归纳定义

「归纳定义」并非一个陌生的概念，下面就是两例：

e.g.1 按如下约定定义合式公式(wff)：

① 任何命题符号都是 wff；
② 若 α 是 wff，则 $(\neg\alpha)$ 也是；
③ 若 α 与 β 均是 wff，则 $(\alpha\wedge\beta)$、$(\alpha\vee\beta)$、$(\alpha\to\beta)$ 也是；
④ 别无其它。

e.g.2 「递归函数」被归纳定义为：

① Z, Suc, π^i_n 均为递归函数
② 若 $f:\mathbb{N}^k\to\mathbb{N}$ 与 $g:\mathbb{N}^{k+2}\to\mathbb{N}$ 均为递归函数，且 $h:\mathbb{N}^{k+1}\to\mathbb{N}$ 由 f 和 g 经原始递归操作得到，则 h 也是递归函数.
③ 若 $f:\mathbb{N}^k\to\mathbb{N}$，$g_1,\dots,g_k:\mathbb{N}^l\to\mathbb{N}$ 均为递归函数，那么 $h=f(g_1,\dots,g_k)$ 也是递归函数.
④ 若 $f:\mathbb{N}^k\to\mathbb{N}$ 是递归函数且满足 $\forall y\in\mathbb{N}$，$\exists x_1,\dots,x_{k-1}\in\mathbb{N}$ 使得 $f(x_1,\dots,x_{k-1},y)=0$，那么 $g(x_1,\dots,x_{k-1}) := \mu y[f(x_1,\dots,x_{k-1})=0]$ 也是递归函数
⑤ 别无其它。

无疑，它们具有相似之处：均从一些「初始元素」(如命题符号、Z) 出发，经过若干条规则「生成」一整个属类。本章的首个目标，就是严格地定义**什么是「归纳定义」**，抽取出它们的共性，并提供一套通用的、简洁的符号系统来表述任何「归纳定义」。

随着归纳定义而来的，还有我们熟知的「归纳证明法」。在讨论 wff 的性质时，我们常用到归纳证明，例如

e.g.3 求证「任何wff之解读都无歧义」时，我们按wff的构造过程作归纳，先讨论「初始元素」之解读无歧义，再说明②③的变换不会破坏这一性质。

而讨论递归函数的性质时，我们也大量应用归纳证明法，比如

e.g.4 求证「递归函数都是可计算函数」时，我们按递归函数的产生过程作归纳，先讨论基础的 Z，Suc，π_i^n 可计算，再证明②③④的操作不破坏可计算性。

无疑，尽管二者所涉对象全然不同，但方法有不少相似之处：从对象被归纳定义（构造）的次序着手，逐步完成证明。归纳证明法的次序，与所涉对象的定义次序耦合在一起，恰好体现出了对象的生成结构。本章的第二个目标，正是研究抽象意义上的「归纳证明技术」，提供归纳证明的模版。从这种高度往下观望具体实例（如wff的归纳法，递归函数的归纳法），就可以轻而易举地信服其正确性。

以下，我们约定用 Σ 代指符号集；「关系」、「断言」、「谓词」、「性质」等诸多等价的说法都用集合来表示，统统称为「集合」。

我们探讨的对象为

1° 怎样「递归定义」集合 $Q \subseteq (\Sigma^*)^k$ $(k \in \mathbb{N})$

2° 怎样「递归证明」有关 Q 的性质。

e.g. $\Sigma := \{0, S\}$. $k=1$

$$NAT := \{0, S0, SS0, SSS0, \cdots\}$$

[即 Σ^* 之中的「自然数」]

如何递归定义 NAT？如何递归证明 NAT 的性质（比如：$\forall x \in NAT, SSx \in NAT$）？

def 判断形式 (Judgement form):

形如「$J(x_1, x_2, \cdots, x_k)$」的式子。其中，

J ~~是~~是一个固定不动的标识符，用于命名。

$x_1, x_2, \cdots, x_k$ 是不定元，均指代 Σ^* 中的对象，但其本身不属于 Σ^*，仅用作占位符。

以上定义未有有些费解，下面举实例来说明。①与②都是判断形式，而③与④则不是。

设 $\Sigma := \{0, S\}$

① $nat(x)$

② $add(x, y, z)$

符号 x, y, z 本身不属于 Σ^*，但它们指代 Σ^* 中的对象。

③ $nat(S0)$

④ $add(SS\cancel{x}, y, Sz)$

式中含有已被「实例化」的对象

remark. 「形式」一词在中文里似乎带有「流于浅表」的贬义。其实，在西方人的语汇中，「form」指的是「至高的形相」或「模版」。柏拉图所谓的「形相」，即不灭的、完全真实的、无杂质的~~存在~~绝对存在，而世间人眼所见的物体，则是依照形相制造出来的。不妨从这个角度理解上述定义。

def 判断实例:

依据判断形式「造出的」实例。换言之，即把判断形式中的部分或全体不定元特殊化、具体化后所得式子。（例如方才看见的③④）

def 判断:

判断形式和判断实例的总称。

值得提请注意的是：「判断」纯粹是个语法对象，目前应当视其为一串符号，无实际语义。比方说，「$nat(x)$」纯粹是个式子而已，即便我们心里期待它表达「x是自然数」这一语义，但它目前尚不能做到。

def 规则:

形如 $\dfrac{J_1, \cdots, J_n}{J}$ 的式子。其中 $J_1, \cdots, J_n$ 及 J 均为判断。~~我们把参数表省略~~

特别地，若规则的 $n=0$，则称其为公理。

e.g. $\frac{nat(x)}{nat(Sx)}$, $\frac{ascend(n::l) \quad leq(m,n)}{ascend(m::n::l)}$

都是规则。

def 推导关系.

设 $\frac{J_1,\dots,J_n}{J}$ 是一条规则，记之为 r。那么：

(1) 称 J 可由 $J_1,\dots,J_n$ 经 r 推得，记作

$$J_1,\dots,J_n \xRightarrow{r} J.$$

(2) 设 r 中出现的全体不定元为 $x_1,\dots,x_s$。选取其中若干，进行实例化，得到 $J_1',\dots,J_n'$ 及 J'。称 J' 可由 $J_1',\dots,J_n'$ 经 r 推得，记作 $J_1',\dots,J_n' \xRightarrow{r} J'$.

注意 $\frac{J_1\cdots J_n}{J}$ 上下不定元是绑定的。上面的 x 换了，下面也要跟着换。

e.g. 设规则 $r := \frac{even(x)}{odd(Sx)}$. 那么

① $even(x) \xRightarrow{r} odd(Sx)$

② $even(S0) \xRightarrow{r} odd(SS0)$

③ $even(SSSx) \xRightarrow{r} odd(SSSSx)$

…

def 生成.

设 R 是若干规则（可以有无穷多）之集合，而 J 是一个判断。如果存在一个有穷的判断序列 $(I_1, I_2, \dots, I_m)$ 满足

(1) $J = I_m$.

(2) $\forall i = 1,2,\dots,m$，要么 I_i 可经 R 中某公理 r 推得（即 $\exists r\in R: \xRightarrow{r} I_i$），要么可由前面的若干判断经 R 中某规则 r 推得（即 $\exists j_1,\dots,j_n < i$ 及 $r\in R: I_{j_1},\dots,I_{j_n} \xRightarrow{r} I_i$）。

那么称 J 可由 R 生成，记作 $R \vdash J$.

我们很容易将该定义类比邏輯系统中「证明」的定义。

e.g. 设 $\Sigma := \{0, S\}$, $R := \left\{\frac{}{nat(0)}, \frac{nat(x)}{nat(Sx)}\right\}$

那么 $nat(S0)$, $nat(SSS0)$ 等判断均可由 R 生成，而 $nat(0S)$, $add(0,0,0)$ 等均不可由 R 生成。

从上例可见，指定了Σ与规则集R以后，R就能生成具有特定「长相」的判断，把这些判断收集起来构成集合，就是「归纳定义」的实质。

def 归纳定义

选定Σ和规则集R。又设$J(x_1,\cdots,x_k)$是一个判断形式。我们称R和J归纳定义了集合

$$Q := \{(c_1,c_2,\cdots,c_k)\in(\Sigma^*)^k \mid R\vdash J(c_1,\cdots,c_k)\}.$$

R和J就好比筛子，把$(\Sigma^*)^k$中的一部分挑选出来，「定义出」集合Q——每取出一组$c_1,\cdots,c_k\in\Sigma^*$，判断形式$J(x_1,\cdots,x_k)$就具体化为$J(c_1,\cdots,c_k)$，而$R\overset{?}{\vdash} J(c_1,\cdots,c_k)$决定了$(c_1,\cdots,c_k)\overset{?}{\in} Q$。

e.g.1 设$\Sigma := \{0,S\}$，$R := \left\{\dfrac{}{nat(0)}, \dfrac{nat(x)}{nat(Sx)}\right\}$，判断形式取为$nat(x)$。那么，$R$与$nat(x)$归纳定义了

$$Q = \{c\in\Sigma^* \mid R\vdash nat(c)\}$$
$$\overset{*}{=} \{0, S0, SS0, SSS0, \cdots\}$$

其中$(*)$步是由直观得到的。

e.g.2 设$\Sigma := \{\emptyset, :, S, 0\}$，

$$R := \left\{\frac{}{nat(0)}, \frac{nat(x)}{nat(Sx)}, \frac{}{list(\emptyset)}, \frac{list(l)\quad nat(x)}{list(x::l)}\right\}$$

判断形式取为$list(l)$。那么，R与$list(l)$归纳定义了

$$Q = \{c\in\Sigma^* \mid R\vdash list(c)\}$$
$$\overset{*}{=} \{\emptyset, 0::\emptyset, S0::\emptyset, SS0::\emptyset, \cdots, 0::0::\emptyset, 0::S0::\emptyset, \cdots\}$$

[即一切「自然数列表」]

其中$(*)$步也是通过观察得到的。

但「直观」毕竟不严格；况且，如果 R 和 $\mathcal{J}$ 很复杂，那么由它们定义的集合未必清晰可辨，难以用「观察」来研究之。在种种情形下，「归纳证明法」成为了我们的得力助手。

Theorem 1 (归纳证明法)

设 R 与 $\mathcal{J}^*(x_1,\cdots,x_k)$ 归纳定义了集合 Q. 设

命题 p：某个包含 k 个自由变元的命题.

命题 $p' :=$「$\forall c_1,\cdots,c_k \in Q$ 均使 $p(c_1,\cdots,c_k)$ 成立」

命题 $p'' :=$「$\forall \dfrac{\mathcal{J}_1(e_{11},\cdots,e_{1k_1}),\cdots,\mathcal{J}_n(e_{n1},\cdots,e_{nk_n})}{\mathcal{J}^*(e_1,\cdots,e_n)} \in R$ 有：

{若 $R\vdash \mathcal{J}_1(e_{11},\cdots,e_{1k_1})$ … $R\vdash \mathcal{J}_n(e_{n1},\cdots,e_{nk_n})$，
且每个与 $\mathcal{J}^*$ 同型的 $\mathcal{J}_i$ 均使 $p(e_{i1},\cdots,e_{ik})$ 成立，
则 $p(e_1,\cdots,e_k)$ 必成立。}」

我们有 $p'' \Rightarrow p'$.

remark. 命题 p'' 的描述有几点须留神：

① 我们仅讨论形如 $\dfrac{\cdots}{\mathcal{J}^*}$ 的规则。若 R 中有些规则的「分母」并不与 $\mathcal{J}^*$ 同型，则弃之不顾。

② 我们之所以用 e 而非 x 来记判断中的各~~参数~~，是因为规则中的判断未必是判断形式，而有可能是判断实例。使用 e (expression) 来记，显得更清楚。

③ 所谓「同型」（$\mathcal{J}$ 与 $\mathcal{J}^*$）是说，$\mathcal{J}$ 是按形式 $\mathcal{J}^*$ 的模子造出来的。

这么讲解仍难免抽象。在证明以前，不妨先看几个例子。

e.g. 1

设 $R := \left\{ \dfrac{}{nat(0)}, \dfrac{nat(x)}{nat(Sx)} \right\}$ 与 $\mathcal{J}^* := nat(x)$

归纳定义了集合 Q. ($k=1$)

命题 $p(y) :=$「y 形如 $\underbrace{S\cdots S}_{0\text{个或多个}}0$」

命题 $p' :=$「$\forall c\in Q$，$\underbrace{c\text{皆形如}\overbrace{S\cdots S}^{\text{0个或多个}}0}_{\text{即 }p(c)}$」

命题 $p'' :=$「对于 $\dfrac{}{nat(0)}$，若 $\underbrace{\cdots}_{\text{永真}}$ 则 $\underbrace{0\text{形如}\overbrace{SS\cdots S}^{\text{0或多}}0}_{\text{即 }p(0)}$；对于 $\dfrac{nat(x)}{nat(Sx)}$，若 $R\vdash nat(x)$ 且 $\underbrace{x\text{形如}\overbrace{S\cdots S}^{\text{0或多}}0}_{\text{即 }p(x)}$，则 $\underbrace{Sx\text{形如}\overbrace{S\cdots S}^{\text{0或多}}0}_{\text{即 }p(Sx)}$。」

e.g.2（从本例开始不再写出 p）

设 $R := \left\{ \dfrac{}{add(0,0,0)}, \dfrac{add(x,y,z)}{add(Sx,y,Sz)}, \dfrac{add(x,y,z)}{add(x,Sy,Sz)} \right\}$

与 $J^* := add(x,y,z)$ 定义了 Q。$(k=3)$

命题 $p' :=$「$\forall (a,b,c)\in Q$ 有 $R\vdash add(b,a,c)$」

命题 $p'' :=$「以下三者成立：

① 对于 $\dfrac{}{add(0,0,0)}$，$R\vdash add(0,0,0)$ 成立；

② 对于 $\dfrac{add(x,y,z)}{add(Sx,y,Sz)}$，若 $R\vdash add(x,y,z)$ 且 $R\vdash add(y,x,z)$，则 $R\vdash add(y,Sx,Sz)$；

③ 对于 $\dfrac{add(x,y,z)}{add(x,Sy,Sz)}$，若 $R\vdash add(x,y,z)$ 且 $R\vdash add(y,x,z)$，则 $R\vdash add(Sy,x,Sz)$。」

e.g.3

设 $R := \left\{ \dfrac{}{nat(0)}, \dfrac{nat(x)}{nat(Sx)}, \dfrac{nat(x)}{leq(0,x)}, \dfrac{nat(x)}{geq(x,0)}, \dfrac{leq(x,y)}{leq(Sx,Sy)}, \dfrac{geq(x,y)}{geq(Sx,Sy)} \right\}$ 与

$J^* := leq(x,y)$ ~~定义了Q~~ 注意并非 R 中所有规则的分母皆与 J^* 同型。

命题 $p' :=$「$\forall (x,y)\in Q$ 有 $R\vdash geq(y,x)$」

命题 $p'' :=$「以下两者成立：

① 对于 $\dfrac{nat(x)}{leq(0,x)}$，若 $R\vdash nat(x)$，则 $R\vdash geq(x,0)$ 成立；

② 对于 $\dfrac{leq(x,y)}{leq(Sx,Sy)}$，若 $R\vdash leq(x,y)$ 且 $R\vdash geq(y,x)$ 成立，则 $R\vdash geq(Sy,Sx)$ 成立。」

看罢几例，相信不再会有理解上的

困难。下面我们证明之。

proof.

假设 p'' 成立，我们尝试推出 p'。

任取 $c_1,\cdots,c_k \in Q$，据 Q 的定义有 $R \vdash J^*(c_1,\cdots,c_k)$。又根据「生成」的定义，判断实例 $J^*(c_1,\cdots,c_k)$ 能够经有穷的判断序列 $(I_1,\cdots,I_m)$ 推得。由于每一步推导必然是通过 $\frac{\cdots}{J^*(\cdots)}$ 这样的规则完成的，因而结合 p'' 易知，判断序列 $I_1,\cdots,I_m$ 上的每一条判断之参数事实上都将使 p 成立（可用 $\mathbb{N}$ 上的数学归纳法证明，此处略去）。作为特例，$J^* = I_m$ 之参数 $(c_1,\cdots,c_k)$ 当然也使 p 成立。

于是 p' 成立。■

Theorem1 想说明的道理非常简单：既然 Q 中的元素 $(c_1,\cdots,c_k)$ 之所以 $\in Q$ 皆因能由 R 逐步 [redacted] 生成，那么，只要我们保证生成过程的每一步均不破坏 p 的成立，则可最终保证 $p(c_1,\cdots,c_k)$ 的成立。p'' 所保证的正是这么一件事。

但你也许会问：平时遇到的许多命题貌似长得不像 p' 呀！凡此类命题，又可奈之何？其实不然——一大类命题均可转成 p' 的样式。现在即来探讨这样的转化。

e.g. 原命题 $q :=$「若 $wff(\alpha)$ 则 …」

它等价于 $p' :=$「$\forall \alpha \in WFF$ 有 …」

其中 WFF 是由 wff 判断形式定义的集合。

e.g. 原命题 $q :=$「若 $wff(\alpha)$ 且 $wff(\beta)$ 则…」.

它等价于 $p' :=$「$\forall \alpha \in WFF$ 有 $(wff(\beta) \to \dots)$」

e.g. 原命题 $q :=$「若 $nat(Sx)$ 则 …」

它等价于 $p' :=$「$\forall c \in NAT\ (c = Sx \to \dots)$」

e.g 原命题 $q :=$「若 $add(0,x,y)$ 则 …」

它等价于 $p' :=$「$\forall (a,b,c) \in ADD$ 有 $(a = 0 \to \dots)$」

e.g. 原命题 $q :=$「若 $add(x,y,z)$ 或 $add(x,z,y)$，则 …」

它等价于 $p' :=$「$\forall (a,b,c) \in ADD$ 有 …，且 $\forall (a,c,b) \in ADD$ 有 …」

归结起来，技巧就是：利用逻辑规律，把较长前提切分成两半，一半仍作前提，另一半作为附加前提。

此时，有一个有趣的问题出现了：面对诸如「若A且B则…」的命题，究竟是把A提前，还是把B提前呢？不同的选择将导致不同的归纳步骤——谨记，归纳法总是严格依照大前提的结构而进行的。

e.g. $q :=$「若 $ascend(l)$ 且 $copy(l,l')$ 则 $ascend(l')$」。我们有两种方法切分：

(1) $p' :=$「若 $ascend(l)$，则 $(copy(l,l') \to ascend(l'))$」。这样，就应依照 ascend 的结构施以归纳。

(2) $p' :=$「若 $copy(l,l')$，则 $(ascend(l) \to ascend(l'))$」。这样，就应依照 copy 的结构施以归纳。

两种方法的难度有别。(2) 显然占优，因为「Copy」这一结构已经把 l 与 l' 均确定下来了；反观 ascend，只能确定其一，另一则保持浮动，不太便于处理。

本章最后，我们来对前面所述的理论稍加推广。

(1) ~~递归~~ 归纳定义不仅适用于描述字符串间的关系（即：定义 $Q \subseteq (\Sigma^*)^k$），还适用于定义别的数学对象间的关系。本章一开始讨论过的「递归函数」正是一例。

(2) 在对 ~~归~~ 归纳定义熟悉以后，我们就不必拘泥于 $J(x_1, \dots, x_k)$ 这样的判断形式，而可以大胆采用中缀/后缀表示。比如，此前我们用 $add(x, y, z)$ 这一判断形式来表达「x 与 y 之和是 z」。可是，这显然不如 $x+y=z$ 来得直观。以后我们便可采用后者来书写该判断形式，只是得弄清「$\cdot + \cdot = \cdot$」构成一个整体，不能割开来读。于是

$$\left\{ \frac{}{add(0,0,0)},\ \frac{add(x,y,z)}{add(Sx,y,Sz)},\ \frac{add(x,y,z)}{add(x,Sy,Sz)} \right\}$$

改写作

$$\left\{ \frac{}{0+0=0},\ \frac{x+y=z}{Sx+y=Sz},\ \frac{x+y=z}{x+Sy=Sz} \right\}.$$

(3) 当 $k=1$ 时，我们还可采用 Backus-Naur 记号来书写规则集。例如，

$$\left\{ \frac{}{nat(0)},\ \frac{nat(x)}{nat(Sx)} \right\}$$ 可写作

$$\begin{aligned} n ::= {} & 0 \\ \mid {} & Sn \end{aligned}$$

（类似 CFG 的记法）

（但当 $k>1$ 时，这记号就不太方便了）