの場合分け matching」である. 関数 ${}_nC_m$ の引数を表す, 関数 ${}_nC_m$ の再帰的定義を表す program 内での symbol n, m を, **仮引数**と言う. 従って, 仮引数は再帰の中で移り変わっていくが, 実引数はその計算によって定義される関数本体の引数になる. 主従関係である.

注 7.1.2. 0. 僕が 7/6/2018 に考えていた二項係数の特徴付けは, これであった. **再帰的関数定義による関数の特徴付け**に辿り着いていたのだ！その際は次の 3 つで関数 $\binom{n}{m}$ を定義していた.

1. 両端は 1 $\binom{n}{0} = \binom{n}{n} = 1 \ (n \in \mathbb{Z}_{\geqslant 0})$
2. 生成原理 $\binom{n}{m} + \binom{n}{m+1} = \binom{n+1}{m+1} \ (n, m \in \mathbb{Z}_{\geqslant 0})$
3. 0 の海 $\binom{n}{m} = 0 \ (n < 0 \vee m < 0 \vee m > n)$

そして, 次の様に明示的に定義された関数はこの特徴付けを満たす.

$$\binom{n}{m} = \frac{n!/(n-m)!}{m!} \ (0! = 1)$$

1. この二項係数の再帰的定義を視覚的に表現するのが pascal の三角形である. 僕は「モデル」と呼んで居た.

2. $n < m < 0$ は停止条件を満たさない.

7.1.1　$\mathbb{Z}$ 上の再帰関数

例 **7.1.3** (階乗).

program 7.1.2　factorial

```
1    fact(n) =
2        if n=0 then 1
3        else n*fact(n-1)
```

例 **7.1.4** (冪).

program 7.1.3　exponential

```
1        expt(n,m) =
2            if m=0 then 1
3            else n*expt(n,m-1)
```

は，再帰をするたびに計算して，計算量が多い．

program 7.1.4　exponential

```
1        expt(n,m) =
2            if m=0 then 1
3            else if m=1 then n
4            else if mod(m,2) then expt(n*m,m/2)
5            else n*expt(n*n,(m-1)/2)
```

と言う定義で，関数への引数を先に計算する様にすると（値呼び），掛け算の回数は半分に減る（$\log_2 m$ 回）．

例 **7.1.5** (fibonacci number).

program 7.1.5　fibonacci number

```
1    fib(n) =
2        if n=<1 then n
3        else fib(n-1) + fib(n-2)
```

は極めて計算量が多い．しかし，$M = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ として，行列積計算関数を導入すれば，

program 7.1.6　fibonacci number

```
1         expt(M,m) =
2             if m=0 then E
3             else if m=1 then M
4             else if mod(m,2)=0 then expt(M*M,m/2)
5             else M*expt(M*M,(m-1)/2)
```

と言う様に，冪の力を借りて，$\log_2 m$ 回で計算できる．

7.1.2　相互再帰関数

例 **7.1.6** (catalan 数)**.**

$$c_n := c_0c_{n-1} + c_1cn_2 + \cdots + c_{n-1}c_0 \ (c_0 = 1)$$

program 7.1.7　catalan number

```
1     catalan(n) =
2         if n<=1 then 1
3         else catalan2(n,0)
4     catalan2(n,i) =
5         if i>=n then 0
6         else
7             catalan(i)*catalan(n-i-1)+catalan2(n,i+1)
```

7.1.3　$\mathbb{R}$ 上の再帰関数

例 **7.1.7** (指数関数)**.**

program 7.1.8　exponential

```
1     e(n) =
2         if n<0 then 0
3         else e(n-1) + (1/fact(n))
```

fact(i) を n 回計算してしまう．

program 7.1.9　exponential2

```
1         e3(i,n,x) =
2             if i>n then 0
3             else x+e3(i+1,n,x/(i+1))
```

を用いて $e(n) = e3(0, n, i)$ とする．

例 **7.1.8** (Newton 法)**.**

program 7.1.10　Newton method

```
1       Newton(x,e) =
2           if |f(x)|<e then x
3           else Newton (x-f(x)/f'(x),e)
```

例 **7.1.9** (Monte Carlo 法)**.**

program 7.1.11　Monte Carlo method

```
1       incircle(n) =
2           if n<0 then 0
3           else if random(2*n)**2 + random(2*n+1)**2 < 1 then
                incircle(n-1) + 1
4           else incircle(n-1)
```

次の $incircle(n) = incircle3(n, 0, r_0, r_1)$ によって計算量を落とせる．

7.1.4　条件と述語

再帰的関数定義には if 節を多様した．

定義 7.1.10 (predicate)**.** TV $= \{\text{True}, \text{False}\}$ として，TV 値関数を述語という．

従って，述語自体も再帰的関数定義による表現を持つ．

7.1.5　and, or, not

and, or, not という TV 値関数を用いると，if 節のいくつかの条件式をまとめることができる．complete だからである．

7.2　簡約

計算行為は簡約と見れる．数学者が行う計算 $(1+2) \times 5$ を 3×5 は，ラムダ式 $+ = \lambda x.\lambda y.x + y$ に定数 $1, 2$ を関数適用したものである．計算可能な世界では，β 正規形とは，値のことである．

7.3　種々の再帰的関数

原始帰納法と場合分けによる関数定義とを有限回繰り返すことで定義できる関数を原始再帰的関数という．極限を許さない分けだ．これは計算可能であるから，必ず停止する．

7.3.1　原始再帰的関数

定義 7.3.1 (原始再帰的関数／原始帰納法)**.** 漸化式 g と値 $c \in \mathbb{N}$ が存在して，

program 7.3.1　primitive recursive function

```
1        f(n) =
2            if n=0 then c
3            else g(n-1,f(n-1))
```

として定義できる関数 f を**原始再帰的関数**という．g も c も関数ならば良い，引数の数は関係ない．

定義 7.3.2 (total)**.** 引数の値に依らず停止する様な関数と total な関数という．

注 7.3.3. 原始再帰的関数は，自然数についてトータルな関数のクラスである．

7.3.2　相互再帰

7.3.3　末尾再帰

定義 7.3.4 (末尾再帰関数)**.** 次の形の再帰的定義による関数を，末尾再起的という．

```
1        f(x1,...,xn) =
2            if ... then ...
3            else f(a1,...,an)
```

注 7.3.5. 末尾再帰的な関数は実用の観点から重要である．条件に見合う場合分けを見た後，引数の値を更新し，次のループに移るという非常に単純なループのみによる定義であるからである．

7.4　帰納的定義

$\mathbb{N}, \mathbb{Z}, \mathbb{R}$ などの全体空間を定めてきた．一般的にこれはデータ構造の 1 つである．こうしたデータ構造を定義するには，帰納的定義が用いられる．そして，その上の再帰的関数を定義するにあたっての初期関数に，構成子が引き継がれる．自然数上の原始再帰的定義とは，僕らの使い慣れている帰納的定義・数学的帰納法に直接に対応している．

ところでどうやら構成子と言った時，関数は単射に限る．

7.4.1　自然数

定義 7.4.1 (natural number)**.** 自然数とは，対象と関数（零と後者関数と言う）との組 $(0, \mathrm{Succ})$ である．この 2 つの constructor から構成される自由群みたいな閉包が自然数である．この時の「自由」と言うのは，constructor function Succ が関数であり（左一意）また単射であることである．このように，constructor を指定する形でのデータ構造の定義を**帰納的定義**といい，次の様に書くこととする．

program 7.4.1　inductive definition

```
1        inductive set N with
2            0 : N
3            Succ : N -> N
```

注 7.4.2. このようにして帰納的定義によりデータ構造を定義すると，正規形は constructor のみからなる式である．$\mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))$ など．

7.4.2　$\mathbb{N}$ 上の関数

論理関数は事前に定めるとする．また関係=も定める．

例 **7.4.3** (isZero, isS)**.**

program 7.4.2　isZero / isS

```
1        is_0(x) =
2            if x=0 then True
3            else False
4        is_s(x) =
```

```
5               if x=0 then False
6               else True
```

例 7.4.4 (四則演算)**.**　四則演算は，次のように，初期関数 s=Succ のみを使って原始帰納法により定義される．

program 7.4.3　plus / times

```
1         plus(x,y) =
2             if is_0(x) then y
3             else s(plus(s^-1(x),y))
4         times(x,y) =
5             if is_0(x) then 0
6             else plus(times(s^-1(x),y),y)
```

自然数上の，初期関数 s, s^{-1} を用いた原始帰納法とは，自然数の帰納的定義の直接の表現である．

7.4.3　自然数の対

定義 7.4.5 (組)**.** 自然数の組は，$\mathbb{N}$ から，ただ一つの構成子 $\mathbb{N}^2 \to$ Npair のみを用いて次のように定義できる．

program 7.4.4　Npair

```
1         inductive set Npair with
2             pair : N * N -> Npair
```

従って，次が従う．

$$\mathrm{pair}(x_1, y_1) = \mathrm{pair}(x_2, y_2) \;\Rightarrow\; x_1 = x_2 \wedge y_1 = y_2$$

これが組と言う数学的対象の定義だと思っていたが，constructor の単射性とも見れるのだな．

定義 7.4.6. 構成子 pair の逆関数を $\mathrm{pair}_1^{-1}, \mathrm{pair}_2^{-1}$ とする．

命題 7.4.7. fibonacci 数列 (a_n) について，次が成り立つ．

$$\begin{cases} a_{2n-1} = a_n a_n + a_{n-1} a_{n-1} \\ a_{2n} = a_n a_n + 2 a_{n-1} a_n \end{cases}$$

これを用いて関数定義すると，随分複雑になり，何度も $\mathrm{fib2}(n-1), \mathrm{fib2}((n-1)/2)$ が出てくるので，値呼びでは不利である．そこで，let 式を考える．

7.4.4　let 式

数学からの輸入である．

定義 7.4.8 (let 式)**.** 次のような仮引数変換を let 式という．

program 7.4.5　let formula

```
1         let z=x+y in
2             z^3 + z^2 + z + 1
```

これを δ-簡約すると，$(x+y)^3+(x+y)^2+(x+y)+1$ を得る．

注 7.4.9. 値呼びの下では，let z=e in ... にて，e を先に簡約する．しかし... 部は必ず let 項全体を簡約した後に簡約する．

7.4.5　パターンを持つ let 式

let 式を用いて fibonacci 数列を書き直すと，次の fib2 を用いて $\mathrm{fib}(n) = \mathrm{pair}_1^{-1}(\mathrm{fib2}(n))$ と定められる．

program 7.4.6　fibonacci constructor

```
1     fib2(n) =
2         if n=0 then pair(0,1)
3         else if mod(n,2) = 0 then
4             let pair(x,y) = fib2(n-1) in pair(y,x+y)
5         else
6             let pair(x,y) = fib2((n-1)/2) in pair(y*y+x*x,y*y+2*x*y)
```

定義 7.4.10 (pattern)**.** let 式の前件が pair(x,y) = fib2(n-1) などという形をしていた場合，この左辺の項 pair(x,y) をパターンという．まずこの pattern matching を解決して x, y に値を入れてから，let 項が簡約される．

例 7.4.11 (pattern matching)**.** 次の let 式を簡約すると 2^2+3^2 を得る．

program 7.4.7　pattern matching

```
1        let pair(x,y) = pair(2,3) in x^2+y^2
```

7.5　リスト

数学的な対象として数学基礎論で用いられるのは自然数 $\mathbb{N}$ であるが，実用上の観点から，計算機言語としてはリストが一番基本的と言うべきだろう．情報からなるベクトルである．これは情報幾何から再吸収できるのかもしれないが，今はリストの構造を考える．

7.5.1　自然数のリスト

記法 7.5.1 (list)**.** Lisp : (1 4 9 16 25)

Prolog : $[1, 4, 9, 16, 25]$

注 7.5.2. 空なリストには nil という参照名をつけるとする．即ち，$\mathrm{nil} = []$（object としての等価）.

7.5.2　自然数のリストの帰納定義

定義 7.5.3 (list)**.** Python の class に constructor があるが，これを指していたのか！

program 7.5.1　list

```
1        inductive set Nlist with
2            nil: Nlist
3            cons: N*Nlist -> Nlist
```

[] を用いたリストの表現は，この略記である．

注 7.5.4. concat: Nlist*N -> Nlist を構成子として用いても，帰納的定義から得られる空間は変わらない．

例 7.5.5. 全く同様に，is_nil, is_cons, そして

$$\mathrm{first}([1, 4, 9, 16, 25]) := \mathrm{cons}_1^{-1}([1, 4, 9, 16, 25]) = 1$$
$$\mathrm{rest}([1, 4, 9, 16, 25]) := \mathrm{cons}_2^{-1}([1, 4, 9, 16, 25]) = [4, 9, 16, 25]$$

が定義される．

7.5.3　ℕ-list 上の関数

例 **7.5.6** (和).

program 7.5.2　sum

```
1       sum(x)=
2           if is_nil(x) then 0
3           else first(x) + sum(rest(x))
```

または

program 7.5.3　sum

```
1       sum(x)=
2           if is_nil(x) then 0
3           else let cons(n,y) = x in n + sum(y)
```

例 **7.5.7** (長さ).

program 7.5.4　length

```
1       length(x) =
2           if is_nil(x) then 0
3           else length(rest(x))+1
```

例 **7.5.8** (point-wise 自乗).

program 7.5.5　point-wise square

```
1       square(x) = n*n
2       mapsquare(x) =
3           if is_nil(x) then nil
4           else cons(square(first(x)),mapsquare(rest(x)))
```

例 **7.5.9** (多項式 (Horner)). 実は多項式も list のデータ構造をしている.

program 7.5.6　Horner's polynomial

```
1       polyvalue(a,x) =
2           if is_nil(a) = then 0
3           else first(a) + x*polyvalue(rest(a),x)
```

例 **7.5.10** (append)**.**

program 7.5.7　append

```
1        append(x,y) =
2            if is_nil(x) then y
3            else cons(first(x),append(rest(x),y))
```

例 **7.5.11** (reverse)**.**

program 7.5.8　reverse

```
1        reverse(x) =
2            if is_nil(x) then nil
3            else append(reverse(rest(x)),cons(first(x),nil))
```

7.5.4　ℕ-リスト上の原始帰納法

定義 7.5.12 (リスト上の原始帰納法)**.** f に依らない関数 g を用いて次のように定義される f を，リスト上の原始再帰的な関数という．

program 7.5.9　append

```
1        f(x) =
2            if is_nil(x) then c
3            else g(first(x),rest(x),f(rest(x)))
```

reverse はこの形をしており，append は余分な引数 y を 1 つもち，g(first(x),rest(x),f(rest(x),y),y) という形をしている．

7.5.5　ℕ-リストの対

定義 7.5.13.

program 7.5.10　pair of Nlist

```
1        inductive set Nlistpair with
2            pair': Nlist*Nlist -> Nlistpair
```

この上の関数を定義する．

例 **7.5.14** (リスト分割)**.** リストを 2 つに分割する関数を定義する.

program 7.5.11　partition

```
1        partition(n,x) =
2            if is_nil(x) then pair'(nil,nil)
3            else
4                let pair'(y,z) = partition(n,rest(x)) in
5                    if first(x)<n then pair'(cons(first(x),y),z)
6                    else pair'(y,cons(first(x),z))
```

応用例 7.5.15. quick sort はこのリスト分割を用いて行われる．これを用いた再帰的関数として qsort(x) 関数が定義できる．全ては関数である.

7.6　高階函数

さて，リストの上に関数を定義すると，これは高階関数と呼ばれる．この対象を自由に扱えることが，関数型言語の優位性である.

7.6.1　関数引数

次の抽象化としては，関数自体を引数に取りたい．list の次として mapping 型へ手を出す分けである．いちいち関数 f を書き直して Newton 法をやるのではなくて，関数 f を引数として食わせて Newton 法を行いたい！

program 7.6.1　Newton method

```
1    Newton(f,f',x,e) =
2        if |f(x)|<e then x
3        else Newton(f,f',x-f(x)/f'(x),e)
```

例 **7.6.1** (map)**.** 例 7.5.8 での mapsquare を一般化したい.

program 7.6.2　map

```
1        map(f,x) =
2            if is_nil(x) then nil
3            else cons(f(first(x)),map(f,rest(x)))
```

を用いて，

$$mapsquare(x) = map(square, x)$$
$$maptwice(x) = map(twice, x)$$

例 7.6.2 (iterate)**.** リストに対して和や積を求める sum,product を一般化する．

program 7.6.3　iteration

```
1       iterate(f,c,x) =
2           if is_nil(x) then c
3           else f(first(x),iterate(f,c,rest(x)))
```

に対して

$$sum(x) = iterate(plus, 0, x)$$
$$product(x) = iterate(times, 1, x)$$

を得る．

例 7.6.3 (forall と exists)**.** 量化子とは，リストに対する述語である．

program 7.6.4　quantifier

```
1       forall(p,x) =
2           if is_nil(x) then true
3           else if p(first(x)) then forall(p,rest(x))
4           else false
5       exists(p,x) =
6           id is_nil(x) then false
7           else if p(first(x)) then true
8           else exists(p,rest(x))
```

これから，list x について，全称的な述語，存在的な述語の構成の種になる．

7.6.2　λ 式

例 7.6.4. リスト x の各元に m をたすには，

$$mapplus(m, x) = map(\lambda n.m + n, x)$$

とすれば良い．実はこの等式は Curry 化の例となっている．

7.6.3　Currying

定義 7.6.5 (Currying)**.**

$$f(m) = \lambda n.m + n$$
$$g(m, n) = m + n$$

は,

$$f(m)(n) = g(m, n)$$

を満たす.

7.6.4　高階関数の例

例 7.6.6 (Sieve of Eratosthenes)**.**　素数かどうかを判定する高階的な $\mathbb{N}$ 上の述語 isPrime は次のように定義できる.

program 7.6.5　isPrime

```
1        for(p,i,j) =
2            if i>j then true
3            else if p(i) then for(p,i+1,j)
4            else false
5        isPrime(n) =
6            for(lambda i. mod(n,i) != 0, 1, n-1)
```

エラトステネスの篩を用いると次のようにかける. $p = p(x)$ を自然数に関する述語とする.

program 7.6.6　Sieve of Eratosthenes

```
1        sieve(p,i) =
2            lambda(x).
3                if x>i and mod(x,i)=0 then false
4                else p(x)
5        q(x) =
6            if x>i and mod(x,i)=0 then false
7            else p(x)
```

ん？？なんでこんな複雑なんだ？

例 **7.6.7** (行列)**.**

7.7　S 式

現実のプログラミング言語として Lisp を考える．これはリストをさらに拡張した「万能データ構造」であり，その上で原始再帰的な関数を扱える言語である．すごく自由生成っぽい帰納的定義になってるな．この定義を適宜狭めたもの，即ち S 式の中への符号化として，下部データ構造を定義する．

7.7.1　S 式とは

2 分木の本質は，帰納的定義から見れば，$(\mathrm{cons}^n : S^n \to M)$ と $\mathrm{atom} : A \to S$ との 2 要素のみである．

定義 7.7.1 (Symbolic expression)**.** 集合 A 上の S 式とは，A の元を葉とし，枝分かれするときは必ず二股になるような木のことである．この全体集合を $S(A)$ と書く．$S(A) \setminus A =: M$ は（この元を **molecule, dotted pair** という）必ず左の部分木 x と右の部分木 y をもち，$(x.y)$ と表せる．各 x, y も S 式で，これについて繰り返す．

注 7.7.2. 即ち，1. $A \subset S$, 2. $x, y \in S \Rightarrow (x.y) \in S$ を構成子として帰納的に定義されるデータ構造をいう．ただし，1 による構成子は a ではなく $\mathrm{atom}(a) \in S(A)$ と label をつける．

program 7.7.1　inductive definition of S-formula

```
1        inductive set S with
2            atom: A -> S
3            cons: S*S -> S
```

これらは，is_atom, is_cons を定める．

7.7.2　S 式-リスト

定義 7.7.3. いま，atom の集合 A を $A = \mathbb{Z} \cup A_S$ $(\mathbb{Z} \cap A_S = \varnothing)$ と定める．A_S の元を symbol, symbolic atom という．

定義 7.7.4 (list)**.** 任意の symbolic atom$\mathrm{nil} \in A_S$ と $n \in \mathbb{N}$ と S 式 $x_1, \cdots, x_n \in S$ に

ついて，次の形の S 式を S 式-list という．

$$(x_1\ x_2\ \cdots\ x_n) := (x_1.(x_2.(\cdots(x_n.\mathrm{nil})\cdots)))$$

従って，S 式-リスト全体の集合を L とすると，$L \subset \{\mathrm{nil}\} \cup M$ が成り立つ．左辺は list 記法といい，入出力に用いられるのはこちらの記法である．

注 7.7.5. list 記法は dot 記法によって符号化されている．略記である．

7.7.3　S 式上の基本関数

定義 7.7.6. $\mathrm{cons} : S \times S \to M, \mathrm{car} = \mathrm{cons}_1^{-1} : M \to S, \mathrm{cdr} = \mathrm{cons}_2^{-1} : M \to S$ を基本関数という．カーとクダー（クードゥル）と読む．

注 7.7.7. $(1\ 2\ 3\ 4\ 5) = (1.(2\ 3\ 4\ 5)) = \mathrm{cons}(1, (2\ 3\ 4\ 5))$ であるから，$\mathbb{N}$-リストの時と，first=car, rest=cdr に対応する．

7.7.4　S 式-リスト上の関数

例 7.7.8 (append, reverse, sum)**.** is_nil(x) に当たる述語を $x = ()$ として，

program 7.7.2　append

```
1         append(x,y) =
2             if x=() then y
3             else cons(car(x),append(cdr(x),y))
4         reverse(x) =
5             if x=() then ()
6             else append(reverse(cdr(x)),cons(car(x),()))
7         sum(x) =
8             if x=() then 0
9             else car(x) + sum(cdr(x))
```

7.7.5　S 式上の関数

例 7.7.9 (アトムの数)**.** nil 以外の atom の数を数える．

program 7.7.3　natoms

```
1        natoms(x) =
2            if is_atom(x) then
3                if x=() then ()
4                else 1
5            else natoms(car(x)) + natoms(cdr(x))
```

例 **7.7.10** (flatten)**.** nil 以外の atom の list を変える.

program 7.7.4　flatten

```
1        flatten(x) =
2            if is_atom(x) then
3                if x=() then ()
4                else cons(x,())
5            else append(flatten(car(x)),flatten(cdr(x)))
```

7.7.6　真偽値

定義 7.7.11. $\mathrm{t}, \mathrm{nil} \in A_S$ をそれぞれ真と偽と定める.

$$(1 = 1) = \mathrm{t} \qquad\qquad (1 = 2) = \mathrm{nil}$$

注 7.7.12. または nil 以外の全ての S 式は真を表しているとみなすこともある.

7.8　Lisp

数学の内容から飛び出す．更なる S 式の表現を考えて，一度現実のプログラミング言語としての完成を考える．Lisp は，Lisp program 自体も S 式とみなせる点が，天才的な設計発想である.

7.8.1　Lisp のプログラム

7.9　Lisp プログラミング

現実のプログラミング言語を扱う技術を考える.

7.9.1　三目並べ

7.9.2　原住民と宣教師のパズル

7.9.3　マッチングと簡単な定理証明

第 8 章

論理プログラミング

8.1 帰納的述語

今までは，述語は真理値関数だと見做して，if 条件節による再帰的関数定義の一貫で定義していたが，帰納的定義，データ構造を構成子から定義する手段をスピルオーバーさせて，述語を帰納的に定義する．これは，Prolog の言葉に相当する．

8.1.1 自然数

定義 8.1.1 (自然数の帰納的定義を，述語から書く)**.** N を，広大な議論領域についての「自然数である」という述語とする．$\mathrm{N}(0.5) = \mathrm{N}([]) = \mathrm{False}$ である．これを次のように定義することで，自然数の定義と等価になる．

program 8.1.1 natural number

```
1        inductive predicate N with
2            N(0).
3            N(s(n)) <- N(n).
```

8.1.2 述語の帰納的定義

例 **8.1.2** (k 以下での因数を持たない)**.** nil 以外の atom の数を数える．

program 8.1.2 prime2

```
1        inductive predicate prime2 with
2            prime2(n,1).
```

```
3             prime2(n,k+1) <- prime2(n,k), mod(n,k+1)!=0.
```

記号<-とは，右辺が全て成り立つ場合に限って左辺を成り立つと定義する，との義である．

program 8.1.3　prime

```
1         inductive predicate prime with
2             prime(n) <- prime2(n,n-1)
```

定義 8.1.3 (述語の帰納的定義)**.** 帰納的定義を成り立たせる，述語，またはそれらを<-で繋いだものを**確定節 (definite clause)**，**節**，**生成規則 (production rule)** という．述語の帰納的定義は再帰的たり得る．

8.1.3　述語による関数の表現

一般に n 引数の関数は，$n+1$ 引数の述語と考えることができる．これはグラフ表示に他ならない．

$$f(n) = m \qquad\qquad F(n, m) = \mathrm{True}$$

従って，述語は関数よりも表現力が高い．左全域性や右一意性を必要としないためである．

8.1.4　判定と探索

Prolog とは，述語の帰納的定義を直接に実行することの出来るプログラミング言語である．

定義 8.1.4 (query)**.**

$$\leftarrow prime(7)$$

という形の記述を質問 (query) といい，標準出力からは yes. などと返ってくる．変数を含む query を入力すると，探索してくれる．

> 関数を用いる場合，簡約に関する規則によって式が順に簡約されることによって値が求まる．これに対して，述語を用いる場合は，述語が正しくなるようなあらゆる可能性が探索されることによって値が求まる．従って，述語の方が一般的な計算を行うことができる [8].

述語と探索のスキームで考えれば，どんな問題も一言で表せる．

8.1.5　純粋 Prolog

純粋 Prolog とは，構成子 (0,s,nil,cons) しか書くことのできない計算機言語をいう．

例 **8.1.5** (Add)**.** まずは自然数上の演算も基本構成子から定義せねばならぬ．

program 8.1.4　natural number summation

```
1        inductive predicate Add with
2            add(0,m,m).
3            add(s(n),m,s(k)) <- add(n,m,k).
```

例 **8.1.6** (Append)**.** $append(x, y) = z$ について，

program 8.1.5　Append

```
1        inductive predicate Append with
2            Append(nil,y,y).
3            Append(cons(n,x),y,cons(n,z)) <- Append(x,y,z).
```

注 **8.1.7.** これは例 7.5.10 の再帰関数定義と完全に対応している．

program 8.1.6　append

```
1        append(x,y) =
2            if is_nil(x) then y
3            else
4                let cons(n,x)=x in cons(n,append(x,y))
```

Append(cons(n,x),y,cons(n,z)) <- Append(x,y,z) の第 1,2 成分は pattern matching に，第 3 成分は if 節の返り値に対応している．

> この例から分かるように，一言で言うと，Prolog は，構成子によるデータの生成，構成子を用いたパターンマッチングによるデータの分解を基本操作とし，探索によって計算を行うプログラミング言語なのである [8].

8.1.6　定理証明と Prolog

パラダイムを変えよう．「述語と探索」とは，定理証明によって計算が実装されている計算機言語が Prolog だという言語だということになる．

8.2　Prolog

8.2.1　開発史と背景

Prolog は自動定理証明と人工知能の初期の研究が産んだシステムである．それは Herbrand によって開拓された霊性で（第 8.2.4 節），数々の仕事（Prawitz, Gilmore は Herbrand の方法をそのまま実装しようとし，Davis と Putnam はいくつかの heuristics を導入したが効率改善は若干のものでしかなかった）が結実したのが Robinson の導出原理である（第 8.2.6 節）．これが一番計算機上での実装に向いているもので，これが Prolog を産んだ．現在この霊性はどこに受け継がれているのだろうか．

> Kowalski による論理プログラミングの中心的考え方の一つは，アルゴリズムは論理と制御の二つの異なる部分からなるということである論理とは解くべき問題は何であるかについての記述で，制御とはそれをどう解くかについての記述である．論理プログラミングが理想とするのはプログラマはアルゴリズムの論理の部分のみを指定しさえすればよく，制御は論理プログラミングのシステムが勝手に実行してくれるような状況である．残念ながらこの理想は，現在の論理プログラミングのシステムでは，まだ成し遂げられていない [11][12].

これは，宣言型言語の理想そのものだろう．課題は 2 つ．

1. 節と節内のリテラルの順序による制御，論理以外の制御ではカットなど，一部命令的な制御の機構が簡単化すること．
2. Horn 節の表現力不足により，否定が negation as failure という不完全な形で実装されていること．更なるアルゴリズムが待たれている．

歴史 8.2.1 (Prolog)**.** 「論理がプログラミングの言語としても使える」という発想 [10]

はあり，一階述語論理を使ってプログラムの仕様記述を行い，導出原理を用いて計算機上で検証や解析を行う研究は行われていたが，Prolog システムは 1972 年ごろにフランスのアラン・カルメラウアー (Colmerauer) と Robert Kowalski によって，**Horn 節に限れば一階論理は計算機の言語として使える**というアイデアが考案されて誕生した．従ってその名称は，「論理を使ったプログラミング」を意味するフランス語"programmation en logique"に由来している．フィリップ・ルーセル (Phillipe Roussel) によって最初の実装が言語 ALGOL-W や，Gerard Battani と Henri Meloni により Fortran により書かれた interpreter であった．David H. D. Warren がこれを Edinburgh に持ち込み，これが alternative front-end である DEC-10 を産んだ．

論理の手続き的解釈 by Kowalski, Colmerauer

プログラムの節 (Horn clause) は手続きの定義で，クエリーのゴール節の各リテラルは呼び出し (call) の定義になっている．

注 8.2.2 (論理の手続きとして以外の解釈)**.** [11] には次の 2 つが挙げられている．

1. Database としての解釈．この場合には論理プログラミングはデータベースとみなされる．結果として関係データベースをとても自然かつ強力なものへ一般化でき，そこでは関係データベースは基礎単位節 (ground unit clause) のみから構成される論理プログラムに対応する．このように，データ，プログラム，問合せ，view，安全性の制約 (integrity constraint) を統一的に扱う言語として論理を用いるという発想には，理論的にも実用的にも大きな潜在的可能性がある．
2. process としての解釈．この場合にはゴール節を並列して動く process の系とみなされる．計算の一ステップはプロセスをプロセスのけいに置き換えることで，共有される変数はプロセス間の通信チャンネルとして働く．このような解釈により，論理は，オペレーティングシステムへ応用することや，オブジェクト指向プログラミングに使用することが可能である．

歴史 8.2.3 (Fifth Generation Computer Systems (FGCS))**.** 通商産業省（現経済産業省）が 1982 年に立ち上げた国家プロジェクト (initiative) の開発目標である。570 億円を費やし、1992 年に終結した。

1980 年代に入り、日本のコンピュータ産業は輸出も増え、市場規模も 2 兆円まで成長した。従来、通産省は 1983 年ごろまで貿易自由化対策としてコンピュータ企業への助成

金を出していたが、既にそのような直接的な助成金は意義を失っていた。また、海外からも IBM 互換機を輸出する日本に対して風当たりが強くなっていた時期でもある（IBM 産業スパイ事件が起きたのは 1982 年）。そこで、次は第四世代と言われていた時代に、あえて更に先の第五世代コンピュータを開発するプロジェクトを立ち上げ、日本の独自性を打ち出そうとした。この検討が開始されたのが 1979 年である。当時、電子技術総合研究所（現在の産業技術総合研究所）の渕一博らは述語論理によるプログラミングに強い関心を持っていた。渕らは独創性を求めるこのプロジェクトを絶好の機会として働きかけ、第五世代コンピュータの目標は「述語論理による推論を高速実行する並列推論マシンとそのオペレーティングシステムを構築する」というものになった。当初の予定から 1 年延びた 1992 年、プロジェクトは「当初の目標を達成した」として完了した。

注 8.2.4 (この時代の日本ぶいぶい言わせすぎじゃないか？？)**.** 1981 年、第五世代コンピュータに関する国際会議が通産省主導で開催された。ここで、通産省側は八方美人的に野心的な目標をいくつも掲げた。「人工知能が人間知能（人間脳）を越えること」すなわち人間の脳は高速処理や大量処理には向いていないので、それを越える人工知能をつくることが目標と説明された。その代表的な例が、エキスパートシステムである。たとえば、医学の診断や、多様な場合分けに対応する高速な機械制御など。特に期待されたのは、自然言語処理である。正確な機械翻訳や、高度な言語理解を通じた専門的判断など。

これは主に予算獲得のためであった。渕一博は一貫して並列推論マシンの開発が目標であると明言している。渕はプラットフォームが高性能化すれば自然にその応用が出てくると考えていた。しかしながら、実際に大量の資金が投じられて完成したのはアプリケーションのほとんどない並列推論システムだけだった。10 年と 570 億円をかけたプロジェクトは、通産省が喧伝した目標についてはまったく達成しなかった。「本来の目標については達成した」としているが、しかし成果が産業に影響を与えることはほとんどなかった。単に、学術振興と人材育成に寄与しただけだったと言えよう。

この遺産「並列推論システム」を何かに使えないのだろうか．数学のサイバー化で日本がリードし直す，とか．

応用例 8.2.5 (Expert system, Edward Feigenbaum 36-)**.** 「一部では」エキスパートシステム、自然言語処理などの知的処理を伴うプログラムの開発に応用されている。

エキスパートシステムは AI ソフトウェアとして初めて真の成功を収めた。フランスでも盛んに研究され、特に推論の自動化と論理エンジンの研究が進んでいる。Prolog は 1972 年、フランスで開発された言語であり、エキスパートシステムの発展において重要

である。Prolog は一種のシェルであり、任意のエキスパートシステムを受容し動作させるソフトウェア構造と言える。一階述語論理を使ったエンジンを備え、規則と事実を記述できる。エキスパートシステムの開発の道具であり、実際に使える初の宣言型言語であり、人工知能開発用言語として広く使われた。しかし、Prolog は扱いやすい言語とは言えず、その論理の階層は人間の論理とは乖離している。

注 8.2.6. 日本の「第 5 世代コンピュータ」プロジェクトで活発に研究されたためか，珍しく日本語版 wiki の方が詳しい.

定義 8.2.7 (KB: Knowledge base, Knowledge management)**.** 大企業では方法論などのミームを効率的に自己組織化させるために，知識ベースを開発する．これも Expert System の例である．知識ベースの重要な点は内包する情報の質である。良い知識ベースは注意深く書かれた文章を保持して更新し続け、優れた検索システムを有し、格納形式も注意深く考えられていて、分類構造もうまくできている。知識ベースは、構造（格納実体のタイプと個々の知識の関連付け）やその分類基準を示すのにオントロジーを使う場合がある。オントロジーは、クラスのインスタンス群とともに知識ベースの構成要素となる。

オントロジーとは，知識をあるドメイン内の概念と概念間の関係のセットとみなしたときの形式的表現であり、そのドメイン内のエンティティ（実体）を理由付けしたり、ドメインを記述するのに使われる。『共有されている概念化の形式的・明示的仕様』といったように言われることもある。オントロジーは、あるドメインをモデル化するために使われている語彙を提供する。それらはドメイン内に存在しているオブジェクトや概念の型、プロパティ、関係である

応用例 8.2.8. 並行論理プログラミングの考え方を取り入れた言語としては、Relational Language、Concurrent Prolog、Guarded Horn Clauses (GHC) と GHC の拡張である KL1、PARLOG、Strand などがある。 これらの言語では、多くの場合ストリームで通信を行う動的なプロセスの集まりでプログラムを構成する。そのためこれらの言語の処理方式はストリーム並列とも呼ばれる。

Guarded Horn Clauses (GHC) は、1984 年末に設計され 1985 年に発表された並行論理プログラミング言語である [1]。第五世代コンピュータプロジェクトで並列マシンの核言語の検討をしていた上田和紀により設計された。核言語の候補だった Concurrent Prolog を分析する過程で問題点を見付け、それを解決するさらに単純化した言語として設計した。GHC のバリエーションである Flat GHC を基に、近山 隆により KL1 (Kernel Language One) が設計され、第五世代コンピュータプロジェクトでハードウェアと応用

ソフトウェアとの間を繋ぐ核言語として、並列マシンのオペレーティングシステムや KL1 を含む様々な言語処理系、各種応用プログラムの作成に利用された [2]。

　でも本当に金の投入は学問を進めるな……．僕らの芸術はガイアの上に成り立っているのと全く同様に，政治の上に成り立っている．

応用例 8.2.9 (Datalog)**.** Datalog is a declarative logic programming language that syntactically is a subset of Prolog. It is often used as a query language for deductive databases. In recent years, Datalog has found new application in data integration, information extraction, networking, program analysis, security, cloud computing and machine learning.

8.2.2　処理系

歴史 8.2.10 (DEC-10)**.** Edinburgh 大学で DEC-10 と言う計算機上に実現されていた DEC-10 Prolog と言う処理系で用いられた syntax が，その後の Prolog のほとんどの処理系で用いられるミームとなっている．

例 8.2.11 (Ψ)**.** 日本の第五世代プロジェクトで開発された Ψ と言う Prolog 専用計算機の Prolog も，DEC-10 Prolog の syntax を拡張している．

注 8.2.12. 第五世代コンピュータプロジェクトにおいて開発された並列論理型言語 GHC（Guarded Horn Clauses；ガード付きホーン節）の Horn Clauses とはこのホーン節である。

例 8.2.13 (SICStus Prolog)**.**

8.2.3　一階述語論理

定義 8.2.14 (clause, Horn clause)**.** 次の形をした論理式を，**節**と言う．

$$L_1 \vee L_2 \vee \cdots \vee L_n, \qquad (L_1, \cdots, L_n \text{ are literals}).$$

また，否定を含まない literal（＝原子論理式）を含んだとしてもたった 1 つである節を，**ホーン節**と言う．なお，それぞれの場合について，全てが否定形の literal からなる節を **goal 節**，原子論理式を 1 つのみ含む節を**確定 (definite) 節**または strict Horn clause と

言う.

例 8.2.15. 空な論理式は goal 節である．また，positive(=unnegated) literal 1 つのみからなる節は確定節であるが，これを特に**単位節 (unit)** といい，その中でも特に変数を含まないものを**事実 (fact)** という.

注 8.2.16. Horn clause を記号 $\to$ で書き換えると，第一形式を用いて次のようにかける．なお，第一形式 $\to$ と第二形式 $\leftarrow$ に意味論的差異はない.

$$\begin{aligned} L_2 \wedge \cdots \wedge L_n &\to L_1 && \text{（確定節の場合）,} \\ L_1 \wedge L_2 \wedge \cdots \wedge L_n &\to && \text{（ゴール節の場合）.} \end{aligned}$$

注 8.2.17 (Prolog では)**.** これら確定節を Prolog の構文で書くと，次のようになる.

program 8.2.1　Prolog

```
1        L1 :- L2, L3, ..., Ln.
2        L1.
```

また，単位節は，記号:-を省略して 2 のように表される．これは L1 :- true の省略系だと見做せる．ただし，true/0 を L-述語とした．1 の形をした Horn 節を rule，2 の形をした Horn 節を fact と呼ぶ．記号:-に対して，左辺を頭部 (head)，右辺を身体 (body) または目標 (goals) と言う.

定義 8.2.18 (formula)**. L-論理式**とは，次のように帰納的に定義される記号列のことである.

1. literal は論理式である.
2. 論理式 φ, ψ について，$(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \to \psi)$ は論理式である.
3. 論理式 φ と変数 x について，$(\exists x\varphi)$, $(\forall x\varphi)$ は論理式である.

L-論理式全体の集合を Fml_L と書くこととする.

以降，一般の L-論理式についてではなく，2 種の標準形：冠頭標準形と Skolem 標準形のみに議論を絞る.

応用例 8.2.19. Horn clauses play a basic role in constructive logic and computational logic. They are important in automated theorem proving by first-order resolution, because the resolvent of two Horn clauses is itself a Horn clause（何かしらの resolvent を導くため）, and the resolvent of a goal clause and a definite clause is a goal clause

（空節を導くため）. These properties of Horn clauses can lead to greater efficiencies in proving a theorem (represented as the negation of a goal clause).

Propositional Horn clauses are also of interest in computational complexity. The problem of finding truth value assignments to make a conjunction of propositional Horn clauses true is a P-complete problem, solvable in linear time, and sometimes called HORNSAT. (The unrestricted Boolean satisfiability problem is an NP-complete problem however.) Satisfiability of first-order Horn clauses is undecidable.

8.2.4　始まりは証明論から：Herbrand の定理

Herbrand の一階述語論理に対する考察は「命題論理として見れる部分は命題論理の技術を使おう」という姿勢のもので，Herbrand 基底 H という無限集合に注目すると，そこからの真理値関数を 1 つ定めるごとに，Herbrand 領域全体の論理式（L-閉項）に対する真理値表が定まる．すると，元の論理式 A についての全ての解釈が真理値関数 $H \to 2$ という表現を持つことになる．

Herbrand の定理とは，Jacques Herbrand が 1930 年に提出した証明論に於ける定理であり，一階述語論理の論理式の充足不能性（または恒真性）の判定を，Herbrand 基底上の有限回の命題論理の充足不能性の判定に還元する定理である．一階述語論理と命題論理の充足不能性の意味論的定義は全く異なり，後者に於ける充足不能性は各 Herbrand 基底に対する真理値の付与の仕方について有限回の機械的な操作で判定可能である．従って，この定理が以降自動定理証明システム開発に当たっての理論的基盤となった．なお，Herbrand 定理が与えられたからと言って，実際に直接的に証明を構成する，または充足可能性／不能性を判定する（ようなアルゴリズムを与える）という課題は極めて困難で，論理プログラミング言語として代表的な Prolog の誕生は 1965 年の John Alan Robinson による導出原理の提出の仕事を待たねばならない．また，Prolog も，扱う一階述語論理の論理式に強い制約をつけたものになっている．自動定理証明システムの開発はまだまだこれからである[†1].

先ず，Herbrand の定理は，一階述語論理の意味論と命題論理の意味論の接続にあたって，特別な構造「Herbrand 構造」を用意する．これは Herbrand 基底と呼ばれる L-原子論理式を命題変数として，一階述語論理の L-論理式を命題論理的な意味論の世界へ解釈

[†1] 一切の制約のない一般の一階述語論理の論理式の恒真性の判定問題は，Turing 機械の非停止性の認識問題に帰着されるため，決定不能であることはすでに知られている．

するような L-構造である．このことを見ていく．

定義 8.2.20 (Herbrand universe)**.** L-論理式 A の**エルブラン領域**とは，言語 L の記号のうち，次の当てはまるものからなる部分集合 L_A から生成される L_A-閉項全体 Tm_{L_A} のことをいう．

1. A に出現する定数記号
2. A に自由出現する変数記号
3. A に出現する関数記号

ただし，A に 1 も 2 も存在しない場合は，定数記号を L から自由に 1 つ選んで L_A の元とする．

注 8.2.21. 具体的に取れる点の全体領域／閉包を確保したことになる．

定義 8.2.22 (Herbrand structure)**.** L-論理式 A の **Herbrand 構造**とは，Herbrand 領域 Tm_{L_A} を領域とし，その上の L-閉項を次のように解釈する写像 $F_{Tm_{L_A}}$ を解釈写像とする構造 $\mathcal{M}_{L_A} := (L_A, \mathrm{id}_{Tm_{L_A}})$ のことである．解釈写像 $F_{Tm_{L_A}}$ は，L_A に属する記号はそれ自身に，即ち Tm_{L_A} の元はそれ自身に写し，述語記号 $P/n \in L$ は，勝手な関数 $(L_A)^n \to \{\top, \bot\}$ に写す（従って述語記号の解釈は各 Herbrand 構造について異なり得る）ものとする．また，論理記号は命題論理の結合子として解釈する．

定義 8.2.23 (Herbrand-satisfiable)**.** 論理式（の集合）が **Herbrand 充足可能**であるとは，その論理式（の集合）を充す Herbrand 構造が存在することをいう．

命題 8.2.24. Skolem 標準形の論理式は，充足可能ならば Herbrand 充足可能である．逆は自明に成り立つから，即ち Skolem 標準形の論理式にとって，充足可能性と Herbrand 充足可能性とは同値である．

［証明］. Skolem 標準形の論理式 A について，これを充足する構造 $\mathcal{M}$ が存在するとする：$\mathcal{M} \vDash A$．この構造の述語記号 $P/n \in L$ への解釈を，真理値関数と見た時，これが命題 8.2.28 の方法で定める Herbrand 構造 $\mathcal{M}_{L_A}$ について（即ち，次の式の通りに定めた $\mathcal{M}_{L_A}$ について），$\mathcal{M}_{L_A} \vDash A$ である．

$$P^{\mathcal{M}_{L_A}}(t_1, \cdots, t_n) := P^{\mathcal{M}}(t_1^{\mathcal{M}}, \cdots, t_n^{\mathcal{M}})$$

□

例 8.2.25. 論理式 $\neg P(c) \wedge \exists x P(x)$ を考える．これは充足可能である（例えば，領域を $\{\top, \bot\}$ として，そこへの解釈関数を $c \mapsto \bot, c' \mapsto \top$ とし，述語 P を適切に解釈すれば良い）が，Herbrand 領域は論理式内に arity 1 以上の関数記号が含まれないために $Tm_{L_A} = \{c\}$ であり，これでは Herbrand 充足にはなり得ない．

一方，元の論理式に Skolem 関数 c'（ここでは arity 0 の定数記号である）を導入して Skolem 化した論理式 $P(c') \wedge P(c)$ は，言語は $L \cup \{c'\} =: L'$ に拡張され，Herbrand 領域は $Tm_{L'_A} = \{c, c'\}$ となり，上記の証明中に示した方法で，充足する構造 $\mathcal{M}$ を作るたびに対応する Herbrand 構造 $\mathcal{M}_{L'_A}$ も作ることができる．実際，この事実は一般化され，命題 8.2.28 が成り立つ．

定義 8.2.26 (Herbrand basis)**.** L-論理式 A に出現する述語記号と，A の Herbrand 領域 Tm_{L_A} の元から作られる L-原子論理式の全体を，論理式 A の **Herbrand 基底**という．

例 8.2.27. 元の Skolem 標準形の論理式が $\forall x \forall y(P(x) \wedge Q(x, f(x, c)))$ である時，Herbrand 基底は $Base_A = \{P(c), P(f(c, c)), P(f(f(c, c), c)), \cdots, Q(c, c), Q(f(c, c), c), \cdots\}$ となる．

命題 8.2.28 (Herbrand 構造と付値)**.** L-論理式 A の Herbrand 構造 $\mathcal{M}_{L_A} = (L_A, id_{Tm_{L_A}})$ と，L-論理式 A の Herbrand 基底 $Base_A$ 上の真理値関数 $Base_{L_A} \to \{\top, \bot\}$ とは一対一対応する．

［証明］．真理値関数 $\chi_{L_A} : Base_{L_A} \to \{\top, \bot\}$ の値を，各述語 $P/n \in L$ と L_A-閉論理式 $t_1, \cdots, t_n \in L_A$ について，$\chi_{L_A}(P(t_1, \cdots, t_n)) = P^{\mathcal{M}}(t_1, \cdots, t_n)$ と定める．この対応付けは可逆であり，L-論理式 A の Herbrand 構造全体の集合と，集合 $Hom(Base_{L_A}, \{\top, \bot\})$ 上に全単射を定める． □

定義 8.2.29 (H-instance)**.** Skolem 標準形の L-論理式 $A \equiv \forall \vec{x} A(\vec{x}; \vec{a})$ $(\vec{x} = (x_1, \cdots, x_n))$ について，母式 $A(\vec{x}; \vec{a})$ に自由出現している変数に，L-閉項を代入して得る論理式 $A(\vec{t}; \vec{a})$ またはその有限個の連言 $\wedge\{A(\vec{t^i}; \vec{a}) \mid i \leqslant r\}$ を A の**例**という．特に，L-閉項 $\vec{t} = (t_1, \cdots, t_n)$ が全て Herbrand 領域のものである時：$t_j \in Tm_{L_A}$ $(j = 1, \cdots, n)$ これを **H-例**という．ここで，特に連言で結ばれていない H-例の全体を $\Gamma = \{A(t_1, \cdots, t_n) \mid t_1, \cdots, t_n \in Tm_{L_A}\}$ と書くこととする．

以上，準備が整った．まとめると次のとおりである．

Herbrand 構造

Skolem 標準形の論理式 $\mathcal{A}$ の母式 $A(\vec{x};\vec{a})$ は量化記号なしであるから，Herbrand 基底 $\mathrm{Base}_{\mathcal{A}}$ の要素を命題記号（命題変数）とする命題論理式と解釈できる．従って，各 Herbrand 基底への真理値の付値 $\chi_{\mathcal{A}} : \mathrm{Base}_{\mathcal{A}} \to \{\top, \bot\}$ を定めるごとに，命題論理式の解釈を 1 つ得る．これは一階の論理式 $\mathcal{A}$ の解釈と一対一対応する．

先ず，次が成り立つ．

補題 8.2.30. Skolem 標準形の L-論理式 $\mathcal{A}$ の H-例の集合 Γ が，命題論理式の集合として充足可能であるならば，$\mathcal{A}$ は充足可能である．

注 8.2.31. L-論理式 A の母式を $\mathcal{A}$ とする．Γ が命題論理式の集合として充足可能であるとする．すると，任意の $t_1, \cdots, t_n \in \mathrm{Tm}_{\mathrm{L}_{\mathcal{A}}}$ に対して $A^{\mathcal{M}_{\mathcal{A}}}(t_1, \cdots, t_n) = \top$ とする Herbrand 構造 $\mathcal{M}_{\mathcal{A}}$ が存在する．従って命題 8.2.28 より $\mathcal{A}$ は充足可能である．

定理 8.2.32 (Herbrand's theorem)**.** 次の 2 条件は同値である．

1. $\mathcal{A} \equiv \forall \vec{x} A(\vec{x})$ が充足不能である．
2. ある H-例 $\bigwedge\{A(\vec{t^i}) \mid i \leqslant r\}$ が命題論理式として充足不能である．

［証明］. 1.⇒2. について．補題 8.2.30 の対偶より，$\mathcal{A} \equiv \forall \vec{x} A(\vec{x})$ が充足不能ならば，Γ は命題論理式の集合として充足不能である．すると，命題論理のコンパクト性より，そのある有限部分集合も充足不能である．従って，それに対応する H-例 $\bigwedge\{A(\vec{t^i}) \mid i \leqslant r\}$ も充足不能である．

2.⇒1. は明らかである．　□

なお，ある例が充足不能（恒偽）であるとは，その否定が恒真 (tautology) であることと同値であるから，次の双対的な定理も成り立ち，こちらが Herbrand の定理と呼ばれることもある．

定理 8.2.33. Skolem 標準形の L-論理式 $\mathcal{A}$ について，次の 3 条件は同値である．なお，1 と 3 が同値であるという主張が**エルブランの定理**に当たり，1 と 2 が同値であるという主張は Gödel の完全性定理に当たる．

1. A は（一階述語論理の証明体系にて）証明可能である：$\vdash A$.
2. A は恒真 (tautology) である：$\vDash A$.
3. A のある（選言）H-例 $\bigvee\{A(\vec{t^i};\vec{a}) \mid i \leqslant r\}$ が恒真である．

8.2.5　新井先生の展開

定義 8.2.34 (Herbrand normal form)**.** 冠頭標準形の L-論理式 A に対して，その Herbrand 標準形 A^H を，Skolem 標準形の双対として定義する．即ち，次の通りに定義する．

φ 中の各全称量化記号 $\forall y_l$ ($l \in \mathbb{N}$) について，それより前に存在量化記号が n_l 個 $\exists x_1^l, \cdots, \exists x_{n_l}^l$ とあるとすれば，n_l-変数の新たな関数記号 f_l を導入して，φ の母式中の変数 y_l を $f_l(x_1^l, \cdots, x_{n_l}^l)$ で置き換えることで，全称量化記号 $\forall y_l$ を消去することができる．これは拡張された言語 $L \cup \{f_i\}$ での ∃-論理式であり，これを **Herbrand（選言）標準形**という．

なお，この時 Herbrand 領域の構成子の集合 L_A も拡張された．これを $L_H (:= L_A \cup \{f_l\}_{i \in I})$ とする．

定義 8.2.35 (instance : ∃-形)**.** A を ∃-論理式とする．（即ち，量化記号なしの L-論理式 R について，$A \equiv \exists \vec{x}\, R(\vec{x}; \vec{a})$ とする）．この時，式の列 $\vec{t}^i = (t_1^i, \cdots, t_n^i)$ の列 $\{\vec{t}^i \mid i \leqslant r\}$ について，$\vee\{R(\vec{t}^i; \vec{a}) \mid i \leqslant r\}$ を A の**例**と言う．

特に，式の列の列 $\{\vec{t}^i \mid i \leqslant r\}$ が全て Herbrand 領域に入っている時は，**H-例**と言う．

Herbrand 標準形 A^H が ∃-論理式であることに注意すれば，より一般に A が冠頭標準形の場合について，H-例は定義できる．

定義 8.2.36 (instance : 冠頭標準形)**.** A を冠頭標準形の L-論理式 $\exists \vec{y}, \forall \vec{x}, R(\vec{y}; \vec{x}; \vec{a})$ であるとする．変数の列の列 $\{\vec{x}^i \mid i \leqslant r\}$ と L_A-式の列の列 $\{\vec{t}^i \mid i \leqslant r\}$ に対して，言語 L_A での量化記号なしの論理式 $\vee\{R(\vec{t}^i; \vec{x}^i; \vec{a}) \mid i \leqslant r\}$ を，A の **H-例**という．

定理 8.2.37 (Herbrand's theorem : 冠頭標準形について)**.** 冠頭標準形の L-論理式 A について，次の 3 条件は同値である．なお，1 と 3 が同値であるという主張を**エルブランの定理**といい，1 と 2 が同値であるという主張を Gödel の完全性定理という．

1. A は（一階述語論理の証明体系にて）証明可能である : $\vdash A$.
2. A は恒真 (tautology) である : $\vDash A$.
3. A のある H-例が恒真である．

注 8.2.38. これは，任意の一階述語論理の式は冠頭標準形に変換できるから（命題 1.3.25），任意の恒真な論理式 A が恒真であることの証明は，有限資源内で確認出来るこ

との理論的保証となっている．

この定理の証明を自分の言葉でまとめることは叶わなかった．

定理 8.2.39 (Herbrand の定理：単純形)**.** F を L-節の有限集合とする．次の 2 条件は同値である．

1. F は充足不能である．
2. F から得られる基礎例（エルブラン基底）の有限集合で充足不能なものが存在する．

系 8.2.40. 冠頭標準形の論理式 F, G について，

$$F = \exists \vec{y},\ \forall \vec{x},\ R(\vec{y}; \vec{x}; \vec{a}), \qquad G = \neg F = \forall \vec{y},\ \exists \vec{x},\ \neg R(\vec{y}; \vec{x}; \vec{a}),$$

とする．次の 4 条件（正味 7 条件）は全て同値である．

1. F は恒偽（充足不能）である：$\models \neg F$．即ち，G は恒真である：$\models G$.
2. F のある H-例の否定 $\wedge\{R(\vec{t}^{\,i}; \vec{x}^{\,i}; \vec{a})\}_{i \leqslant r}$ が恒偽である．即ち，G のある H-例 $\vee\{\neg R(\vec{t}^{\,i}; \vec{y}^{\,i}; \vec{a})\}_{i \leqslant r}$ が恒真である．
3. F の否定が証明可能である：$\vdash \neg F$．即ち，G が証明可能である：$\vdash G$.
4. F から導出により空節を導くことができる．

注 8.2.41. 条件 2 において，G のある H-例 $\vee\{\neg R(\vec{t}^{\,i}; \vec{y}^{\,i}; \vec{a})\}_{i \leqslant r}$ が goal 節の形をしている．これは何か大事な意味を持つのだろうか？

［証明］. 条件 1~3 にて，「即ち」で結ばれた 2 条件が同値であるのは，$G = \neg F$ から従う．

条件 1~3 の特に論理式 G についての主張は，Herbrand の定理と Gödel の完全性定理 **??**から従う．

条件 3 が成り立つとする．即ち，$\vdash \neg F$．この時 F を仮定すると $F \wedge \neg F = \bot$ を導くことができる． □

定義 8.2.42 (Herbrand basis)**.** 論理式 A のエルブラン領域のうち，原子論理式の族を**エルブラン基底**と言う．

例 8.2.43. 述語 $P(x), Q(g(a, y), f(z))$ からなる論理式のエルブラン基底は，

$$P(a), P(f(a)), P(f(f(a))), P(g(a, a)), P(g(a, f(a))), Q(g(a, a), f(a)), \cdots$$

である．

注 8.2.44. 一般に変数をもたない述語または節のことを基礎例 (ground instance) と言

う。エルブラン基底は節集合から得られる基礎例である。エルブラン基底を導入することで、論理式を命題論理式として扱うことができ、論理式の意味を構文的に決めることができる。

定義 8.2.45 (Herbrand interpretation)**.** エルブラン基底の任意の部分集合 I を**エルブラン解釈**という．直感的には、エルブラン基底の要素の内 I に含まれるものは真、それ以外は偽を表し、真偽値の割り当てにより論理式全体に対する 1 つの解釈を定めたことになる。

本当はまずこちらが示せてから定理 8.2.37 を得る．

定理 8.2.46 (Herbrand の定理 : ∃-論理式について)**.** ∃-論理式 $A \equiv \exists \vec{x}\, R(\vec{x}; \vec{a})$ について，次の 2 条件は同値である．

1. A は tautology である（$\vdash A$）.
2. A のある H-例が tautology になる．

8.2.6　Robinson による導出原理と反駁による証明

単一化のアルゴリズムはすでに Herbrand の論文に含まれていたが，「ユニフィケーションを初めて形式的に研究したのは John Alan Robinson で、一階述語論理の導出手続きを構築する際に一階のユニフィケーションを基盤として使い、組合せ爆発の原因の 1 つ（項を例化したものの探索）を排除することで自動推論技術への大きな一歩とした。」

定義 8.2.47 (導出：命題論理)**.** 次の命題論理での演繹規則を**導出原理**という．

$$\frac{l \vee P \quad \neg l \vee Q}{P \vee Q} \quad (l \text{ is literal})$$

上式は前提となる**親節**，下式を**導出節 (resolvent)** という．

また一般に，２つの節 C_1, C_2 に対して，$L \in C_1, \neg L \in C_2$ を満たす literal L が存在するならば，次の節 C_R を導く規則を導出という．

$$C_R = (C_1 \setminus \{L\}) \cup (C_2 \setminus \{\neg L\})$$

注 8.2.48. 筆者は resolution を「解決」に当たる意味を持つ語として用いられていると思い込んでいたが，どうやら「溶解」「融合」の方である．resolvent は溶液くらいの意味なのかもしれない．例えば SLD resolution の定義**??**の式がわかりやすいが，SLD resolution では節の集合から，２つの節ずつ，ある literal を基点として（例えば SLD resolution の

英語 wiki ではこの literal を"the literal resolved upon in a clause"と表現するところからも語感が分かる）溶け合うようにして導出規則が適用されていき，最終的に一つの節にまとまる．その証明は線型になる．この過程を「融合」「溶解」になぞらえて名付けられたと思われる．

これを一階述語論理上に拡張するにあたって，このような literal L を見つけるにあたって，適切な同一視をする必要がある．これを pattern matching または単一化 (unification) と呼ぶ．これをまずは，最も一般的な形で定義する．

定義 8.2.49 (substitution)**.** ある論理式に対して，変数の置換を**代入**という．これを $\sigma : \mathrm{dom}(\sigma) \to \mathrm{Tm}_L$ と表す．$\mathrm{dom}(\sigma)$ は置換が定義される論理式に出現する変数に依存するが，$\sigma(x) = x$ $(x \notin \mathrm{dom}(\sigma))$ と定めることにより，σ は全変数上に一意に拡張される．すると，Tm_L の構成子に従って，Tm_L 上の写像に帰納的に拡張される．

この代入 $\sigma : \mathrm{Tm}_L \to \mathrm{Tm}_L$ を，式 $t \in \mathrm{Tm}_L$ に作用させることを，式 t 中の変数 $x \in \mathrm{dom}(\sigma)$ を一斉に $\sigma(x)$ に置換することとし，これを $t\sigma := \sigma(t)$ と書くこととする．

定義 8.2.50 (instance)**.** 2 つの代入 σ, τ について，ある代入 θ が存在して関係 $\sigma = \tau\theta$ が成り立つことを，σ は τ の**例**であるという．これは 2 つの変数変換が本質的に同じものであることを意味する．

定義 8.2.51 (unifier)**.** Tm_L 上の等式の有限集合を $E = \{t_i = s_i \mid i < n\}$ とする．

1. 代入 σ が E の**単一化子 (unifier)** であるとは，$\forall i < n,\ t_i\sigma \equiv s_i\sigma$ が成り立つことをいう．
2. E の単一化子 σ が**最汎単一化子 (MGU: Most General Unifier)** であるとは，どんな E の単一化子 τ も σ の例になっていることをいう．この時，$\sigma = \mathrm{mgu}(E)$ と書くこととする．

与えられた集合 E に対して，これに単一子が存在するかどうかを決定する問題を単一化問題という．

定理 8.2.52 (unification algorithm)**.** 単一化問題（下記）は決定可能である．

> 与えられた等式集合 E が単一化可能でないならば"NO"と答え，単一化可能ならば"YES"と答え，この時の最汎単一化子 mgu(E) を出力せよ．

定義 8.2.53 (導出：一階述語論理)**.** 導出原理は，一階述語論理では，導出木の表記法で，

次のように書ける．

$$\frac{p(s) \vee C_1 \quad \neg p(t) \vee C_2 \quad \theta = \mathrm{mgu}(s = t)}{(C_1 \vee C_2)\theta}$$

定義 8.2.54 (refutation)**.** **反駁**とは，節の集合から導出原理のみにより空節を導くことをいう．

注 8.2.55. この語を用いて，系 8.2.40 の主張の一部は次のように書き直せる．

1. 節の集合 C が恒偽である．
2. 節の集合 C から反駁が成功する．

従って，query として与えられた述語 P の否定と，プログラムとして与えられた Horn 節の列 $H_1 \wedge \cdots \wedge H_n$ との選言 $\neg P \wedge H_1 \wedge \cdots \wedge H_n$ から反駁することが出来たら，$\neg P, H_1, \cdots, H_n$ は充足不能だとわかる．よって，$H_1, \cdots, H_n$ に矛盾がない限り，$\neg P$ が $H_1, \cdots, H_n$ の下で偽であること，即ち P が $H_1, \cdots, H_n$ の下で真（論理的帰結である：$H_1, \cdots, H_n \vdash P$）であることがわかる．

特に，Prolog では，節の中でも Horn 節に導出の対象を制限する．この場合の導出を SLD 導出という．

定義 8.2.56 (Selective Linear resolution for Definite clause)**.** query として得たゴール節 $L_1 \wedge \cdots \wedge L_i \wedge \cdots \wedge L_n$ と，規則として定義されたホーン節 $L \leftarrow K_1, \cdots, K_m$ に対する導出を **SLD 導出**という．ただし，L_i と L は，単一化子 $\theta := \mathrm{mgu}(L_i = L)$ の下で単一化されるとする．このとき，示したいゴール節 $\neg L_1 \vee \cdots \vee \neg L_i \vee \cdots \vee \neg L_n$ から次の goal 節が導かれる．

$$\begin{aligned} & (\neg L_1 \vee \cdots \vee \neg (K_1 \wedge \cdots \wedge K_m) \vee \cdots \vee \neg L_n)\theta \\ \Longleftrightarrow \quad & (\neg L_1 \vee \cdots \vee \neg K_1 \vee \cdots \vee \neg K_m \vee \cdots \vee \neg L_n)\theta \end{aligned}$$

注 8.2.57. SLD resolution とは，Robert Kowalski が導入した導出規則に対して，Maarten van Emden がつけた名前である [?]．Linear とは，導出規則を適用する 2 つの節の内の片方を固定し続ける戦略のことをいい，この証明戦略を使った証明が論理式（節）の有限列 $C_1, C_2, \cdots, C_n$ となることから，また Selective とは，節の中で literal として解決される部位が予め定まっている性質をいう．SLD 導出では，確定節では暗黙のうちに頭部の 1 つに定まっており，ゴール節では任意である．Prolog では，プログラム内に記述された順番で照合するという順番があるのみである．

SLD resolution は初めのゴール節を根とした探索木（導出木）を定める．各節はゴール節のいずれかの肯定形リテラルと単一化可能である場合に対応する．最終的に，葉のうち空節であるものと根を結ぶ道が反駁による証明である．Prolog では，back-tracking アルゴリズム（自動後戻りの深さ優先探索）でこの探索を行い，この証明中で構成された解を返すようになっている．

deterministic かでまとめると次のとおり．

> SLD resolution is also non-deterministic in the sense, mentioned earlier, that the selection rule is not determined by the inference rule, but is determined by a separate decision procedure, which can be sensitive to the dynamics of the program execution process.[†2]

Prolog が述語の形式をホーン節に限定した理由は、もし頭部に項の連言を認めるならば、導出時の計算量が爆発的に増大して、全ての解を得ることの保証が難しくなることが必至だからである。

> このようなシステムは，ある種の定理証明システム，例えば群論を扱うシステムが直面する計算の複雑さの問題を回避している．というのも，Horn 節に限定し，証明戦略を制限したことにより，多くの論理プログラムは殆ど決定的に動き（言い換えると殆ど後戻りせずに動き），そのことが演繹を計算と考える有用性の根拠となっている [11].

8.3　Prolog のプログラムの書き方

8.3.1　List

特別な述語を List 形式で書く．

定義 8.3.1 (list 記法)**.**

$$\mathrm{cons}(\mathrm{X1}, \mathrm{cons}(\mathrm{X2}, \mathrm{cons}(\mathrm{X3}, \mathrm{nil}))) = [\mathrm{X1} \mid [\mathrm{X2} \mid [\mathrm{X3} \mid []]]] =: [\mathrm{X1}, \mathrm{X2}, \mathrm{X3}]$$

注 8.3.2. Lisp 記法と対応させると，[X1 | X2] は (X1 . X2) に当たり，[X1, X2, X3] は (X1 X2 X3) に当たる．

[†2] en.wikipedia.org "SDL_solution"

8.4　宣言的意味論

論理プログラミングの宣言的な側面について調べる．一階述語論理と不動点理論の基本的概念を導入し，論理プログラム・モデル・正解代入・不動点と言った概念を正確に定義する枠組みを準備する．また単一化アルゴリズムも見る．

8.4.1　論理プログラムのモデル

8.5　手続き的意味論

宣言的な概念は SLD-導出によって実行可能になるが，この健全性と完全性，及びそれが計算規則とは独立であることを示す．Prolog 処理系から出現チェックを外すとどうなるかも見る．

8.6　否定

現在の Prolog 処理系では失敗を否定と見なす規則により論理的否定を実装しているが，この規則の健全性と完全性を示す．

8.7　無限に続くプロセス

並列プログラムのかける Prolog が日本で誕生した．そのこともあり，無限に続くプロセスについての意味論は大変重要になってきた．

第 9 章

関数型言語

第 10 章

有限オートマトンと正則言語の理論

有限オートマトンと正則言語の言葉で，全ての問題は定式化出来る（PCP，古典的渡船問題など）．その他の計算モデルも本質的には同じだが，観念を符号化する記号と，その間の変化によって人のメンタルモデルが全て得られるのは予想される，計算はこの意味でいくらか人間的な概念である．おそらく，私の一番好きな計算モデルである．

すると，記号列を L と $\Sigma^* \setminus L$ の 2 つに分ける過程を計算と定義し，これを実行するオートマトンの存在性によって計算可能性を定義できる．

10.1 言語

議論領域はアルファベット Σ という有限集合を定めることで，Σ^* に決定される．議論領域はいつでも，文字列の連結についてモノイド $(\Sigma^*, \cdot, \epsilon)$ の構造を持っている．この中で，言語と呼ばれる部分集合 $L = \{x \in \Sigma^* \mid P(x)\}$ を（帰納的定義などで $P(x)$ を）指定することでオートマトン理論が始まる．また，オートマトン理論では言語と問題は同一視され，決定問題のうち「文字列の言語への帰属関係決定問題」に議論を絞ることになる．これは，述語 $P(x)$ を中心に議論が回転するからだと思われる．$P(x)$ の表現に，集合としての L を使うか，決定問題としての L か，グラフとしての L か，論理式としての L かは自由に使う．

10.1.1　全体空間の措定：アルファベット，文字列とその構造

定義 10.1.1 (alphabet)**.** 空でない記号の有限集合 Σ を**アルファベット**という．

例 10.1.2.

1. $\Sigma = \{0, 1\}$
2. $\Sigma = \{a, b, c, \cdots, z\}$
3. 全てのアスキー記号の集合
4. タンパク質のアルファベット $\Sigma = \{A, R, N, D, B, C, Q, E, Z, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$.
5. 計算機言語のアルファベット $\Sigma = \{\text{begin, if, end, for, while, do, else}\}$．この要素を token とも言う．
6. DNA のアルファベット $\Sigma = \{a, t, c, g\}$.

定義 10.1.3 (string / word)**.** アルファベットの有限列 ${}^{<\omega}\Sigma$ の元を**文字列**または**語**という．

文字列 ω について，その長さ n を定義し，$|\omega| := n$ と書く．文字列の i 番目の要素を，数列と見て x_i，文字列に作用する演算子と見て $[i]$ などと書く．

例 10.1.4.

1. Σ^0 の元は空列 $\epsilon := [\,](:= \varnothing)$ のみである．空列は λ とも書く．

注 10.1.5. 概念としても，長さ 1 の列とアルファベット 1 つとは違うし，実装も"1 Null"と"1"とで実体が違うことが多いだろう．

記法 10.1.6.

1. 記号を表すのは $a, b, c, \cdots$ で，記号列を表すのには $z, y, x, w, \cdots$ を使う．
2. アルファベット Σ からなる全ての有限列の集合を $\Sigma^* := {}^{<\omega}\Sigma$ とも書く．$\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$ とする．

注 10.1.7. これは正規表現に通じる．発想は，積 $*$ の中立元が 0 で，和 $+$ の中立元が 1 だというところだろうか．

定義 10.1.8 (concatenation)**.** 列の連結を $xy := x \cdot y$ と書くこととする．すると，$(\Sigma^*, \cdot, \epsilon)$ はモノイドをなし，$(\Sigma^+, \cdots, \epsilon)$ は半群である．

また，指数を $x^0 = \epsilon, x^1 = x, x^2 = x \cdot x, \cdots$ と定義する．

注 10.1.9. 連結は積で，自由モノイド・自由群みたいだ．だから，$\cdot, \epsilon$ などの記法を採用するのか．

定義 10.1.10 (prefix, sufflix, subword, subsequence)**.** アルファベット Σ による 2 つの記号列 $x, y \in \Sigma^*$ について，

1. $\exists u \in \Sigma^*, x = yu$ が成り立つ時，y を x の**接頭語**と言う．
2. $\exists u \in \Sigma^*, x = uy$ が成り立つ時，y を x の**接尾語**と言う．
3. $\exists u, w \in \Sigma^*, x = uyw$ が成り立つ時，y を x の**部分語**と言う．
4. $x = x_1 \cdots x_n$ と部分列 $1 \leqslant i_1 < \cdots < i_m \leqslant n$ が存在して，$y = x_{i_1} \cdots x_{i_m}$ を満たす y を，**部分系列**と言う．

定義 10.1.11 (reverse)**.** $x = x_1 \cdots x_n$ に対し，$x^R := x_n \cdots x_1$ とする．

10.1.2　言語

定義 10.1.12 (language)**.** アルファベット Σ に対して，部分集合 $L \subset \Sigma^*$ を**言語**という．多くは，Σ^* 上に帰納的定義により建設される．自然言語では，L の決め方，アルファベット Σ の決め方が生物学的な理由で発生する．

注 10.1.13. 一階論理の文脈でいう「言語」とは意味がずれる．この意味では，むしろ well-formed formula が一階論理の「言語」をなす．

例 10.1.14 (Post Correspondence Problem)**.**

$$\mathrm{PCP} := \left\{ u_1 \# v_1 \# \cdots u_n \# v_n \;\middle|\; \begin{array}{c} u_i, v_i \in \{0,1\}^* \text{であり，ある列 } i_1, \cdots, i_m \in [n] \text{ が存在し，} \\ u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m} \text{が成り立つ．} \end{array} \right\}$$

定義 10.1.15 (言語の積・反転・Kleene closure)**.** Σ 上の 2 つの言語 L, L' について，

1. $L_1 \cdot L_2 := \{xy \mid x \in L, y \in L'\}$ と定義する．
2. $L^0 = \{\epsilon\}, L^1 = L, L^2 = LL, \cdots$ と定義する．
3. 言語 L の閉包を $L^* = \cup_{n \in \mathbb{N}} L^n, L^+ = \cup_{n=1,2,\ldots} L^n$

注 10.1.16. この定義は，線型部分空間の和と全くパラレルな定義である．文字が形式的に等しくなければ，直積との同型が存在する．

定義 10.1.17 (problem)**.** オートマトン理論で**問題** L とは，決定問題のうち，与えられたアルファベット Σ の文字列 $w \in \Sigma^*$ に対して $w \in L$ かどうかを決定するクラスの「文字列の言語への帰属関係決定問題」を指す.

注 10.1.18. 言語と問題は同一視される．いずれも，$L = \{x \in \Sigma^* \mid P(x)\}$ を決定している述語 $P(x)$ が定めているからである.

また，このクラスの決定問題は理論的に扱いやすいだけでなく，例えば，C 言語のコンパイラが現実的な時間内で快適に使用可能なものが作れるかどうかの問題と「還元 (reduction)」の関係にある.

10.2　非決定性有限オートマトン

定義 10.2.1 (Moore, Mealy, transducer)**.** オートマトンとは machine という概念の数学的抽象化であり，その外界との interaction として，出力は内部状態と同一視し，入力のみを考えるものを Moore model という.

この見方を特に**変換機**という．トランスデューサ（変換機、transducer）は、与えられた入力と動作を伴う状態（両方または一方）に基づいて出力を生成する。

定義 10.2.2 (nondeterministic finite automaton)**. 非決定性有限オートマトン**とは，次の条件を満たす 5-組 $M = (Q, \Sigma, \delta, q_0, F)$ のことである.

1. Q：状態からなる空でない有限集合.
2. Σ：入力アルファベットからなる集合.
3. $q_0 \in Q$：初期状態.
4. $F \subset Q$：受理状態の集合.
5. $\delta \subset Q\Sigma Q$：**状態遷移関係**と呼ばれる 3 項関係.
   (i) 要素 $(p, a, q) \in \delta$ を遷移と呼び，$p \xrightarrow{a} q$ と表す.
   (ii) 状態遷移関係 δ が，$\forall p \in Q,\ \forall a \in \Sigma,\ \exists! q \in Q,\ (p, a, q) \in \delta$ を満たすとき，このオートマトン M を**決定的**であると言う.
   (iii) このとき，$\delta \in \mathrm{Map}(Q \times \Sigma, Q)$ となるので，特に**状態遷移関数**と呼び，決定性の有限オートマトンを特に**有限オートマトン**と言う.

定義 10.2.3 (accepting computation, accepting language)**.** 有限オートマトン M が記

号列 $w = a_0 \cdots a_n$ を受理するとは，遷移の系列

$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_n} q_{n+1} \quad (q_{n+1} \in F)$$

が存在することをいう．この系列を**受理計算**と言う．

次元を一つあげる．$L(M) := \{w \in \Sigma^* \mid M \text{ は } w \text{ を受理する}\}$ を**受理言語**といい，M が集合 L を受理すると言った時は $L = L(M)$ であることをいう（$L \subset L(M)$ ではなく，意味が非常に強いことに注意．言語と一致一対応させたいからである）．

注 10.2.4.

1. $q_0 \in F$ であるとき，このオートマトン M は空列 ϵ を受理する．
2. オートマトンの持つ遷移 δ は非常に圏的である．これを視覚化するオートマトン独自の図式 (diagram) がある．

決定性の場合は，各アルファベットに対する挙動 δ が定まった時点で，全ての語について，挙動が確定している．この挙動 δ^* を考えたい．アルファベットを用いた文字列全体からなる集合 $\Sigma^* := {}^{<\omega}\Sigma$ というデータ構造が持つ帰納的構造に注意して，慎重に議論を進める．

定義 10.2.5 (遷移関数が，文字列の集合上に定める関数)**.** 有限オートマトン $(Q, \Sigma, \delta, q_0, F)$ の遷移関数 $\delta : Q \times \Sigma^* \to Q$ を，次のように，$\Sigma \to \Sigma^*$ の構成に関する帰納法により $q^* : Q \times \Sigma^*$ に拡張する．

1. $\delta^*(q, \epsilon) = q \quad \forall q \in Q.$
2. $\delta^*(q, xa) = \delta(\delta^*(q, x), a) \quad \forall q \in Q, x \in \Sigma^*, a \in \Sigma.$

命題 10.2.6 (M が x を受理することの特徴付け)**.** 次の 2 条件は同値.

1. M は x を受理する.
2. $\delta^*(q_0, x) \in F.$

命題 10.2.7. 有限オートマトン M と記号列 $x, y \in \Sigma^*$ に対して，$\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$.

定義 10.2.8 (accessible)**.** 2 つの状態 $p, p' \in M$ について，p から p' に**到達可能**であるとは，記号列 $x = a_0 \cdots a_n$ と，それが定める遷移系列 $p \xrightarrow{a_0} p_1 \xrightarrow{a_1} \cdots \xrightarrow{a_n} p'$ が存在することをいう.

命題 10.2.9 (有限オートマトンの射)**.** M に対して，次のように定めるオートマトン M' について，$L(M) = L(M')$．また，M が決定的ならば M' も決定的である．

1. 初期状態 p_0 から到達可能でない状態を取り除く．
2. それらを端点とする繊維を取り除く．

10.3　正則言語

正則言語を定義する前に，Aut の中に最後の同型をとっておく必要がある．

定理 10.3.1 (非決定性有限オートマトンと決定性有限オートマトンの等価性)**.** 言語 $L \in P(\Sigma^*)$ に対して，次の 2 条件は同値．

1. L は非決定性有限オートンマトンによって受理される．
2. L は決定性有限オートマトンによって受理される．

［証明］**.** 定義より 2.⇒1. は成り立つから，⇐ を示す．　□

定義 10.3.2 (regular set / language)**.**（決定性）オートマトンによって受理される集合を**正則集合**または**正則言語**という．即ち，関手（？）$L : \mathrm{Aut} \to P(\Sigma^*)$ の値域である．

第 11 章

文脈自由文法

第 III 部

集合論

第 12 章

ZFC と von Neumann hierarchy

Russell の逆理は，素朴な意味での「ものの集まり」と言った言及を再帰的に許すことによって生じる．従って，「ものの集まり」に階層付けすることが一つの対応となる．ここでは ZFC 公理系を考える．公理から集合を定義し，そこから数や関数や基数の定義を引き出して，確かに数学全体の基礎を与える方法の 1 つであることを確認し，同時に ZFC 内外での集合やクラスの更なる取り扱いを議論する．

12.1 準備

定義 12.1.1 (axiom schema)**.** 論理式全体 $\mathrm{Fml}_{\mathrm{L}}$ を走る変数を伴った 2 階の論理式が存在して，その変数に論理式を代入した結果だと捉えられる，論理式の集合のことを**公理図式**という．

例 12.1.2. 推論規則は公理図式である．

定義 12.1.3 (集合の公理系 ZF)**.** 集合の公理は，集合論の言語 $L = \{=, \in\}$ による一階述語論理の言葉で，次のように表せる．ただし，自由変項から始まる公理は，全称量子化 (universally quantified) されているものとする．

0. (Set Existence). $\exists x(x = x)$
1. (Extentionality). $\forall z(z \in x \leftrightarrow z \in y) \to x = y$
2. (Foundation). $\exists y(y \in x) \to \exists y(y \in x \wedge \neg\exists z(z \in x \wedge z \in y))$
3. (Comprehension Scheme). 自由変項 y を持たないような全ての式 φ について，
$$\exists y \forall x\, (x \in y \leftrightarrow x \in z \wedge \varphi(x))$$
4. (Pairing). $\exists z(x \in z \wedge y \in z)$
5. (Union). $\exists A \forall Y \forall x(x \in Y \wedge Y \in \mathcal{F} \to x \in A)$
6. (Replacement Scheme). 自由変項 B を持たないような全ての式 φ について，
$$\forall x \in A \exists! y \varphi(x, y) \to \exists B \forall x \in A \exists y \in B \varphi(x, y)$$

以降，一階述語論理の文字列に対して，次の略記となるような述語を採用する．

記法 12.1.4.

a $x \subset y :\Leftrightarrow \forall z(z \in x \to z \in y)$
b $x = \emptyset :\Leftrightarrow \forall z(z \notin x)$
c (Ordinal successor function). $y = S(x) :\Leftrightarrow \forall z(z \in y \leftrightarrow z \in x \vee z = x)$
d $w = x \cap y :\Leftrightarrow \forall z(z \in w \leftrightarrow z \in x \wedge z \in y)$
e (isSingleton). $SING(x) :\Leftrightarrow \exists y \in x \forall z \in x(z = y)$

7. (Infinity). $\exists x(\emptyset \in x \wedge \forall y \in x(S(y) \in x))$
8. (Power Set). $\exists y \forall z(z \subset x \to z \in y)$
9. (Choice). $\emptyset \in F \wedge \forall x \in F \forall y \in F(x \neq y \to x \cap y = \emptyset) \to \exists C \forall x \in F(SING(C \cap x))$

注 12.1.5.　1. この Zermelo-Fraenkel style では，宇宙は，全ての遺伝的集合のクラスとなる．特に 7. 無限公理の採用の仕方が，順序数を念頭においている．これが特徴的で最初は慣れなかった．2. 整礎性公理は，全ての集合 x について，順序数 α

が存在して $x \in V_\alpha$ であることに同値である．

2. 1 9 を ZFC, 1 9 を ZF，これらから 6. 置換公理を除いたものをそれぞれ ZC, Z といい，これら 4 つからそれぞれ 2. 整礎性公理を除いたものを Z^-, ZF^-, ZC^-, ZFC^- という．

集合はその元によって決まる．この大前提を外延性公理という．外延と内包という二項対立の中の，一方の極にポジションを取った．

12.2　集合生成規則

12.2.1　集合全体のクラス V

12.3　順序，整礎関係，超限帰納法

12.4　整列順序と順序数

12.4.1　整列順序

12.4.2　順序数

定義 12.4.1. 集合 x のランクとは，$x \in V_\alpha$ を満たす最小の順序数 α である．

12.4.3　順序数演算

12.4.4　木

12.5　濃度と基数

12.5.1　基数演算

12.5.2　帰納的定義入門

12.6　フィルターと閉非有界集合

12.6.1　集合上のフィルター

12.6.2　ブール代数

12.6.3　閉非有界集合

12.6.4　poset と Martin の公理 MA

12.7　Ramsey の定理

12.8　遺伝的有限集合

遺伝的有限集合とは，有限個の遺伝的有限集合からなる集合のことを指す．遺伝的有限集合全体のクラス V_ω は，ランク ω の累積的階層であり，従って遺伝的整礎集合全体のクラスの部分クラスである．また，ZF 公理系の 7. 無限公理をその否定で置き換えた公理系のモデルをなす．即ち，無限公理は，他の公理から独立である．各 V_n の濃度は ${}^{n-1}2$ であるから，遺伝的有限集合は全部で可算無限個存在する．同型の例として，Ackermann の符号化が構成できる．

定義 12.8.1 (遺伝的有限集合 hereditarily finite set)**.** 遺伝的有限集合を，次のように帰納的に定義する．

1. 空集合は遺伝的に有限である．
2. 遺伝的に有限な集合 $a_1, \cdots, a_n$ について，集合 $\{a_1, \cdots, a_n\}$ は遺伝的に有限である．

3. 以上によって遺伝的に有限だとわかる集合のみが遺伝的有限集合である．

注 12.8.2. 実は「遺伝的」の部分は，Material Set Theory による構成の仕方を指定しているだけで，結局はただの有限集合である．しかし，「有限集合全体の集合」は一般に集合とは言えないが，「遺伝的有限集合全体の集合」は構成方法までよく定まっていて，確かに可算な集合である．As a property of a set, being hereditarily finite is equivalent (up to isomorphism of sets) to simply being finite. So the ‘hereditary’ part is meaningful only in material set theory, not structurally, unless you see it as a property of a pure set represented structurally as a membership tree.[†1]

命題 12.8.3. 1. 集合は遺伝的に有限である．

2. 集合の推移的閉包は有限集合である．

命題 12.8.4 (遺伝的有限集合全体のクラス)**.** 集合 V_n を次のように定める．

1. $V_0 = \varnothing$
2. $V_{n+1} = \mathrm{P}(V_n)$

最後に，$V_\omega = \bigcup_{n<\omega} V_n$ とすれば，これは遺伝的有限集合全体のクラスである．

注 12.8.5.

1. これは，von Neumann 宇宙（遺伝的整礎集合全体のクラス）の部分クラスになる．
2. $V_{\omega+\omega}$ は ordinary mathematics の宇宙であり，Zermelo の公理系のモデルである．
3. V_κ は ZFC 集合論のモデルである．

これは，ZF 公理系において，無限公理をその否定と置き換えたもののモデルになる．即ち，無限公理は他の ZF の公理からは独立であることがわかる．

命題 12.8.6 (Ackermann (1937))**.** 遺伝的有限集合全体のクラス V_ω について，$V_\omega \simeq \omega$ である．

［証明］**.** Ackermann の符号化 $f : \mathbb{N} \to V_\omega$ を，遺伝的集合の階数に関する帰納法によって，　□

[†1] nLab

第 IV 部

証明論

証明論とは

証明論とは，証明という行為について数学的な解析を行う数理論理学の分野である．より一般的に，推論や思弁について考える哲学の論理学の一分野とも言える．

次が wikipedia の「形式的証明と非形式的証明」の欄であるが，この形式と非形式，形式という言語機能を高度に抽象化させたものと非形式と自然言語との対応について考えたい．どうして言語機能の先に，計算能力を持ったのか．

> 数学で日常的に行われている「非形式的」証明は、証明論で言う「形式的」証明とは異なる。しかしながら、それは形式的証明の高度に抽象化されたスケッチのようなもので、専門家が十分な時間と忍耐を持っていれば、非形式的証明から形式的証明を適切に再構築できるようなものである場合が多い。比喩的に言えば、そのような場合に完全な形式的証明を書くことは、機械語でプログラミングをするようなものである。
>
> 現代では、形式的証明は一般に計算機支援証明を補助としてコンピュータを使って構築される。また、その証明がコンピュータで自動的に検証される点も重要である。形式的証明の検証は簡単だが、証明そのものをコンピュータが構築すること（自動定理証明）は一般には非常に困難である。一方、数学における非形式的証明は査読による検証に何週間も要し、それでもまだ誤りが含まれていることが多い。

第 13 章

様々な証明体系

定義 13.0.1 (形式論理／記号論理)**.** 数学や哲学に於ける論証 (argument) という営みを抽象化した「モデル」または「形式言語」を，**形式論理・記号論理**という．このモデル化のことを**形式化**という．

注 13.0.2. 形式化は，現代数学の基本的な戦略であると同時に，計算機が扱える対象への変換をも意味する．従って，形式論理における推論を計算機を用いて自動化することも可能である．

13.1 形式論理前史

証明論の始祖は Gentzen であると言えるが，その前の大事な母胎は Alfred Whitehead や Bertrand Russell，また Frege と Hilbert が作った．当時から論理計算が存在した．

> 公理的命題論理のバージョンによって採用する公理や推論規則が異なる。例えば、フレーゲは、6 つの公理と 2 つの規則を採用した。バートランド・ラッセルとアルフレッド・ノース・ホワイトヘッドは、5 つの公理からなる体系を示唆した。[†1]

13.2 自然演繹 (natürlichen Schließens)

> この G. Gentzen の学位論文 (G. Gentzen, Untersuchungen über das logische Schließen, Math. Zeitschr. 39(1934/35), I, 176-210: II, 405-431.) こそが証明論

[†1] ja.wikipedia.org「自然演繹」

という分野をいまあるかたちにあらしめたものである．[1]

自然演繹は，論証はいくつかの定まった推論 (inference) から構成され，その構成子の種類によって論証の性質が定まる，という非常に直観的なモデル設計である．換言すれば，論証を推論の繰り返し，簡約の繰り返し，従って計算と見てる計算モデルである．実際，自然演繹の体系は型付ラムダ計算と同一視出来る．これを**論理計算 (logical calculus)** という．

> 論理計算は Frege, Russell, Hilbert 以来，既にあり，その完全性も K. Gödel によって示されていたのだが，新たにこれらの論理計算を導入した理由を Gentzen は「実際の推論にできる限り近い形の形式的な論理体系を作る」ためであったと書いている．さらに Gentzen は NJ, NK が「或る独特な性質（正規化定理のこと）をもっているということや，さらにその点に関しては，直観主義者が否認するところの‘排中律’が特殊な地位を占めているということもわかってきた」と続ける．導入則と除去則がセットになった 9 つの推論規則があり，これらは BHK 解釈 (Brouwer-Heyting-Kolmogorov interpretation) に似て，いわばそれぞれの論理結合詞の (操作主義的) 意味を規定する．[1]

学位論文で提出されたのは 2 つのモデル，NK と NJ である．

13.2.1　推論規則

自然演繹では，証明を**証明図 (proof figure)** または**証明木 (proof tree)** として実装する．

定義 13.2.1 (proof figure / proof tree)**.** **証明図**とは，「仮定」と呼ばれる 0 個以上の論理式 A と「結論」と呼ばれる 1 個以上の論理式 C とを持ち，**推論規則**に対応する長い横棒の 3 組 (A, B, I) である．これを $\dfrac{A}{C}I$ とも書く．**証明木**とは，論理式を節点とし推論規則を枝とする木である．

注 13.2.2. 証明図は射である．例えば，A を論理式として，$A = \{A\}, C = \{A\}, I = \varnothing$ という単位射も証明図である．というより，退化した証明図を命題という．また，前件が空な証明図を定理という．

また，合成について閉じている．

記法 13.2.3 (証明図)**.** Π, Δ などで証明図を表す．

$$\begin{array}{c} B_1 B_2 \cdots B_n \\ \vdots\,\Pi \\ A \end{array}$$

で仮定を $B_1, \cdots, B_n$（この集合を $\{B_1, \cdots, B_n\} =: \Gamma$ と置くことも多い），結論を A とする証明図 Π を表す．ただし，仮定はいくつか省略し，大事なものだけを書くことも多い．

記法 13.2.4 (ならば)**.** ならばの記号は Sequent 計算にて衝突が起こるので，$\supset$ を使う．

定義 13.2.5 (証明図の構成（合成）)**.** 証明図の構成は次の様に行う．２つの図式

$$\begin{array}{c} \Gamma \\ \vdots\,\Pi_1 \\ A \end{array} \qquad \begin{array}{c} \Gamma \\ \vdots\,\Pi_2 \\ B \end{array}$$

があったとする．この時，次の証明図を構成できる．この時の**長い横棒を推論規則と呼び，推論規則を公理により定める**ことで自然演繹の体系が作られる．

$$\begin{array}{cc} \Gamma & \Gamma \\ \vdots\,\Pi_1 & \vdots\,\Pi_2 \\ A & B \end{array}$$

$$\frac{\qquad\qquad}{A \wedge B}\ \wedge I$$

この証明図の仮定は，特に解除もされていないので Γ のままである．証明図が構成されるにあたって，仮定が変化すること（のうち特に仮定が外れること）を脱落 (close, discharge) といい，discharge される仮定に記号 [ ] を付けて，また discharge される構成に所属する横棒の隣に推論規則の名前に discharge に対応する添字を添えて表現される．

以上の証明図の構成を，よく新たに作られた証明図のみを書き，証明図の構成法である推論規則の表現とする．

従って，証明の構成／論証の構成は，命題 A からなる自明な証明図から開始して，証明図の構成の連鎖として理解される．

定義 13.2.6 (weakening)**.** 証明図内に新たな仮定を加える構成を**水増し**という．

定義 13.2.7 (証明可能性)**.** 規定された推論規則により導出可能であることを，**NJ,NK**

にて証明可能といい，それぞれ $\vdash_{\mathrm{NJ}}$, $\vdash_{\mathrm{NK}}$ と表す．図式記号の Π や Γ や Φ,Ψ と同様，メタ言語の記号である．

注 13.2.8. 述語 $\vDash$ はモデル論のものか？

証明図は書いてみた感じ，普段の証明の形式化そのものである．statement を見れば，「仮定して矛盾を示せば良いな」．「これは仮定して矛盾を示す方針しかあり得ないが，二重否定則がないと帰ってこれないので，NK の範囲でしか成り立たないな」とわかる（特定の方向の De Morgan 則など）．NJ,NK はそれぞれ自然な閉包だと感じられるが，なんの閉包なのかあまりわかっていない．

13.2.2　NJ：直観主義論理

この範囲で型付きラムダ計算との対応を持つ．

定義 13.2.9 (NJ)**.** 推論体系 NJ は次の 9 つの推論規則（証明図から次の証明図を得る際に許された操作）からなる．4 つの論理式についての導入則 (I) と除去則 (E) と $\bot$ の扱いについての特徴付け 1 つからなる．

1. $\dfrac{\Phi \quad \Psi}{\Phi \wedge \Psi} \wedge \mathrm{I}$
2. $\dfrac{\Phi \wedge \Psi}{\Phi} \wedge \mathrm{E_L}$, $\dfrac{\Phi \wedge \Psi}{\Phi} \wedge \mathrm{E_R}$
3. $\dfrac{\begin{array}{c}[A]\\ \vdots \\ \Psi\end{array}}{\Phi \supset \Psi} \supset \mathrm{I}$
4. $\dfrac{\Phi \quad \Phi \supset \Psi}{\Psi} \supset \mathrm{E}$
5. $\dfrac{\Phi}{\Phi \vee \Psi} \vee \mathrm{I_L}$, $\dfrac{\Psi}{\Phi \vee \Psi} \vee \mathrm{I_R}$
6. $\dfrac{\Phi \vee \Psi \quad \begin{array}{c}[A]^{l}\\ \vdots \\ X\end{array} \quad \begin{array}{c}[B]^{l}\\ \vdots \\ X\end{array}}{X} \vee \mathrm{E};l$
7. $\dfrac{\begin{array}{c}[\Phi]^{l}\\ \vdots \\ \bot\end{array}}{\neg\Phi} \neg \mathrm{I};l$

8. $\dfrac{\Phi \quad \neg\Phi}{\bot}\neg\mathrm{E}$（$\bot\mathrm{I}$ とも見れる）
9. $\dfrac{\bot}{A}\,\bot$

仮定 Γ の省略の流儀に沿った．

NJ では矛盾に関する規則は次が使われる．

定義 13.2.10 (absurdity rule)**.** 次の推論規則を**不条理則・$\bot$ 除去則**という．

$$\frac{\bot}{A}\,\bot$$

命題 13.2.11 (AR よりも弱い命題なら常に成り立つ)**.**

$$\forall\phi,\ \bot \vdash_{\mathrm{NJ,NK}} \neg\phi$$

［証明］**.** bot, $[\phi]^{l}$ について，1.$\wedge$I，2.$\wedge$E，7.$\neg$I から導ける． □

注 13.2.12. どうやら [7] はこれが absurdity rule と同値という立場なのかな？

命題 13.2.13 (AR の妥当性)**.** 次の Workspace を，選言三段論法 (Disjunctive syllogism / modus tollendo ponens) という．

$$\frac{\Phi\vee\Psi \quad \neg\Phi}{\Psi}\mathrm{DS}$$

次の 2 条件は，NJ 上同値である．

1. 不条理則 $\bot$E が成り立つ．
2. $\vee$I かつ選言三段論法が成り立つ．

［証明］**.** まず 1.⇒2. は，場合分けにより，不条理則のみを使って DS を示せる．

$$\dfrac{\Phi\vee\Psi \quad \begin{array}{c}\Psi\\ \vdots\\ \bot\mathrm{I}\end{array} \quad \begin{array}{c}[\Psi]^{l}\\ \neg\Phi\end{array}}{\Psi}\vee\mathrm{E};l$$

次に，2.⇒1. は，$\bot\cdots_{\vee\mathrm{I}}\bot\vee\Psi$ と $[\bot]^{l}\cdots_{\neg\mathrm{I}}\neg\bot$ から，DS;l により Ψ を得ることが出来る：$\bot\vdash_{\mathrm{NJ,NK}}\Psi$． □

13.2.3　NK：古典論理

NJ で不条理則に当たるものは，次の背理法で置換される．

定義 13.2.14 (reductio ad absurdum)**.** **背理法・帰謬法・不条理則**は次の証明図に表される推論規則をいう．普通仮定は省略される．

$$\dfrac{\dfrac{[\neg A]^{l}}{\vdots}\ \bot}{A}\ \mathrm{RAA};l$$

その結果，全ての命題に 2 値のいずれかが対応する体系となる．なお，この（古典的）背理法，二重否定の除去則，Pierce の法則，あるいは排中律 $\Phi \vee \neg\Phi$ のどれを 9 個目として採用しても等価な体系を得る．

命題 13.2.15 (double negation elimination)**.** 次の推論規則を**二重否定の除去則**という．

$$\dfrac{\dfrac{}{\vdots}\ \neg\neg A}{A}\ \neg\neg\mathrm{E}$$

次の 2 条件は，NJ 上同値である．

1. 排中律かつ不条理則が成り立つ．
2. 二重否定則が成り立つ．

命題 13.2.16 (二重否定の除去則は不条理則よりも強い)**.**

$$\bot \vdash_{\mathrm{NK}} \Phi$$

[証明]． $\dfrac{\bot}{\neg\Psi \supset \bot}$ と $[\neg\Psi]^{l}$ から $\supset \mathrm{E}$ より $\bot$ を得る．$\neg\mathrm{I};l$ より $\neg\neg\Psi$ を得る．二重否定の除去則より Ψ を得る． □

注 13.2.17. こういうものは $\Phi, \neg\Phi \vdash_{\mathrm{NK}} \Psi$ を $\bot$ に置き換えることで証明を得る．

同様に，NJ の証明をこれで置き換えると全て NK の証明になる．

命題 13.2.18 (二重否定の除去則)**.** 二重否定の除去則が成り立つならば，不条理則も成り立つ．

命題 13.2.19 (Peirce's law)**.** 次の証明図を**パースの法則**という．

$$\frac{}{((A \supset B) \supset A) \supset A}\ \text{Peirce's law}$$

$\vdash_{NK}$ Peirce's law である．

命題 13.2.20 (the Law of Excluded Middle)**.** 次の証明図を**排中律**という．

$$\frac{}{\Phi \vee \neg\Phi}\ \text{EM}$$

$\vdash_{NK}$ EM である．

13.2.4　一階への拡張

この体系に無理やり同型対応を入れたら，どんなラムダ計算が出来上がるのだろう？

定義 13.2.21. 直観主義論理，古典論理の別に拘らず，次が一階の言語についての拡張となる．

1. $\dfrac{\begin{array}{c}\vdots\\ \forall x A[x]\end{array}}{A[t]}\forall E$
2. $\dfrac{\begin{array}{c}\Gamma\\ \vdots\\ A[y]\end{array}}{\forall x A[x]}\forall I$（ただしこの時 Γ とは $A[y]$ を導くにあたって undischarged な全ての仮定を表している，また Γ 内に y の自由な出現はないものとする）．「y を任意に取る」というやつである．
3. $\dfrac{\begin{array}{c}\vdots\\ A[t]\end{array}}{\exists x A[x]}\exists I$
4. $\dfrac{\exists x A[x] \quad \begin{array}{c}[A[y]]^{1} \quad \Gamma\\ \vdots\\ C\end{array}}{C}\exists E;1$

これに $=$ の導入則 $\dfrac{}{t = t} = I$ と除去則 $\dfrac{A[t] \quad t = s}{A[s]}$ を含める流儀もある．

注 13.2.22. $=E$ が「等しい両辺ならくっつけられる」とかではなく，論理式への置換であるところが強力である．多分．

13.2.5　Curry-Howard 対応

W. Howard, The formulae-as-types notion of construction, in J. P. Seldin, J. R. Hindley (eds), To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, 479-490(1982). で指摘された.

> 自然演繹での証明とプログラムの対応および命題と型の対応，さらに証明の変形と型付きラムダ計算における計算との対応は，[Howard82] において Curry-Howard 対応 (isomorphism) として定式化されて証明論と理論的計算機科学の交流をもたらした，cf. [Girard89]. さらに Per Martin-Löf, An intuitionistic theory of types, Twenty-five years of constructive type theory, Oxford Logic Guides 36, pp. 127 - 172, Oxford UP(1998). における型理論は BHK 解釈と自然演繹に刺激を受けているものと考えられる.

13.2.6　Glivenko の定理と Gödel の二重否定翻訳

定理 13.2.23. Γ を命題論理式の有限集合，Φ を命題論理式とする．次の 2 条件は同値である.

1. $\Gamma \vdash_{\mathrm{NK}} \Phi$
2. $\Gamma \vdash_{\mathrm{NJ}} \neg\neg\Phi$

注 13.2.24. 於戯面白い，これは古典論理と直観主義論理との関係についての直観もちょうど芽生えていたものだ．この定理はつまり，「古典論理と直観主義論理は，"戻ってこれない"点を除いて違いはない」ことを保証したもので，古典論理の証明は，最後に一回だけまとめて二重否定の除去則を使えばいいという証明の標準的な方針を与える.

しかし，一階論理では次の段階について次の結果が成り立つので，一般に Glivenko の定理は成り立たない.

命題 13.2.25 (クリプキモデル)**.** (1) を $\neg\neg\forall x\Phi(x)$，(2) を $\forall x\neg\neg\Phi(x)$ とする．次の 2 条件が成り立つ.

1. (1) $\vdash_{\mathrm{NJ}}$ (2) が成り立つ.
2. (2) $\vdash_{\mathrm{NJ}}$ (1) は成り立たない.

13.2.7　正規化定理とシーケント計算

正規化定理はステートメントも証明も Gentzen の学位論文にはなく，正規化定理の証明の出版には D. Prawitz, Natural deduction. A proof-theoretical study, Almqvist and Wiksell, 1965. を待たなければならなかった．なお，Gentzen は A. Turing による型付きラムダ計算に対する正規化定理と同工異曲の証明を少なくとも NJ に対しては持っていたことがわかっている．「この [Prawitz65] は当時の証明論に自然演繹の復権をもたらしたと思う.」とのことである [1].

> Gentze は「標準形になおされた証明の最も本質的な性質を一言にして言えば，それは‘まわり道がない’ということである．証明の最後の結論に含まれている概念は，その結論を得るために必然的に用いざるを得ないものであるが，それ以外の一切の概念は，この証明には現れない」と述べている．[1]

定理 13.2.26 (normalization theorem)**.** 純粋に論理的な証明は，すべて「導入則の後に除去則が使われることがない形」に変形することができる.

> さて Gentzen が sequent calculi LJ, LK を導入した理由は，自然演繹 NJ, NK では基本定理を述べるためにふさわしくないからだと言っている．少し長くなるが [Gentzen34/35] より引用しよう.
>
> 基本定理をすっきりした形で表現し証明することができるようにするためには，特にそれに適した論理計算を基礎におかねばならなかった．この目的のためには，自然な論理計算が不適当であることはわかっていた．確かにそれは基本定理を成立させるための本質的な性質をすでに示してはいる．しかし，前にも注意しておいたように，この性質に関連して排中律が特殊な地位を占めている限りは，このことは直観主義論理にのみ限定されてしまうのである.
>
> つまり正規化定理としての基本定理は直観主義論理 NJ についてはきれいに述べて証明することができるが，古典論理 NK についてはそうではない．両方の論理に共通に成り立つ事実として定式化して証明するために論理計算自体を変更したのである．[1]

13.3　Sequent calculus

ゲンツェンは数論の一貫性を確立したいと考えており、自然演繹をさっそく応用した。彼は、その証明の複雑性に不満を持ち、1938 年にはシークエント計算を新たな証明の道具として考案した。1961 年と 1962 年の一連の講義で、Dag Prawitz は自然演繹の包括的なまとめを行った。彼の 1965 年の学術論文 Natural deduction: a proof-theoretical study は自然演繹の最終版ともいうべきもので、様相論理や二階述語論理への応用も含んでいた。[†2]

13.4　ラッセル=ヒルベルト系

G. Frege が開発し，B. Russell と D. Hilbert が整えた証明論である．

[†2] ja.wikipedia.org「自然演繹」

参考文献

[1] 新井敏康『Gentzen から始まる証明論 50 年』

[2] Turing, A.M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem". Proceedings of the London Mathematical Society 2, 42: 230 - 265.

[3] Herman H. Goldstine, John von Neumann, (1948). "Planning and Coding of Problems for an Electronic Computing Instrument". The Institute for Advanced Study.

[4] Sanjeev Arora, and Boaz Barak, (2009). "Computational Complexity: A Modern Approach".

[5] Alonzo Church, (1936). "An unsolvable problem in elementary number theory," American Journal of Mathematics, Vol. 58, 1936, pp. 345-363.

[6] Alonzo Church, (1936). "A note on the Entscheidungsproblem," Journal of Symbolic Logic, Vol. 1, 1936, pp. 40-41.

[7] 萩谷昌己，西崎真也著『論理と計算のしくみ』（岩波オンデマンドブックス，2007）

[8] 萩谷昌己著『関数プログラミング』（日本評論社，情報数学セミナー，1998）

[9] J. Alan Robinson, A Machine-Oriented Logic Based on the Resolution Principle. JACM, Volume 12, Issue 1, pp. 23 - 41. 1965.

[10] Kowalski, R. A., Predicate Logic as a Programming Language, IFIP74, 569-574.

[11] John Wylie Lloyd 著，佐藤雅彦，森下真一訳『論理プログラミングの基礎』（産業図書株式会社，1987）

[12] 森下真人著『知識と推論』（共立出版，情報数学講座第 10 巻，1994）

[13] John E. Hopcroft, Jeffrey D. Ullman, Rajeev Motwani, "Introduction to Automata Theory, Languages, and Computation (2nd)" (2000).