类型和证明

rainoftime txyyss marisa
rainoftime@gmail.com txyyss@gmail.com marisa@lolisa.moe

目录

1 简介

定理证明是一种非常重要的形式化技术，广泛用于数学定理证明、协议验证和软硬件安全等领域。无论是 SAT、QBF、SMT 等特定问题的求解器，还是通用的定理证明工具如 Coq、Isabelle，都取得了巨大的理论进展和实践成就。定理证明器和计算代数系统、符号计算系统等也密切相关。

λ 演算是由 Church、Kleene 等人在 20 世纪 30 年代提出的形式系统。λ 演算及其扩展是函数式程序语言的基础，并且广泛用于并发理论、类型论等领域的研究中。

本文包括 λ 演算、逻辑与定理证明的基础知识，重点论述类型论与逻辑逻辑证明的联系，并简单介绍定理证明辅助工具 Coq 的原理和使用。

数理逻辑中对形式系统的研究有两类方法：语义方法和语法方法。语义方法又称模型论方法，研究命题的语义（即命题的意义和真值）、重言式（永真）、语义后承等。语法方法亦称证明论方法，研究形式推演系统、形式定理等。本文主要从语法的角度阐述 λ 表达式类型和逻辑证明的关系，并将介绍它们在语义层面的本质联系。

Part I

基础

2 无类型 λ 演算

本节由 txyyss 和 rainoftime 提供。我们力求严格、形式化地介绍无类型 λ 演算，在这部分把自由变量、α 变换、β 规约等基础概念说明清楚，也方便后续内容更集中在要点上。读者可选择性阅读。

定义 2.1. (λ 项)

假设我们有一个无穷的字符串集合，里面的元素被称为**变量**（和程序语言中变量概念不同，这里就是指字符串本身）。那么 λ 项定义如下：

1. 所有的变量都是 λ 项（名为**原子**）；
2. 若 M 和 N 是 λ 项，那么 $(M\,N)$ 也是 λ 项（名为**应用**）
3. 若 M 是 λ 项而 ϕ 是一个变量，那么 $(\lambda\phi.M)$ 也是 λ 项（名为**抽象**）。

例 2.1. （一些 λ 项） 下面这些都是 λ 项：

$$(\lambda x.(x\,y)) \qquad (x(\lambda x.(\lambda x.x))) \qquad ((((a\,b)\,c)\,d)\,e)$$
$$(((\lambda x.(\lambda y.(y\,x)))\,a)\,b) \qquad ((\lambda y.y)\,(\lambda x.(x\,y))) \qquad (\lambda x.(y\,z))$$

λ 项是一种形式语言，换句话说，就是一类特殊形式的字符串罢，没有任何内在的意义，只是个 “形式”。通常情况下，当讨论一个形式语言的时候，我们需要用另一种元语言来指称形式语言里的元素。就如讨论自然数时，我们经常说 “对任意自然数 n”，这里的 n 本身并不是自然数，用来指称自然数罢了。我们也需要一些 “n” 来表示 λ 项中的元素。因此，我们作如下符号约定：

符号约定 1. 本节中我们用大写英文字母表示任意 λ 项，用除 λ 以外的小写希腊字母如 ϕ，ψ 等表示任意 λ 项中的变量。

对于括号，则有如下的省略规定：

1. λ 项中最外层的括号可以省略，如 $(\lambda x.x)$ 可以省略表示为 $\lambda x.x$；
2. 左结合的应用型的 λ 项，如 $(((M\,N)\,P)\,Q)$，括号可以省略，表示为 $M\,N\,P\,Q$；
3. 抽象型的 λ 项 $(\lambda\phi.M)$ 中，M 最外层的括号可以省略，如 $\lambda x.(y\,z)$ 可以省略为 $\lambda x.y\,z$。

也就是说，我们把省略形式视同定义 2.1 中的 λ 项。

例 2.2. （省略表示） 下面给出了一些省略表示的 λ 项。

省略表示	完整的 λ 项
$\lambda x.\lambda y.y\,x\,a\,b$	$(\lambda x.(\lambda y.(((y\,x)\,a)\,b)))$
$(\lambda x.\lambda y.y\,x)\,a\,b$	$(((\lambda x.(\lambda y.(y\,x)))\,a)\,b)$
$\lambda g.(\lambda x.g\,(x\,x))\,\lambda x.g\,(x\,x)$	$(\lambda g.((\lambda x.(g\,(x\,x)))\,(\lambda x.(g\,(x\,x)))))$
$\lambda x.\lambda y.a\,b\,\lambda z.z$	$(\lambda x.(\lambda y.((a\,b)\,(\lambda z.z))))$

2.1 α 替换

这一小节将形式化的定义什么是 λ 项的替换操作。直观的来讲，就是把 λ 项中不被 $\lambda\phi$ 约束的变量 ϕ 替换成另一个 λ 项罢了。但这么一个操作的精确定义却不是直接和易于理解的。

定义 2.2. （语法全等）

我们用恒等号 "$\equiv$" 表示两个 λ 项完全相同。换句话说

$$M \equiv N$$

表示 M 和 N 有完全相同的结构，且对应位置上的变量也完全相同。这意味着若 $M\,N \equiv P\,Q$ 则 $M \equiv P$ 且 $N \equiv Q$，若 $\lambda\phi.M \equiv \lambda\psi.P$ 则 $\phi \equiv \psi$ 且 $M \equiv P$。

定义 2.3. （自由变量）

对一个 λ 项 P，我们可以定义 P 中*自由变量*的集合 $\mathrm{FV}(P)$ 如下：

1. $\mathrm{FV}(\phi) = \{\phi\}$
2. $\mathrm{FV}(\lambda\phi.M) = \mathrm{FV}(M) \setminus \{\phi\}$
3. $\mathrm{FV}(M\,N) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$

从第 2 可以看出抽象 $\lambda\phi.M$ 中的变量 ϕ 是要从 M 中被排除出自由变量这个集合的。若 M 中有 ϕ，我们可以说它是被*约束*的。据此可以进一步定义*约束变量*集合。值得注意的是，对同一个 λ 项来说，这两个集合的交集未必为空。

例 2.3. （自由变量）

λ 项 P	自由变量集合 $\mathrm{FV}(P)$
$\lambda x.\lambda y.x\,y\,a\,b$	$\{a, b\}$
$a\,b\,c\,d$	$\{a, b, c, d\}$
$x\,y\,\lambda y.\lambda x.x$	$\{x, y\}$

上面最后一个例子里，最左边的 x, y 是自由变量，而最右侧的 x 则是约束变量。若对 λ 项 P 有 $\mathrm{FV}(P) = \emptyset$，则称 P 是 *封闭的*，这样的 P 又称为*组合子*。

定义 2.4. （出现）

对于 λ 项 P 和 Q，可以定义一个二元关系*出现*。我们说 P 出现在 Q 中，是这样定义的：

1. P 出现在 P 中；
2. 若 P 出现在 M 中或 N 中，则 P 出现在 $(M\,N)$ 中；
3. 若 P 出现在 M 中或 $P \equiv \phi$，则 P 出现在 $(\lambda\phi.M)$ 中。

有了上面这些定义，我们终于可以定义什么叫 λ 项的替换操作了：

定义 2.5.（替换）

对任意 M, N, ϕ，定义 $[N/\phi]\,M$ 是把 M 中出现的自由变量 ϕ 替换成 N 后得到的结果，这个替换有可能改变部分约束变量名称以避免冲突。具体精确定义是一个对 M 的归纳定义：

1. $[N/\phi]\,\phi \equiv N$
2. $[N/\phi]\,\alpha \equiv \alpha$ 对所有满足 $\alpha \not\equiv \phi$ 的原子 α
3. $[N/\phi]\,(P\,Q) \equiv ([N/\phi]\,P\,[N/\phi]\,Q)$
4. $[N/\phi]\,(\lambda\phi.P) \equiv \lambda\phi.P$
5. $[N/\phi]\,(\lambda\psi.P) \equiv \lambda\psi.P$ 若 $\phi \notin \mathrm{FV}(P)$
6. $[N/\phi]\,(\lambda\psi.P) \equiv \lambda\psi.[N/\phi]\,P$ 若 $\phi \in \mathrm{FV}(P)$ 且 $\psi \notin \mathrm{FV}(N)$
7. $[N/\phi]\,(\lambda\psi.P) \equiv \lambda\eta.[N/\phi][\eta/\psi]\,P$ 若 $\phi \in \mathrm{FV}(P)$ 且 $\psi \in \mathrm{FV}(N)$

对其中从 5 到 7 的各条来说，$\phi \not\equiv \psi$；而对 7，η 是满足 $\eta \notin \mathrm{FV}(N\,P)$ 的任意变量。

下面解释一下这个看上去很长很恐怖的定义，其实只要带着“替换 λ 项中的自由变量 ϕ 为 N”这个直观去看，就不难理解。前两条无非是说，要是一个原子刚好是要被替换的变量，那就替换，不然就不动。第 3 条也好说，遇到应用了，那就替换各子项。第 4 条，P 中的 ϕ 是被约束的，不能替换所以结果不变。同样的情况发生在第 5 条，P 中没有自由变量 ϕ，结果也不变。顺着这个思路往下想，P 中要是有自由变量 ϕ，是不是就可以替换了呢？是的，这就是第 6 条的内容了。不过这条还有个附加条件，$\psi \notin \mathrm{FV}(N)$，为什么呢？因为 P 是被 $\lambda\psi$ 约束着的，把 N 替换进去的时候，要是 $\mathrm{FV}(N)$ 里面有 ψ，那替换进去的 N 中的 ψ 就从自由变量变成约束变量了。这多少有点让人不放心，所以我们先把这个条件加上，这样替换不会把自由变量变成约束变量，这就是第 6 了。但要是不如意事长八九，N 中偏偏有 ψ 怎么办呢？为了避免这个冲突，干脆把原先约束的变量先换成绝对不会起冲突的吧，找一个 $\eta \notin \mathrm{FV}(N\,P)$ 换掉 ψ，然后再按第 6 条来办，这就是第 7 的内容了。特别注意这一条中，不仅把 P 中的 ψ 换成了 η，连最前面的 $\lambda\psi$ 都被换成了 $\lambda\eta$。

上面的解释只是为了让读者理解一下定义 2.5 的合理性，其实还是有一些没有解释清楚的疑问，比如为什么从自由变量变成约束变量就不好了？第 7 条中 η 有无数种选择，这种不确定性会不会有什么问题？这些问题的答案是：没什么为什么，定义如此。定义不是定理，它说了算，就算有它内蕴的合理性，也并不需要在这个时候证明。

定义 2.5 的第 7 条里，替换的不确定性暗示了可以任意替换约束变量而不改变“意义”。

这个可以类比的理解，例如定积分

$$\int_a^b f(x)\mathrm{d}x$$

里的 x 是所谓的哑变量，换成别的如 $f(t)\mathrm{d}t$ 不会改变积分式的值。又比如正弦函数是写成 $\sin x$ 还是 $\sin t$ 都不影响它的意义。不过这只是我为了能直观理解而加上的一种解释。实际上 λ 项是形式系统，只是形式，一堆字符串罢了，哪里有什么“意义”。

可以说为了补救任意替换约束变量给人带来的不安，我们有了下面的定义：

定义 2.6. **（α 变换和 α 等价）** 设 $\lambda\phi.M$ 出现在一个 λ 项 P 中，且设 $\psi \notin \mathrm{FV}(M)$，那么把 $\lambda\phi.M$ 替换成

$$\lambda\psi.[\psi/\phi]\,M$$

的操作被称为 P 的 α 变换。当且仅当若 P 经过有限步（包括零步）α 变换后，得到新的 λ 项 Q，则我们可以称 P 与 Q 是 α 等价的，又写作

$$P \quad \equiv_\alpha \quad Q$$

例 2.4.

$$\lambda x.\lambda y.x(x\,y) \equiv_\alpha \lambda x.\lambda v.x(x\,v) \equiv_\alpha \lambda u.\lambda v.u(u\,v)$$

容易证明，$\equiv_\alpha$ 满足自反性、对称性和传递性，所以确实是等价关系。有了 α 等价，定义 2.5 就显得更加合理了，第 7 带来的疑问也得到解答了：不是担心任意替换不妥当么？那么我们可以把 α 等价的 λ 项都看成相同的，这不就“补救”了由于不确定替换带来的问题嘛。事实上，有更好的结果：

定理 2.1. *若 $M \equiv_\alpha M'$ 且 $N \equiv_\alpha N'$，则 $[N/x]\,M \equiv_\alpha [N'/x]\,M'$*

强调一下，这个小节里定义的 $[N/\phi]\,M$ 并不是 λ 项这个形式语言里的东西，它只是一种记号，用来指称替换后的 λ 项而已，切记切记。而我各种解释说明中的意义和类比都只是为了方便直观的理解，并不是说 λ 项有这些“意义”。它始终只是“名”，和“实”不相关。作来体现。

2.2 β 规约

现在我们有了 λ 项，有了替换的规则，那么什么时候需要进行替换呢？这就是本小节要讨论的话题：规约。有了规约，整个关于 λ 项的形式系统才算完整，才可以说是 λ 演算。

先介绍 β 规约，β 规约可以这么直观的理解：我们可以把 $(\lambda\phi.M)$ 看成是参数为 ϕ，函数体为 M 的一个函数；把 $(M\,N)$ 看成是函数 M 作用到实际参数 N 上[1]。平时我们要是定义了函数 $f(x) = x + 5$，那么函数应用 $f(6)$ 就是把 $x + 5$ 中的 x 替换成 6，得到 $f(6) = 6 + 5 = 11$。替换是函数应用的实质啊。

[1]还是再强调一下，这只是直观的理解，并不是说两种 λ 项一个是函数，一个是函数应用。λ 项不是这些意义，只是字符串。

定义 2.7. （β **规约**）形如

$$(\lambda\phi.M)\,N$$

的 λ 项被称为 β **可约式**，对应的项

$$[N/\phi]\,M$$

则称为 β **缩减项**。当 P 中含有 $(\lambda\phi.M)\,N$ 时，我们可以把 P 中的 $(\lambda\phi.M)\,N$ 整体替换成 $[N/\phi]\,M$，用 R 指称替换后的得到的项，那么我们说 P 被 β 缩减为 R，写做：

$$P \rhd_{1\beta} R$$

当 P 经过有限步（包括零步）的 β 缩减后得到 Q，则称 P 被 β **规约**到 Q，写做：

$$P \rhd_{\beta} Q$$

例 2.5. （β **缩减**）

$$\begin{array}{rlll}
(\lambda x.x\,(x\,y))\,m & \rhd_{1\beta} & m(m\,y) & \\
(\lambda x.y)\,n & \rhd_{1\beta} & y & \\
(\lambda x.(\lambda y.y\,x)z)\,v & \rhd_{1\beta} & [v/x]\,((\lambda y.y\,x)z) & \equiv \quad (\lambda y.y\,v)z \\
 & \rhd_{1\beta} & [z/y]\,(y\,v) & \equiv \quad z\,v \\
(\lambda x.x\,x)\,(\lambda x.x\,x) & \rhd_{1\beta} & [(\lambda x.x\,x)/x]\,(x\,x) & \equiv \quad (\lambda x.x\,x)\,(\lambda x.x\,x) \\
 & \rhd_{1\beta} & [(\lambda x.x\,x)/x]\,(x\,x) & \equiv \quad (\lambda x.x\,x)\,(\lambda x.x\,x) \\
 & \cdots & \text{etc.} & \\
(\lambda x.x\,x\,y)\,(\lambda x.x\,x\,y) & \rhd_{1\beta} & (\lambda x.x\,x\,y)\,(\lambda x.x\,x\,y)\,y & \\
 & \rhd_{1\beta} & (\lambda x.x\,x\,y)\,(\lambda x.x\,x\,y)\,y\,y & \\
 & \cdots & \text{etc.} &
\end{array}$$

从上面的例子可以看出，β 缩减虽然名为缩减，但实际情况是复杂的。像最后这两个例子，一个永远缩减为自身，一个甚至更糟糕，越来越长，非但不是缩减反而是增长了。

定义 2.8. （β **范式**）

若一个 λ 项 Q 不含有 β 可约式，则称 Q 为 β **范式**。若有 P 可被 β 规约到 Q，则称 Q 是 P 的 β 范式。

从示例 2.5 可以看出，并不是所有的 λ 项都能 β 规约到 β 范式。这其实还不是最让人担心的，有一个之前读者也许没注意到的问题：定义 2.7 里关于 β 规约的定义非常“狡猾”，或者叫非常含糊，它只要求存在一个有限长的 β 缩减链就行，当 λ 项中含有不止一个 β 可约式的时候，可以有不同顺序的缩减方式。

例 2.6. 让我们重新看一下示例 2.5 里的 $(\lambda x.(\lambda y.y\,x)z)\,v$，最外层的 $(\lambda\ldots)\,v$ 是一个 β 可约式，里层的 $(\lambda y.y\,x)\,z$ 也是一个 β 可约式。那么就有两种缩减方式：

$$\begin{array}{lll} (\lambda x.(\lambda y.y\,x)z)\,v & \triangleright_{1\beta} & (\lambda y.y\,v)\,z \quad \text{缩减 } (\lambda x.(\lambda y.y\,x)z)\,v \\ & \triangleright_{1\beta} & z\,v \qquad\qquad\quad \text{缩减 } (\lambda y.y\,v)\,z \\ (\lambda x.(\lambda y.y\,x)z)\,v & \triangleright_{1\beta} & (\lambda x.z\,x)\,v \quad \text{缩减 } (\lambda y.y\,x)\,z \\ & \triangleright_{1\beta} & z\,v \end{array}$$

从示例 2.6 可以看到，虽然缩减顺序不一样，但最后得到了同样的结果。这个是不是对所有的 λ 项都成立呢？要是不成立，也就说顺序会影响规约结果，就不好了。这个问题的答案可以说既让人满意也让人不满意。

定理 2.2. **（*Church–Rosser* 定理）**

若 $P \triangleright_\beta M$ 且 $P \triangleright_\beta N$，则存在一个 λ 项 T 使得

$$M \triangleright_\beta T \quad \text{且} \quad N \triangleright_\beta T.$$

Church–Rosser 定理最重要的一个应用就是可以用来证明如下重要结果：

定理 2.3. *若 P 有 β 范式，则该范式在模 $\equiv_\alpha$ 的意义下唯一；也就是说若 P 有 β 范式 M 和 N，则 $M \equiv_\alpha N$。*

有了这个定理，我们可以放一半的心了：如果你对一个 λ 项做 β 规约，最后得到个范式，可以确定这个范式某种程度上是唯一的，不用担心你因为规约顺序上的随意性导致这个范式和别人不一样。但为啥说是放一半的心呢？*因为，这个定理并没有说：若 P 有 β 范式，那么你随便按任意顺序规约，都能得到 β 范式。*

例 2.7. 让我们看下面有 β 范式的 λ 项的例子：

$$\begin{array}{rcl} (\lambda x.\lambda y.x)\,a\,((\lambda x.x\,x)\lambda x.x\,x) & \triangleright_{1\beta} & ([a/x]\,(\lambda y.x))\,((\lambda x.x\,x)\lambda x.x\,x) \;\equiv\; (\lambda y.a)\,((\lambda x.x\,x)\lambda x.x\,x) \\ & \triangleright_{1\beta} & [((\lambda x.x\,x)\lambda x.x\,x)/y]\,a \qquad\qquad \equiv\; a \end{array}$$

这个 λ 项其实包含三项，我们要是先归约前两项 $(\lambda\ldots)$,a 就能得到一个 β 范式。但注意第三项也是一个 β 可约式，完全可以先对它进行规约。但如我们在示例 2.5 里的倒数第二项看到的，$((\lambda x.x\,x)\lambda x.x\,x)$ 会不断规约到自身，所以要是执着于把它先规约了，就永远得不到 β 范式了。

于是我们就要问了，若 P 有 β 范式，怎么才能规约得到它？按任意顺序规约就有可能出现示例 2.7 中的问题。可要是全部穷举，含有 n 个 β 可约式的话就有 $n!$ 种规约顺序，一一列举也太费事了吧。万幸，1958 年 Curry 证明了这么一个定理：

定理 2.4. *对 P 的总是先 β 缩减最左侧最外侧的 β 可约式，若这个过程能无限进行下去，那么对 P 的所有任意顺序的规约都能无穷进行下去。*

换而言之，要是 P 有 β 范式，那么这种最左最外侧优先的规约方式总能保证得到那个 β 范式，这种规约顺序又称为**正则序**。我的解释器实现的规约顺序就是正则序，虽然它规约需要的步数可能较多，但抵消不了这个巨大的优势啊。

到目前为止，我们得到的结果都是满意的：一个 λ 项若是有 β 范式，那么这个 β 范式某种程度上是唯一的，而且我们有确定的规约顺序保证得到这个范式。下面就只有一个问题了：能否有一个通用算法，判定一个 λ 项是否有 β 范式？非常不幸，答案是否定的：

定理 2.5. *λ 项是否有 β 范式是不可判定的。*

注.

到这里我们算是粗粗的把 λ 演算的主要内容：β 规约介绍了一下。还有个 η 规约有兴趣的读者可以自行查找相关资料，这里就不多做介绍了。

3 经典命题逻辑

本节内容主要参考《数理逻辑基础》（赵希顺）和《模型论导引》（沈复兴）。

笔者注意到离散数学等计算机系课程，对数理逻辑的讲解往往不如数学/哲学系中逻辑学课程系统、严谨。本节通过经典命题逻辑，介绍数理逻辑中的基本概念，最后引入 Gentzen 的自然演绎（Natural deduction），为后续的直觉主义逻辑证明系统作基础。读者可选择性阅读此节。

3.1 命题逻辑的语法

命题逻辑的一个**形式系统**是由 (1) 形式符号，(2) 由这些符号形成合式公式（简称公式）的规则，(3) 由若干公式组成的公理，(4) 由公理经一定推演法则得到的定理所组成。以 L 记我们所选用的命题逻辑形式系统。

L 语法由由形式符号和形成规则（亦称语法规则）组成。L 的形式符号有以下三种：

1. **原子命题符** $p_1, p_2, ..., p_n$，又叫命题变元符号
2. **逻辑连接词** $\neg$（否定），$\wedge$（合取），$\vee$（析取），$\rightarrow$（蕴涵）
3. **括号**，(，)

L 的命题变元符号有可数多个，每个命题变元符号代表任意一个可以回答是或者非的命题。L 的符号组成符号串，满足如下形成规则的有限符号串称为 L 的合式公式：

定义 3.1.（**合式公式**，良构式，简称公式）

命题逻辑形式系统的公式可递归定义为：

1. 每一原子命题/命题变元都是公式
2. 如果 A 是公式，则 $(\neg A)$ 也是公式
3. 如果 A, B 是公式，则 $(A \wedge B), (A \vee B), (A \to B)$ 也是公式
4. 每一公式都是通过 (1)-(3) 经有限步得到

注. 上面的定义中，A, B 不是形式系统 L 中的符号，我们只是用 A, B, C 等来表示 L 中的任意公式。如果称 L 中的符号为 L 的语言，则 L 中形成公式的规则叫作语法规则。而我们用来说明 L 所用的语言叫**元语言**，比如 A, B 是元语言的符号。读者要多加注意、思考。

在很多文献中，连接词 $\vee, \wedge$ 在 L 中不出现，因为 $\{\neg, \to\}$ 已经构成连接词的完全集，可以用它们来定义其他连接词。设 A, B 是 L 中的公式，则 $(A \wedge B)$ 可定义为 $(\neg(A \to \neg B)$，$(A \vee B)$ 定义为 $((\neg A) \to B)$。为了后面叙述方便，我们重新定义合式公式，

定义 3.2. （**合式公式**，简称公式）
命题逻辑形式系统的公式可递归定义为：

1. 每一原子命题/命题变元都是公式
2. 如果 A 是公式，则 $(\neg A)$ 也是公式
3. 如果 A, B 是公式，则 $(A \to B)$ 也是公式
4. 每一公式都是通过 (1)-(3) 经有限步得到

为了使公式的写法更简单而不至引起歧义，有时省略最外层的括号，并且约定：

1. $A \to B \to C$ 是公式 $(A \to (B \to C))$ 的简写，
2. $\neg A \to B$ 是 $((\neg A) \to B)$ 的简写，
3. 连接词的优先级按递减顺序依次为 $\neg, \wedge, \vee, \to$

我们约定形式系统 L 只有可数多个命题变元符号，因此 L 中最多有可数多个有限长的符号串，这样 L 中最多有可数多个公式。

3.2 命题逻辑的语义

3.2.1 赋值、真值表

“形式语言的符号本身是无意义的，但是可以赋予它意义”。例如，我们把原子命题符号（或者命题变元符号）p_i 解释为一个命题，它非真即假（本节统一用 1 表示“真”，0 表示“假”），而把逻辑连接词 $\neg, \wedge, \vee, \to$ 分别解释为“非”、“并且”、“或者”“蕴涵”

“A 并且 B” 的真假依赖于 A 和 B 的真假，“并且” 可以看作 $\{0,1\} \times \{0,1\}$ 到 $\{0,1\}$ 上的函数。我们用下表给出逻辑连接词的涵义

注. 虽然称 $\to$ 为“蕴涵”，但它和日常语言中的用法不同。日常语言的“蕴涵” 是“因果蕴涵”，比如“如果他考试不及格，他就不会被录” 这句话是有意义的，但“如果兔子会飞，则

表 1:

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \to B$
1	1	0	1	1	1
1	0	0	0	1	0
0	1	1	0	1	1
0	0	1	0	0	1

月亮是三角形的" 这句话就没有意义，因为兔子会飞和月亮的形状没有因果关系。然而在这里，如果 A 是假的，那么不管 B 是真还是假，"$A \to B$" 总为真。这种 "蕴涵" 称为 "实质蕴涵"。

定理 3.1. *一旦原子命题符号的取值确定，则每一公式的真值也就唯一确定。*

注. 本文的论述并不严密，也略去很多定理的证明过程。如果兴趣，读者可参考专业书籍。

定义 3.3. （**可满足**）

设 A 为一公式，$v_i, v_2, ...$ 为对 A 中原子命题的赋值，若存在一组赋值 v_i 使得 $A = 1$，则 A 是可满足的。

注记. 判断命题逻辑公式是否可满足，叫 "布尔可满足问题（SAT）"，是第一个发现的 NP-完全问题。SAT 求解技术多年来已经取得重大突破，并在软硬件验证中取得广泛应用。

定义 3.4. （**永真式**，重言式，逻辑有效）

如果任意赋值都满足 A，则 A 是永真的。

定理 3.2. *如果 A 是永真的，则 $\neg A$ 是不可满足的。*

注. 以上定理是一些证明逻辑有效性的算法的基础，如 Stalmarck 算法。

定义 3.5. 设 Γ 是以公式集，如果 Γ 的每个有穷子集都是可满足的，则 Γ 是有穷可满足的。

定理 3.3. *（紧致性定理）设 Γ 是一公式集，则 Γ 是可满足的当且仅当 Γ 是有穷可满足的*

注. 这个定理有很大的实用意义，比如在动态符号执行工具 Klee 中，利用了它进行 SAT 求解优化。

定义 3.6. （**语义后承**）

设 Γ 为一公式集，v 是一个赋值。

1. 如果 v 满足 Γ 的每个公式，则称 v 满足 Γ
2. 设 A 是一个公式，如果满足 Γ 的每个赋值都满足 A，则称 A 是 Γ 的语义后承，记作 $\Gamma \models A$

注记. 如果 Γ 是空集，我们把 $\emptyset \models A$ 简记作 $\models A$，显然，$\models A$ 当且仅当 A 是永真的；通常把 $\Gamma \cup \{B\} \models A$ 简记作 $\Gamma, B \models A$

3.2.2 模型论描述 *

命题逻辑形式系统 L 中的全体命题变元符号为 $p_1, p_2, ...$，L 的一个**模型** $\mathcal{M}$ 是全体命题变元组成集合的一个**子集**，即 $\mathcal{M} \subset \{p_1, p_2, ...\}$。$\mathcal{M}$ 可以是空集或者 $\{p_1, p_2, ...\}$ 自身。

L 中任意公式在某个模型中都有一个**赋值**："真" 或者 "假"。公式 A 如果在某个模型 $\mathcal{M}$ 中**取值**为真，记作 $\mathcal{M} \models A$。

定义 3.7. L 的公式 A 在模型 $\mathcal{M}$ 中的**取值**归纳定义如下：

1. 若 A 是某个命题变元 p_i，则 $\mathcal{M} \models A$ 当且仅当 $p_i \in \mathcal{M}$
2. 若 A 是 $\neg B$，则 $\mathcal{M} \models A$ 当且仅当 $\mathcal{M} \nvDash B$，即 B 在 $\mathcal{M}$ 中的取值不为真
3. 若 A 是 $B \to C$，则 $\mathcal{M} \models A$ 当且仅当 $\mathcal{M} \nvDash B$ 或 $\mathcal{M} \models C$

定义 3.8. （**可满足**）

公式 A 是可满足的，如果它至少存在一个模型。

定义 3.9. （**永真式**，重言式，逻辑有效，Tautology）

公式 A 是永真的，如果它的每个语义解释都是一个模型

定义 3.10. （**语义后承**）

如果 A 的每一个模型都是 B 的模型，则称 B 是 A 的一个语义后承，记作 $A \models B$。换句话说：如果在任何一种语义赋值下，只要命题 A 为真，B 一定为真，那么 B 是 A 的语义后承。

语义后承的概念也可以用在命题集合和单个命题之间。如果在任何一种语义赋值下，只要命题集合 Σ 中的每一个命题都为真，那么 A 就一定为真，我们就说 A 是 Σ 的语义后承，记为 $\Sigma \models A$

3.3 公理推演系统

设 A, B, C 代表任意公式。由于经典命题逻辑中逻辑连接词可以互定义，下面我们只用 $\to$ 和 $\neg$。

定义 3.11. （**Hilbert 演绎系统**）

Hilbert 系统由三个公理模式加一个推演规则（MP）组成：

公理模式

1. $(A \to (B \to A))$
2. $((A \to (B \to C)) \to ((A \to B) \to (A \to C)))$
3. $(((\neg A) \to (\neg B)) \to (B \to A))$

分离规则 (MP)

$$\frac{A, A \to B}{B}$$

注. 由于 A, B, C 代表任意公式，上面的每一条都表示无穷多个公理。

定义 3.12. (**形式证明**) 称公式的有穷序列

$$A_1, A_2, ..., A_n$$

是一个形式证明（简称证明），如果对任意 $i, 1 \leq i \leq n$，都有下列之一成立:

1. A_i 是公理，或
2. 存在 $j_1, j_2 < i$ 使得 A_i 是 A_{j_1} 和 A_{j_2} 通过分离规则推出的（或者说，是关于分离规则的直接后承）

一个形式证明的长度是该证明所含公式的个数。

定义 3.13. (**形式定理**)

称公式 A 是一个形式定理（简称定理），如果存在一个形式证明 $A_1, ..., A_n$ 使得 $A = A_n$。此时也称 A 是形式可证的。

定义 3.14. (**形式推演**)

设 Γ 是由一些公式组成的集合，称公式序列

$$A_1, A_2, ..., A_n$$

是一个由 Γ 中公式出发的形式推演，如果对任意 $i, 1 \leq i \leq n$，都有下列之一成立:

1. $A_i \in \Gamma$,
2. A_i 是一公理;
3. 存在 $j_1, j_2 < i$ 使得 A_i 是 A_{j_1} 和 A_{j_2} 关于分离规则 MP 的直接后承。

如果存在由 Γ 出发的推演使得 A 是该推演的最后一个公式，则称 A 可由 Γ 推出（或者说，Γ 推出 A），记为 $\Gamma \vdash A$。常把 Γ 中断公式称为假定公式。

注记.
1. 如果 Γ 是空集，即 $\Gamma = \emptyset$，则我们把 $\emptyset \vdash A$ 记为 $\vdash A$。显然，$\vdash A$ 当且仅当 A 是形式定理。
2. 假设 $\Gamma = B_1, ..., B_k$，我们常把 $\Gamma \vdash A$ 写成 $B_1, ..., B_k \vdash A$
3. 我们把 $\Gamma \cup B \vdash A$ 记为 $\Gamma, B \vdash A$

定理 3.4. （**一致性定理**）

命题演算的形式定理都是永真的。

例 3.1. 证明 $\emptyset \vdash p \to p$，即 $p \to p$ 在系统内可证，$p \to p$ 是定理。

我们给出如下命题序列（证明）：

1. $P \to ((P \to P) \to P)$ （公理 1）
2. $(P \to ((P \to P) \to P)) \to ((P \to (P \to P)) \to (P \to P))$ （公理 2）
3. $((P \to (P \to P)) \to (P \to P))$ （1，2，MP 规则）
4. $P \to (P \to P)$ （公理 1）
5. $P \to P$，（4，3，MP 规则）

3.4 自然演绎系统

Hilbert 系统存在很多缺点：证明过程冗长、不符合实际推理的直觉、不利于研究一致性等问题。1935 年，Gentzen 提出了自然演绎 (Natural deduction) 和相继式演算 (Sequent Calculus)，对数理逻辑尤其是证明论产生了及其深远的影响。先介绍自然演绎系统。

自然演绎系统有以下特点：

1. 每个逻辑连接词对应引入规则和消去规则，推演的过程基于 “假设”(Assumption) 和逻辑连接词对应的规则。
2. 在推演中，根据一定规则，假设可以被 “导入”(introduction, opening) 和 “消耗”(cancelation, closing, discharge)，在被消耗之前该假设可以使用多次
3. 在推演的最后，所有开放 (未被消耗) 的假设必须被声明，剩下的假设越少，则结论也可靠

下面给出经典命题逻辑的自然演绎证明系统。

$$A, B ::= A \to B \mid A \wedge B \mid A \vee B \mid \neg A \mid \bot$$

$\wedge$ **引入**（合取引入）：如果 A 和 B 是可证的，那么 $A \wedge B$ 是可证的。

$$(\wedge I) \frac{A \qquad B}{A \wedge B}$$

$\wedge$ **消去**（合取消去）：如果 $A \wedge B$ 是可证的，那么 A 和 B 都是可证的。

$$(\wedge E) \frac{A \wedge B}{A}$$

$$(\wedge E) \frac{A \wedge B}{B}$$

$\vee$ **引入**（析取引入）：如果 A 或 B 是可证的，那么 $A \vee B$ 是可证的。

$$(\vee I) \frac{A}{A \vee B}$$

$$(\vee I)\ \frac{B}{A \vee B}$$

$\vee$ **消去**（析取消去）：如果 $A \vee B$ 可证，且基于假设 A 和 B 都有结论 C，则可以 C 可证

$$(\vee E)\ \frac{A \vee B \qquad \begin{matrix}[A] \\ C\end{matrix} \qquad \begin{matrix}[B] \\ C\end{matrix}}{C}$$

$\rightarrow$ **引入**（蕴涵引入）：如果基于假设 A 有结论 B，则可得 $A \rightarrow B$

$$(\rightarrow I)\ \frac{\begin{matrix}[A] \\ B\end{matrix}}{A \rightarrow B}$$

$\rightarrow$ **消去**（蕴涵消去）：如果 $A \rightarrow B$ 和 A 是可证的，则 B 是可证的。等价于 Hilbert 系统的 MP 规则。

$$(\rightarrow E)\ \frac{A \rightarrow B \qquad A}{B}$$

$\neg$ **引入**（否定引入）

$$(\neg I)\ \frac{\begin{matrix}[A] \\ \bot\end{matrix}}{\neg A}$$

$\neg$ **消去**（否定消去）

$$(\neg E)\ \frac{A \qquad \neg A}{\bot}$$

Ex falsum quodlibet：从矛盾可以导出任何事物

$$(\bot)\ \frac{\bot}{A}$$

Reductio ad absurdum

$$RA\ \frac{\begin{matrix}[\neg A] \\ \bot\end{matrix}}{A}$$

例 3.2. 在自然演绎系统中推演。（在推演步骤的左边标上了所使用的规则）

$$A \wedge B \vdash_{\mathrm{ND}} B \wedge A$$

$$\dfrac{\dfrac{A \wedge B}{B}(\wedge E) \quad \dfrac{A \wedge B}{A}(\wedge E)}{B \wedge A}(\wedge I)$$

例 3.3. 再看这个例子

$$\vdash_{\mathrm{ND}} A \to \neg\neg A$$

我们给假设标了号 “1”, “2”, 并且如果推演过程中消耗了某假设, 则在右边写上该假设的标号。

$$\dfrac{\dfrac{\dfrac{[A]^2 \qquad [\neg A]^1}{\bot}(\neg I)}{\neg\neg A}1}{A \to \neg\neg A}(\to I)\,2$$

练习 3.1. (1) $A \to B \vdash_{\mathrm{ND}} \neg B \to \neg A$

(2) $A \vee B \vdash_{\mathrm{ND}} \neg A \to B$

(3) $\vdash_{\mathrm{ND}} A \vee negA$

3.5 相继式演算

3.5.1 简介

相继式演算与自然演绎的区别主要在于“基于某假设有某结论”的处理。在自然演绎中, 这一个假设是放在逻辑证明的部分的顶部, 与某一规则相对应（可参见前文对自然演绎系统规则的描述）。而在相继式演算中, 这一假设作为条件在每一个相继式中直接表示出来。这样做的好处是使用起来更加直观。另外, 这样得到的演绎规则更好地体现了逻辑连接词的对偶关系。

3.5.2 相继式

每个相继式具有如下形式:

$$A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots B_m$$

其中, A_i、B_i 均为公式。A_i 称为条件, 而 B_i 称为结论。每个条件、每个结论的顺序不区分。特别的, $\vdash$ 的两侧都可以没有任何公式。当没有结论时, 右侧结论无条件成立; 当没有结论时, 相继式在从条件可推出矛盾时成立。

3.5.3 演绎规则

要证明一个相继式，就是从左侧的所有条件都成立的情况下，推导出右侧所有结论中的任意一个成立。这样做看起来有点不符合直观印象，但是后面我们就会发现它的妙处。在相继式演算中演绎规则的使用与自然演绎类似，也是由规则组成，但是不存在在某一子证明过程中多出的假设这一概念，而且引入、消去规则的区分也变成了左、右逻辑规则，分别对应处理出现在 $\vdash$ 左侧、右侧的逻辑连接词。

下面给出经典命题逻辑的相继式演算证明规则。命题逻辑公式的语法如下：

$$P ::= P \to P \mid P \lor P \mid P \land P \mid \neg P \mid \bot$$

符号的使用如下：

1. A 和 B 是任意命题逻辑的公式
2. Γ、Δ 是任意公式序列，代表任意多个公式（包括 0 个）

初始相继式 (Initial sequent)：以任何公式为条件，可以证明它自身。初始相继式体现了“由左侧所有条件推出右侧任意结论”这一点。为了方便知道是用了哪个公式，我们对 I 做标记。

$$(I_A)\ \frac{}{\Gamma, A \vdash \Delta, A}$$

逻辑割 (Cut)：从条件证明的结论可以用做条件。

$$(\text{Cut})\ \frac{\Gamma \vdash \Delta, A \qquad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$$

以下规则的中文名称不再赘述，符号名称均为逻辑连接词 + L/R 的形式，分别代表这个逻辑连接词的左/右规则。

$$(\lor\text{L})\ \frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \lor B \vdash \Delta} \qquad (\land\text{R})\ \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \land B, \Delta}$$

$$(\land\text{L})\ \frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \land B \vdash \Delta} \qquad (\lor\text{R})\ \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \lor B, \Delta}$$

仔细观察这两条规则，可以发现，逻辑连接词 $\land$ 和 $\lor$ 的对偶关系在相继式演算中很好地体现了出来 —— 这四条规则非常对称。

$$(\neg\text{L})\ \frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta} \qquad (\neg\text{R})\ \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}$$

$$(\to\mathrm{R})\ \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta} \qquad\qquad (\to\mathrm{L})\ \frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}$$

$$(\bot\mathrm{L})\ \frac{}{\Gamma, \bot \vdash \Delta}$$

剩余的是逻辑蕴含和否定的规则，以及假命题可以推出任意命题 (Ex falso quodlibet)。这里没有给出 ¬R，是因为它是没有用的：在 $\vdash$ 右侧去掉不可能证明的结论，对整个相继式能否证明无影响。

3.5.4 相继式在命题逻辑中的简单解释

一个相继式 $A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$ 大致相当于命题逻辑里的 $(A_1 \wedge \cdots \wedge A_n) \to (B_1 \vee \cdots \vee B_m)$。在经典逻辑下，这又相当于 $\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$ 或者 $\neg(A_1 \wedge \cdots \wedge A_n \wedge \neg B_1 \wedge \cdots \wedge \neg B_m)$。由 De Morgan 定律 $\neg(X \vee Y) \leftrightarrow \neg X \wedge \neg Y$ 和 $\neg(X \wedge Y) \leftrightarrow \neg X \vee \neg Y$，我们就大致可以体会到逻辑连接词的对偶性在相继式演算中出现的深层内涵。

3.5.5 相继式演算中的证明过程

注意到除了 I 和 Cut 以外，其它的规则均使相继式中减少了一个逻辑连接词，而遇到可以使用 I 的地方，证明就不需要继续向上了。而且，Gentzen 证明了我们可以在不失去证明能力的情况下把 Cut 规则去掉 (Cut-elimination)。所以，命题逻辑中，相继式演算中的证明就变得非常简单 —— 只需要把每个逻辑连接词都消去，然后在剩下的无连接词的相继式中找到匹配的公式，使用规则 I 就可以了。以下举数例：

$$\cfrac{\cfrac{(I_A)\ \cfrac{}{A \vdash A}}{\vdash A, \neg A}(\neg\mathrm{R})}{\vdash A \vee \neg A}(\vee\mathrm{R})\ (\text{排中律})$$

$$\cfrac{\cfrac{\cfrac{(I_P)\ \cfrac{}{P \vdash Q, P}}{\vdash P \to Q, P}(\to\mathrm{R}) \qquad (I_P)\ \cfrac{}{P \vdash P}}{(P \to Q) \to P \vdash P}(\to\mathrm{L})}{\vdash ((P \to Q) \to P) \to P}(\to\mathrm{R})\ (\text{Peirce 定律})$$

3.6 回顾与小结

数理逻辑中对形式系统的研究有两类方法：语义方法和语法方法。语义方法又称模型论方法，研究命题的语义（即命题的意义和真值）、重言式（永真）、语义后承等。语法方法亦称证明论方法，研究形式推演系统、形式定理等。另一种说法是，对逻辑系统的研究有两种传统，语义传统和语法传统。

例如，实质蕴涵符号 $\to$ 可以在句法系统中由 Hilbert 的前两条公理完全刻画（第三条公理刻画它和否定 $\neg$ 的关系）。而语义系统中，我们说：$\Sigma \models P \to Q$ 当前仅当 $\Sigma \models Q$ 或者 $\Sigma \models \neg P$。换句话说：如果一个实质蕴涵条件句成立，就是说，前件（P）为真的情况下，后件（Q）不可能为假。

再来比较语义后承和语法（句法）后承。考虑它们连接一个命题集合和一个命题的情况。语义后承（semantic consequence），用 $\models$ 表示，如果在任何一种语义赋值下，只要命题集合 Σ 中的每一个命题都为真，那么 A 就一定为真，我们就说 A 是 Σ 的语义后承，记为 $\Sigma \models A$。句法后承（syntactic consequence），用 $\vdash$ 表示。$\Sigma \vdash A$ 表示 A 可以通过句法证明的方式从命题集 Σ 中得出。即存在一个命题序列，其中每个命题要么是公理，要么是前提，要么由前面的命题通过证明规则得到，且最后一个是 A。该命题序列称为一个证明。

回到之前的例子，$\emptyset \vdash P \to P$，句法证明过程如下：

1. $P \to ((P \to P) \to P)$ （公理 1）
2. $(P \to ((P \to P) \to P)) \to ((P \to (P \to P)) \to (P \to P))$ （公理 2）
3. $((P \to (P \to P)) \to (P \to P))$ （1，2，MP 规则）
4. $P \to (P \to P)$ （公理 1）
5. $P \to P$，（4，3，MP 规则）

在语义系统中，如果要说明 $\emptyset \models P \to P$(即，是空集的语义后承，或者说，是永真的)，只需要说明，由于在 P 为真和 P 为假的情况下，根据实质蕴含算子的语义规则（参考上述），我们都能得到 $P \to P$ 为真。因此，我们说这个公式是空集的语义后承（也就是永真）。

读者可能会发现，一些讲解逻辑的文献并没有严格区分 $\vdash$ 和 $\models$，甚至只用 $\Rightarrow$ 等符号表示 “推演” 之类的概念。这在经典逻辑中没有大问题，下面介绍可靠性和完备性的概念。

定义 3.15. （**可靠性**，Soundness）

可靠性是指：如果 $\vdash P$，那么 $\models P$，即凡是可证的都是为为真的

定义 3.16. （**完备性**，Compleness）

完备性是指：如果 $\models P$，那么 $\vdash P$，即凡是为真的公式都是可证的；

定理 3.5. （命题逻辑的可靠性和完备性）

命题逻辑是可靠且完备的。

注. 有的文献可能会提强、弱完备性、可靠性。强完备性是指：如果 $\Sigma \models P$，那么 $\Sigma \vdash P$，强可靠性是指：如果 $\Sigma \vdash P$，那么 $\Sigma \models P$，弱可靠、完备性与上面的可靠、完备性相同。

由于篇幅有限，我们不在此证明。

TODO: 加入 CH 同构下与 delimited/undelimited continuation 的关系，可见 lectures note on curry howard isomorphism. 如果你不想写的话可以给我来写

4 范畴论基础

范畴论产生于上世纪 40 年代对同调代数的研究。经过数十年的发展，范畴论已经成为具有广泛意义的数学理论，在代数学、拓扑学、数理逻辑等领域有深刻的应用。理论计算机科学中，范畴论也广泛用于程序指称语义、程序逻辑、类型理论等领域的研究中。

4.1 范畴

定义 4.1. （**范畴**，category）

一个范畴 $\mathcal{C}$ 包括

1. $\mathrm{Ob}(\mathcal{C})$：一些对象（**Object**）组成的集
2. $\mathrm{Ar}(\mathcal{C})$：对象之间的态射/箭头（**Morphism/Arrow**）组成的集

函数 **dom** 与 **cod**：$\mathrm{Ar}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{C})$，分别表示一个态射的值域与定义域一个从值域 A 到定义域 B 的态射 f 表示为 $f : A \to B$，值域 A 与定义域 B 之间的态射集表示为 $\mathcal{C}(A, B)$，称为 **hom-set**

3. 对任意三个对象 A, B, C，态射复合

$$\mathcal{C}_{A,B,C} : \mathcal{C}(A, B) \times \mathcal{C}(B, C) \to \mathcal{C}(A, C)$$

态射复合 $\mathcal{C}_{A,B,C}(f, g)$ 记为 $g \circ f$ 或 $f\ ;\ g$，图像化表示为 $A \xrightarrow{f} B \xrightarrow{g} C$

4. 对任意一个对象，存在单位态射 $\mathrm{id}_A : A \to A$
5. 对于以上定义，对任意态射 $f : A \to B$，$g : B \to C$，$h : C \to D$，符合公理：

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

$$f \circ id_A = f = id_B \circ f$$

4.2 函子

定义 4.2. （**函子**，functor）函子可以理解为范畴之间的态射，范畴 $\mathcal{C}$ 与 $\mathcal{D}$ 之间的函子 $F : \mathcal{C} \to \mathcal{D}$ 定义为：

- 对象之间的映射：对于 $\mathcal{C}$ 内的每一个对象 A，将其联系至 $\mathcal{D}$ 内的对象 $F\ A$
- 态射之间的映射：对于 $\mathcal{C}$ 内的每一个态射 $f : A \to B$，将其联系至 $\mathcal{D}$ 内的态射 $F\ f : F\ A \to F\ B$，使得结合律与单位态射得到保留：

$$F\ (g \circ f) = F\ g \circ F\ f$$

$$F\ id_A = id_{F\ A}$$

4.3 伴随性

定义 4.3. (**自然性**, naturality) 更进一步地，我们可以定义，自然变化（natural transformation）为函子之间的 “态射”。若 $F, G : \mathcal{C} \to \mathcal{D}$ 为范畴 $\mathcal{C}$ 与范畴 $\mathcal{D}$ 之间的函子，F 与 G 之间的自然变化 $t : F \to G$ 为一个由范畴 $\mathcal{C}$ 中任意对象 A 所 index 的态射族：$\{t_A : F\ A \to G\ A\}_{A \in Ob(\mathcal{C})}$。这个态射族满足：对于范畴 $\mathcal{C}$ 中的所有态射 $f : A \to B$，使得以下图交换：

$$\begin{array}{ccc} F\ A & \xrightarrow{F\ f} & F\ B \\ \downarrow{\scriptstyle t_a} & & \downarrow{\scriptstyle t_B} \\ G\ A & \xrightarrow{G\ f} & G\ B \end{array}$$

若对于所有的 $A \in Ob(\mathcal{C})$，t_A 皆为同构，那么 t 就被称为一个自然同构 (natural isomorphism)。

$$t : F \xrightarrow{\cong} G$$

伴随性可以理解为 Galois Connection 在范畴论之下的抽象。

4.3.1 Galois Connection

定义 4.4. (g-**逼近**) 考虑 $g : Q \to P$ 为两个偏序集之间的单调映射，对于 $x \in P$，x 的 g-**逼近**为一个元素 $y \in Q$，使得 $x \leq g(y)$。x 的 **最佳** g-**逼近**为 $y \in Q$，使得

$$x \leq g(y) \wedge \forall z \in Q.(x \leq g(z) \Rightarrow y \leq z)$$

定义 4.5. (**Galois Connection**) 若存在 $f : P \to Q$，使得

$$\forall x \in P, y \in Q, x \leq g(y) \iff f(x) \leq y$$

则 f 为 g 的**左伴随 (left adjoint)**，g 为 f 的**右伴随 (right adjoint)**。

定理 4.1. *$g : Q \to P$ 为两个偏序集之间的单调映射，若存在 $f : P \to Q$，使得对于任意 $x \in P$ 存在*最佳 *g-逼近 $f(x)$，则 f 与 g 之间存在 Galois Connection。*

证明：根据定义构造 $f(x) := \min\{\ y \in Q \mid x \leq g(y)\ \}$
(1) $x \leq g(z) \Rightarrow f(x) \leq z$.

根据定义，因为 $x \leq g(z)$，所以 $g(z)$ 在集合 $\{\ y \in Q \mid x \leq g(y)\ \}$ 中，根据构造，$f(x) \leq z$。

(2) $f(x) \leq z \Rightarrow x \leq g(z)$.

根据构造，$x \leq g(f(x))$，因为 g 为单调映射以及 $f(x)$ 的构造，$g(f(x)) \leq g(z)$。通过偏序的传递性，$x \leq g(f(x)) \leq g(z)$。 $\square$

定义 4.6. （伴随函子 (adjoint functor)）

现在偏序集 P, Q 被抽象为范畴 $\mathcal{C}, \mathcal{D}$，单调函数 f 与 g 抽象为范畴之间的函子：$F : \mathcal{C} \to \mathcal{D}$，$G : \mathcal{D} \to \mathcal{C}$，$F$ 与 G 为一对伴随函子若存在双射族 θ,

$$\theta_{A,B} : \mathcal{C}(A, G(B)) \xrightarrow{\cong} \mathcal{D}(F(A), B)$$

则 F 为 左伴随，G 为 右伴随，记为 $(F \dashv G).f$

定义 4.7. （笛卡尔闭范畴，CCC）

范畴 $\mathcal{C}$ 是笛卡尔闭范畴 (Cartesian closed category, CCC) 当且仅当：

1. $\mathcal{C}$ 存在 terminal object T
2. 任意两个 $\mathcal{C}$ 的 object X, Y，有 product object $X \times Y \in ob\mathcal{C}$
3. 任意两个 $\mathcal{C}$ 的 object X, Y，有 exponential object $X^Y \in ob\mathcal{C}$

Part II

C-H 同构

5 简单类型 λ 演算

本节所采用的带类型 *lambda* 演算扩展主要参考自 *Lectures notes on the Lambda Calculus*, Peter Selinger。

5.1 语法

类型的 BNF 规则如下：

$$A, B ::= \iota \mid A \to B \mid A \times B \mid 1$$

1. 基础类型 ι 代表 integers, booleans 等的类型；
2. $A \to B$ 为从 A 到 B 的函数的类型；
3. $A \times B$ 是二元组 $\langle x, y \rangle$ 的类型，其中 x 有类型 A，y 有类型 B。

4. 1 表示只有一个元素的类型，类似程序语言中的 “void” 和 “unit”。

我们约定：$\times$ 的结合性比 $\to$ 更强，$\to$ 是右结合的。比如：$A\times B \to C$ 等价于 $(A\times B)\to C$，$A\to B\to C$ 相当于 $A\to(B\to C)$。

λ 项 (λ terms，或者 λ 表达式) 的语法：

$$M,N ::= x \mid MN \mid \lambda x^A.M \mid \langle M,M\rangle \mid \pi_1 M \mid \pi_2 M \mid *$$

其中，$\langle M,M\rangle$ 表示一对 λ 项，$\pi_i M$ 是一个 “投射”(projection)，定义为：$\pi_i\langle M_1,M_2\rangle = M_i$。$\star$ 表示类型 “1” 的元素（只有一个）。$\lambda x^A.M$ 表示 x 有类型 A。

注. 很多文献中对 $\lambda x^A.M$ 采用另一种写法 $\lambda x:A.M$

定义 5.1. （**自由变量**）

定义 5.2. （**闭合**）

一个 λ 项（表达式）是闭合的，如果它不包含自由变量

5.2 类型

从程序语言的角度，类型系统 (Type system) 属于 “静态语义”(static semantic)。下面给出简单类型 λ 演算的类型系统

定义 5.3. （**类型断言**，Typing judgement）

又称类型指派，$x:A$：表示变量 x 有类型 A；

定义 5.4. （**类型环境**，Typing environment）

又称类型上下文，一般用 $\Gamma,\Delta,...$ 表示，指代形为 $\Gamma=\{x_1:A_1,...,x_k:A_k\}$ 的类型断言集合，即：$x_1,...x_k$ 分别有类型 $A_1,...,A_k$。

注记. 如果 Γ 是一个类型环境，那么 $\Gamma,x:A$ 表示 $\Gamma\cup x:A$（这样写的时候，总假定 x 不出现在 Γ 中）

有了类型环境，类型断言可以表达为，$\Gamma\vdash M:A$，即在类型环境 Γ 中，M 有类型 A，$\Gamma=\{x_1:A_1,...,x_k:A_k\}$，注意 M 的自由变量必须包含在 $x_1,...,x_k$ 中。类型环境可以看做是一种假定、前提。

定义 5.5. (**定型规则**，Typing rule)

定型规则指一定类型环境中，表达式类型的推理规则。

下面是简单类型 λ 演算的定型规则：

1. (var) 规则：假定 x 有类型 A，则 x 有类型 A。

2. (*app* 规则：类型为 $A \to B$ 的函数可以引用到类型 A 的参数上，并产生类型 B 的结果

3. 等等。

$$\frac{}{\Gamma, x : A \vdash x : A}\ (var)$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ (app)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \to B}\ (abs)$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B}\ (pair) \qquad \frac{}{\Gamma \vdash \star : 1}\ (\star)$$

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A}\ (\pi_1) \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : A}\ (\pi_2)$$

定义 5.6. （**类型导出**，Typing derivation）

给定类型断言 $\Gamma \vdash M : A$，利用定型规则逐步推出它的过程叫做类型导出。如果存在一个类型导出，说明该断言是有效的。

例 5.1. 考察下面表达式，我们自底向上对其类型断言给出一个类型导出。

$$\vdash \lambda x^{A \to A}.\lambda y^A.x(xy) : (A \to A) \to A \to A$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \to A, y : A \vdash x : A \to A} \quad \dfrac{}{x : A \to A, y : A \vdash y : A}}{x : A \to A, y : A \vdash xy : A} \quad \dfrac{}{x : A \to A, y : A \vdash x : A \to A}}{x : A \to A, y : A \vdash x(xy) : A}}{x : A \to A, y : A \vdash \lambda y^A.x(xy) : A \to A}}{\vdash \lambda x^{A \to A}.\lambda y^A.x(xy) : (A \to A) \to A \to A}$$

定义 5.7. （**类型检查**，Type checking）

给定 Γ，M 和 A，找到一个类型导出。即，如果能够找到，类型检查通过。

定义 5.8. （**类型推导**，Type inference）

给定 Γ 和 M，确定 A。

定义 5.9. （**良型的**，Well-typed）

定义 5.10. （**类型可靠性**，Type soundness）

例 5.2. 考察以下类型，并思考：是否能够找到闭合的表达式，分别具有以下给定类型?

1. $(A \times B) \to A$
2. $A \to B \to (A \times B)$
3. $(A \to B) \to (B \to C) \to (A \to C)$
4. $A \to A \to A$
5. $((A \to A) \to B) \to B$
6. $A \to (A \times B)$
7. $(A \to C) \to C$

我们先给出答案：

1. $\lambda x^{A \times B}.\pi_1 x$
2. $\lambda x^{A}.\lambda y^{B}.\langle x, y\rangle$
3. $\lambda x^{A \to B}.\lambda y^{B \to C}.\lambda z^{A}.y(xz)$
4. $\lambda x^{A}.\lambda y^{A}.x$ 和 $\lambda x^{A}.\lambda y^{A}.y$
5. $\lambda x^{(A \to A) \to B}.x(\lambda y^{A}.y)$
6. 无法找到闭合表达式
7. 无法找到闭合表达式

由上例引出如下问题：

定义 5.11. （**类型居留问题**）

给定类型，是否一定存在相应类型的闭合 λ 表达式。找到特定类型的 λ 表达式的问题称为类型居留问题。

类型居留问题揭示了类型和逻辑之间巧妙的联系。下面将类型表达式中的“$\times$”换成逻辑与符号“$\wedge$”，“$\to$”换成逻辑蕴涵符号“ $\to$”，可以获得以下逻辑公式：

1. $(A \wedge B) \to A$
2. $A \to B \to (A \wedge B)$
3. $(A \to B) \to (B \to C) \to (A \to C)$
4. $A \to A \to A$
5. $((A \to A) \to B) \to B$
6. $A \to (A \wedge B)$
7. $(A \to C) \to C$

注. 有的文献为了方便区分，会用 $\supset$ 或者 $\Rightarrow$ 表示逻辑蕴涵，但是为了符合我们的通常印象且不失统一性，本文用 $\to$ 表示逻辑蕴涵符号。尽管它和类型中的符号 $\to$ 重复了，请读者根据上下文区分。

实际上，1-5 都是逻辑有效的（都是永真式，或者重言式），而 6 和 7 不是。后面我们会知道，给定类型，只有当它（通过以上方式）对应的逻辑公式有效时，才能找到相应的 λ 表达式！

简单类型 λ 演算和逻辑的这种联系（或者对应）称为 “Curry-Howard 同构”。更准确地说，此处的 “逻辑” 指代直觉主义命题逻辑。在后续的章节中我们将进一步阐述。

5.3 正则性

有两种正则性，强正则性（Strong Normalization），弱正则性（Weak Normalization)。强正则性表示，对于一表达式，无论采用任何规约方式，到最后都会变成值。弱正则性则是，对于一个表达式，都存在一种规约策略，可以规约成值。

简单类型 λ 演算有正则性，证明大体如下：先定义一个集合 R，一个表达式在 R 里面，当且仅当该表达式是强正则的，并没有自由变量，并且 1：该表达式的类型是基础类型 2：该表达式的类型是函数，并且对于所有输入, 如果输入在 R，输出也在 R

通过在类型推导规则上进行归纳，可以证所有的没有自由变量的表达式都在 R 里面，所以是强正则的。

对于形式化的证明，可以看 `https://www.cis.upenn.edu/~bcpierce/sf/current/Norm.html`

6 直觉主义命题逻辑

直觉主义逻辑又称构造性逻辑，产生于第三次数学危机之后。直觉主义者认为，集合论悖论的出现提醒我们必须根据可信性的要求对整个数学作全面的审查。什么样的概念和方法是可信的呢？他们提出的口号是”存在必须被构造“，就是说，所有的数学概念和方法都必须是构造性的，能够按照可操作的固定的方法，在有限步之内定义和实现。

在经典逻辑中，命题的解释是它断言的某种性质的真假意义，即命题的真值；而在直觉主义逻辑中，命题的解释是它断言的某种性质的可构造性。对于公式 A，A 的直观含义是 “A 是可构造的”，因此 “证明 A” 意味着 “构造 A”。

直觉主义逻辑和经典逻辑的最大区别是不承认排中律，即不承认 $A \vee \neg A$ 是可构造的。直觉主义逻辑认为，公式 A 意味着 “A 是可构造的”，而公式 $(\neg A)$ 意味着 “ ‘A 不能被构造’ 是可构造的”。也就是说，非 “A 是可构造的” 并不等价于对 $(\neg A)$ 的构造，非 “A 是可构造的” 仅意味着 “A 不能被构造” 这个断言本身，但并没有给出对这个断言的构造。

另外，直觉主义不承认实无穷，只承认有限和潜无穷。按照直觉主义逻辑的观点，因为无穷的构造不可能结束，所以不是实际可构造的。

6.1 语法

本节采用直觉主义逻辑形式系统中，有三个逻辑连接词 $\wedge$，$\rightarrow$ 和 $\top$。α 表示原子命题，$A, B...$ 是通过语法规则生成的公式。

$$A, B ::= \alpha \mid A \to B \mid A \wedge B \mid \top$$

6.2 BHK 语义

公式在直觉主义逻辑中是有效（valid）的，当前仅当它对于任何 Heting 代数上的任何赋值总得到 1

本节重点关注证明系统，语义部分待补充。。

6.3 自然演绎系统

请读者回顾经典命题逻辑的自然演绎系统。公式 A 的证明或者导出 (derivation) 建立在一系列假设 (assumptions) 之上。为了方便叙述，我们在这一节引入新的记号，把公式 A 的假设记为 $\Gamma = \{A_1, A_2, ..., A_n\}$。

$$\begin{array}{c}[x_1 : A_1, ..., x_n : A_n] \\ A\end{array}$$

以上形式的涵义是：基于假设集 Γ（也就是 $A_1, A_2, ...$）可以导出 A。值得注意的是，为求方便我们增加了 $x_1, x_2...$ 等符号，它们表示各个假设的“ 名”，或者“ 记号”。导出过程可能使用假设集中的假设零次或者不止一次，某个假设可能会被“消耗”。

下面是直觉主义逻辑自然演绎系统的规则。每个逻辑连接词由相应的导入/消去规则刻画。

1. **公理**：从假设 A 出发，可以证明 A。写在规则右边的 x 表示“名” 为 x 的假设被使用了。

$$(ax)\ \begin{array}{c}[x : A] \\ x \\ A\end{array}$$

2. $\wedge$ **引入**：如果基于 Γ 分别可以证明 A 和 B，则可以证得 $A \wedge B$ 的。换句话说：$A \wedge B$ 的一个证明，是一个 A 的证明和一个 B 的证明。

$$(\wedge I)\ \frac{\begin{array}{cc}[\Gamma] & [\Gamma] \\ A & B\end{array}}{A \wedge B}$$

3. $\wedge$ **消去**：从 $A \wedge B$ 可以证得 A 和 B。

$$(\wedge E)\ \frac{\begin{array}{c}[\Gamma] \\ A \wedge B\end{array}}{A}$$

$$(\wedge E)\ \frac{\overset{[\Gamma]}{A \wedge B}}{B}$$

4. $\top$ **引入**：不需要任何假设就能得到 $\top$。

$$(\top I)\ \frac{}{\top}$$

5. $\to$ **引入**：如果基于假设 $[\Gamma, x : A]$ 可以证明 B，则基于假设 Γ 可以证明 $A \to B$。要注意的是，$x : A$ 并从假设中 “消耗” 了。为表强调我们在规则后写上 x（A 的标记）。

$$(\to I)\ \frac{\overset{[\Gamma, x : A]}{B}}{A \to B}\ x$$

6. $\to$ **消去**：（等价于 Hilbert 系统的 MP 规则）

$$(\to E)\ \frac{\overset{[\Gamma]}{A \to B} \qquad \overset{[\Gamma]}{A}}{B}$$

为了更直观地体现与类型论的共同点，我们介绍自然演绎的另一种等价的表达形式。

在这种形式中，演绎规则不是导出一个个公式，而是使用了断言（judgment）的形式。它表示公式 B 是假设集 $A_1, ..., A_n$ 的后承。

$$x_1 : A, ..., x_n : A_n \vdash B$$

。这种写法可能会导致证明树有大量 “Γ” 而显得冗余，但是实际上更直观。

1. **公理**

$$(ax_x)\ \frac{}{\Gamma, x : A \vdash A}$$

2. $\wedge$ **引入**

$$(\wedge I)\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{B}$$

3. $\wedge$ **消去**

$$(\wedge E_1)\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$$

$$(\wedge E_2)\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

4. $\top$ **引入**

$$(\top I)\ \frac{}{\Gamma \vdash \top}$$

5. $\rightarrow$ **引入**

$$(\rightarrow I_x)\ \frac{\Gamma, x : A \vdash B}{\Gamma \vdash A \rightarrow B}$$

6. $\rightarrow$ **消去**

$$(\rightarrow E)\ \frac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

7 Curry-Howard 同构

下面进入本文的重点：简单类型 λ 演算与直觉主义逻辑的 Curry-Howard 同构。

7.1 类型和证明

比较 λ 演算的定型规则和上节自然演绎系统的规则，不难得出以下 “对应”

表 2: C-H 同构

类型系统	**直觉主义逻辑**
类型	公式（命题）
$\rightarrow$ 类型构造算子	$\rightarrow$ 逻辑连接词
变量	假设
λ 抽象	蕴含引入
λ 应用	蕴含消除

回到本文之前提到的类型居留问题。

定理 7.1. *存在带有特定类型的闭合 λ 表达式，当且仅当这个类型对应于一个逻辑定理。*

在直觉主义命题演算中，给定一个导出（或者断言）$x_1 : A_1, .., x_n : A_n \vdash B$，存在一个对应的 λ 项 M，且 $x_1 : A_1, ..., x_n : A_n \vdash M : B$。

7.2 规约和证明

Refer to book: Lecture on CH Isomorphism? continuation <-> classical, STLC <-> Propositional, cut elimination <-> reduction COC + Universe <-> ZFC System F <-> fragment of Second order intuitionistic logic Hilbert <-> SKI UTLC <-> Exfalso (Why total language matter)

8 λ 演算的指称语义与证明

本节从语义的角度考察 Curry-Howard 同构。

8.1 直觉主义逻辑的范畴论释义

8.2 λ 演算的指称语义

Scott Domain 是笛卡尔闭范畴 (CCC)。

Part III

进阶

9 从类型到逻辑

本文只介绍简单类型 λ 演算 (STLC) 的 Curry-Howard 同构。在后续发展中，研究人员陆续提出了 System F，构造演算等

9.1 回顾

我们重新梳理一下 λ 演算相关内容，首先是无类型 λ 演算。

$$E, F ::= x \mid \lambda x.E \mid (EF) \mid ...$$

然后是简单类型 λ 演算。注意，为了表述方便，本节采用和前文略有不同的记法

$$\sigma, \tau ::= \beta \mid \sigma \to \tau$$

$$E, F ::= x \mid \lambda x : \sigma.E \mid EF \mid ...$$

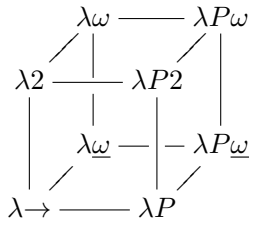
定型规则如下：

$$\text{VAR}\ \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

$$\text{ABS}\ \frac{\Gamma, x : \sigma \vdash E : \tau}{\Gamma \vdash (x : \sigma.E) : (\sigma \to \tau)}$$

$$\text{APP}\ \frac{\Gamma \vdash E : \sigma \to \tau \qquad \Gamma \vdash F : \tau}{\Gamma \vdash EF : \tau}$$

9.2 λ Cube 概述



$$K ::= \star \mid \Box \mid K \to K$$

$$T, U ::= V \mid S \mid TU \mid \lambda V : T.U \mid \Pi V : T.U$$

- Type polymorphism（项依赖于与类型，$\Box \to \star$），如 System F，由 Girard 和 Reynolds 提出。
- Type constructors（类型依赖于类型，$\Box \to \Box$）
- Dependent types（类型依赖于项，$\star \to \Box$）

9.3 依赖类型

9.4 归纳构造演算和 Coq

9.5 语义意义下的同构

10 从逻辑到类型

逻辑和证明论的发展也反过来影响了类型理论：从新的逻辑系统到新的证明论方法，从程序语言设计到定理证明原理。

表 3: Eight systems

类型系统	关系	例
$\lambda_{\to}$	$\star \to \star$	STLC
λ_2	$\star \to \star, \Box \to \star$	System F
$\lambda_{\underline{\omega}}$	$\star \to \star, \Box \to \Box$	Weak λ_ω
λ_ω	$\star \to \star, \Box \to \star, \Box \to \Box$	System F_ω
λP	$\star \to \star, \star \to \Box$	LF
λP_2	$\star \to \star, \star \to \Box, \Box \to \star$	λP_2
$\lambda P_{\underline{\omega}}$	$\star \to \star, \star \to \Box, \Box \to \Box$	Weak λP_ω
λP_ω	$\star \to \star, \Box \to \star, \star \to \Box, \Box \to \Box$	CoC

表 4: My caption

Logic	**Type System**	**Category Theory**
Intuitionistic Propositional Logic	Simple Types Lambda Calculus	Cartesian Closed Category
Second order Intuitionistic Logic	System F	
Classic Linear Logic	Linear Type System	Star-autonomous Category
	Extensional Dependent Type Theory	Locally Cartesian Closed Category
	Homotopy Type Theory without Univalence	Locally Cartesian Closed $(\infty, 1)$-Category
...	...	...

10.1 线性逻辑和线性类型系统

10.2 子结构逻辑和子结构类型 ⋆

10.3 Focusing/Polarizing 和类型理论 ⋆

Part IV

实践

11 Coq 入门

11.1 简述

Coq 是一个基于 Calculus of Construction 的扩展的 Proof Assistant。

Coq 的背后的原理加入的扩展有：

1: 加入了 Definition

2：加入了 Inductive，CoInductive data type

3: 为了一致性，加入了 Universe。也就是说，Type : Type 在 Coq 种不成立，成立的是 Type 0 : Type 1, Type 1 : Type 2, ... (有兴趣的读者可以试试看在 Coq 种开启-type-in-type 使得 Type : Type 后证明矛盾)

11.2 例子

Stay as close to propositional logic/ higher order logic as possible.

11.3 证明手段

Trusted Computing Base Reason of formal verification

Proof by refinement. Step by step, interactive, dialog like

Proof automation, De Brujin Criteria tauto, first order, meaning of formal verification, why tactic is OK

LCF approach Help user. ”Dialog like” property

11.4 其他教程

- *Software Foundations* by Benjamin C. Pierce et al. HTML version
  练习很多，适合 Coq 初学者自学和作为教材使用。前半部分是 Coq 本身和形式化证

明本身的一些东西，后半部分更多和程序验证相关。

- *Programs and Proofs - Mechanizing Mathematics with Dependent Types* by Ilya Sergey. Homepage
  本书亮点是对 ssreflect 用法有一定介绍。同样有练习，但是相对较少。
- *Certified Programming with Dependent Types* by Adam Chlipala. Homepage
  老司机必备

12 更多阅读

12.1 λ 演算、类型论

Syntax and Semantic of Lambda Calculus, Henk Barendregt

The Impact of the Lambda Calculus in Logic and Computer Science, Henk Barendregt

Martin-Lof's (extensional) Intuitionistic Type Theory (1975): taking the proofs-as-programs correspondence as foundational: a constructive formalism of inductive definitions that is both a logic and a richly-typed functional programming language

Girard and Reynolds'System F (1971): characterization of the provably total functions of secondorder arithmetic

Coquand's Calculus of Constructions (1984): extending system F into an hybrid formalism for both proofs and programs (consistency = termination of evaluation)

Coquand and Huet's implementation of the Calculus of Constructions (CoC) (1985)

Coquand and Paulin-Mohring's Calculus of Inductive Constructions (1988): mixing the Calculus of Constructions and Intuitionistic Type Theory leading to a new version of CoC called Coq

Coq 8.0 switched to the Set-Predicative Calculus of Inductive Constructions (2004): to be compatible with classical choice

12.2 数理逻辑和证明论

12.3 自动定理证明

12.4 Coq

SF

CPDT

CoqArt