λC 中逻辑概念的编码
The encoding of logical notions in λC

读书笔记

许博

1 疑问

1. P142. 析取的表示 $A \vee B \equiv \Pi C : *.(A \to C) \to (B \to C) \to C$，对它的解释为：*If A implies C and also B implies C, then C holds on its own*。为什么这里的 A 和 B 是或的关系？是否因为其为真时满足 $(A \vee B)$ 为真，算是强行解释吗？

2 类型理论中的谬论（absurdity）与否定（negation）

在章节 5.4 中，通过编码蕴含式 $A \Rightarrow B$ 为函数类型 $A \to B$，模拟蕴含式的行为，包括它的导入和消解规则。因为 λP 是 λC 的一部分，所以 λC 中同样拥有最小谓词逻辑。

本章将处理更多的接词（connective），比如否定（$\neg$），合取（$\wedge$）和析取（$\vee$）。这些在 λP 中不能表示，但在 λC 中存在非常优雅的方式去编码这些概念。

将否定 $\neg A$ 看作蕴含式 $A \Rightarrow \bot$，其中 $\bot$ 是“谬论（absurdity）”，也可以称为“矛盾（contradiction）”。因此 $\neg A$ 被解释为“A 蕴含了谬论”。为了这个目标，我们需要谬论的编码：

I. 谬论，Absurdity
命题“谬论”或 $\perp$ 的一个独特的性质是：如果 $\perp$ 为真，则每一个命题都为真。

每一个命题都为真，则存在一个接收任意一个命题 α 然后返回 α 的一个成员的函数，而这个函数的类型为 $\Pi\alpha:*.\alpha$。因此“如果 $\perp$ 为真，则每一个命题都为真”可以表述为“如果存在 $M:\perp$，则存在 $f:\Pi\alpha:*.\alpha$”。因此，在类型理论中，定义 $\perp$ 为 $\Pi\alpha:*.\alpha$。

$\perp$-消解（$\perp$-elimination）规则：

$(\perp\text{-elim})\ \dfrac{\perp}{A}$

因为 $\perp\equiv\Pi\alpha:*.\alpha$，则 $s_1=\square$ 且 $s_2=*$，所以 $\perp$ 存在于 $\lambda2$ 中，且 $\perp:*$。

II. 否定，Negation
定义：$\neg A\equiv A\to\perp$。

$A\to\perp$ 是 $\Pi x:A.\perp$ 的简写，其中 $A:*$ 且 $\perp:*$，所以 $(s_1,s_2)=(*,*)$。但因为 $\perp$ 存在，至少 $\lambda2$ 才能够表示否定。

$\perp$-导入（$\perp$-introduction）规则：

$(\perp\text{-intro})\ \dfrac{A \quad \neg A}{\perp}$

或：

$(\perp\text{-intro})\ \dfrac{A \quad A\Rightarrow\perp}{\perp}$

而 $(\neg\text{-intro})$ 和 $(\neg\text{-elim})$ 规则可以以 $(\Rightarrow\text{-intro})$ 和 $(\Rightarrow\text{-elim})$ 规则替换，前者为后者的特殊情况。

需要注意的是，尽管 $(\perp\text{-intro})$ 和 $(\neg\text{-elim})$ 都是 $(\Rightarrow\text{-elim})$ 的特殊情况，但两者具有不同的目的，前者是为了获得 $\perp$，而后者则告诉了我们如何使用一个否定 $\neg A$。

3 类型理论中的合取和析取

I. 合取，Conjunction
合取 $A\wedge B$ 为真当且仅当 A 和 B 都为真。在 $\lambda2$ 中，合取的表示为：

$A\wedge B\equiv\Pi C:*.(A\to B\to C)\to C$

这是一个合取的所谓“二阶”编码，比如 $A\wedge B\equiv\neg(A\to\neg B)$ 的一阶编码更为通用，因为后者只在经典逻辑中有效。

$\Pi C:*.(A\to B\to C)\to C$ 可以读作：对于所有的 C,（A 蕴含（B 蕴含 C)）蕴含 C。若将 A,B,C 都看作是命题，则可以解释为：对于所有命题 C, 如果 A 和 B 共同蕴含 C, 则 C 取决于自身。条件“如果 A 和 B 共同蕴含 C”也即“A 和 B 都为真”是多余的，而为了使条件成立，A 和 B 必须为真。这里我认为，C holds on its own 意为 $C\to C$，也即第二个 C 由第一个 C 蕴含，而 $C\to C$ 恒为真，所以条件是多余的，但条件需要满足（条件满足是命题为真的必要条件），所以 A 和 B 都需要为真。

$\Pi C:*.(A\to B\to C)\to C$ 被称为二阶是因为它是在命题上的泛化，而命题是二阶对象。

$\wedge$ 在自然演绎中的规则以及在类型理论中二阶编码的规则：

$(\wedge\text{-intro})\ \dfrac{A\quad B}{A\wedge B}$

$(\wedge\text{-elim-left})\ \dfrac{A\wedge B}{A}$

$(\wedge\text{-elim-right})\ \dfrac{A\wedge B}{B}$

$(\wedge\text{-intro-sec})\ \dfrac{A\quad B}{\Pi C:*.(A\to B\to C)\to C}$

$(\wedge\text{-elim-left-sec})\ \dfrac{\Pi C:*.(A\to B\to C)\to C}{A}$

$(\wedge\text{-elim-right-sec})\ \dfrac{\Pi C:*.(A\to B\to C)\to C}{B}$

II. 析取，*Disjunction*

析取 $A\vee B$ 的二阶编码：

$A\vee B\equiv \Pi C:*.(A\to C)\to(B\to C)\to C$

与合取的讨论相似，右边的表达式可以读作：对于所有 C,（$A{\to}C$ 蕴含（$B{\to}C$ 蕴含 C)）。将 A,B,C 看作命题，则可以解释为：对于所有的命题 C, 如果 A 蕴含 C 并且 B 也蕴含 C, 则 C 取决于自身，在逻辑上等价于：如果（A 或 B）蕴含 C, 则 C 成立（*hold*)。(这里的逻辑等价可能是指真值表相同?)。条件是多余的，所以 A 或 B 必须成立。

$\vee$ 在自然演绎中的规则：

$(\vee\text{-intro-left})\ \dfrac{A}{A\vee B}$

$(\vee\text{-intro-right})\ \dfrac{B}{A\vee B}$

(∨-elim) $\dfrac{A \vee B \quad A \Rightarrow C \quad B \Rightarrow C}{C}$

$\vee$ 在类型理论中二阶编码的规则：

(∨-intro-left-sec) $\dfrac{A}{\Pi C : *.(A \to C) \to (B \to C) \to C}$

(∨-intro-right-sec) $\dfrac{B}{\Pi C : *.(A \to C) \to (B \to C) \to C}$

(∨-elim-sec) $\dfrac{\Pi D : *.(A \to D) \to (B \to D) \to D \quad A \to C \quad B \to C}{C}$

至此，我们已经定义了类型理论中否定，合取以及析取的变体：

$\neg A \equiv A \to \bot$

$A \wedge B \equiv \Pi C : *.(A \to B \to C) \to C$

$A \vee B \equiv \Pi C : *.(A \to C) \to (B \to C) \to C$

但是在这三者中，A, B 都是自由变量，因此引入单个连词使得可以用于更“抽象”的表达式：

$\neg \equiv \lambda\alpha : *.(\alpha \to \bot)$

$\wedge \equiv \lambda\alpha : *.\lambda\beta : *.\Pi\gamma : *.(\alpha \to \beta \to \gamma) \to \gamma$

$\vee \equiv \lambda\alpha : *.\lambda\beta : *.\Pi\gamma : *.(\alpha \to \gamma) \to (\beta \to \gamma) \to \gamma$

再引入了 $\neg, \wedge, \vee$ 后，我们现在需要 $\lambda\omega$，因为需要依赖于类型的类型。

4 λC 中命题逻辑的一个例子

证明重言式：$(A \vee B) \Rightarrow (\neg A \Rightarrow B)$

(a)	$\boxed{A : *}$	
(b)	$\boxed{B : *}$	
(c)	$\boxed{x : (\Pi C : * \,.\, ((A \to C) \to (B \to C) \to C))}$	
(d)	$\boxed{y : A \to \bot}$	
(1)	$x\,B \;:\; (A \to B) \to (B \to B) \to B$	$(appl)$ on (c) and (b)
(e)	$\boxed{u : A}$	
(2)	$y\,u \;:\; \bot$	$(appl)$ on (d) and (e)
(3)	$y\,u\,B \;:\; B$	$(appl)$ on (2) and (b)
(4)	$\lambda u : A \,.\, y\,u\,B \;:\; A \to B$	$(abst)$ on (3)
(5)	$x\,B\,(\lambda u : A \,.\, y\,u\,B) \;:\; (B \to B) \to B$	$(appl)$ on (1) and (4)
(f)	$\boxed{v : B}$	
(6)	$v : B$	(var) on (f)
(7)	$\lambda v : B \,.\, v \;:\; B \to B$	$(abst)$ on (6)
(8)	$x\,B\,(\lambda u : A \,.\, y\,u\,B)(\lambda v : B \,.\, v) \;:\; B$	$(appl)$ on (5) and (7)
(9)	$\lambda y : A \to \bot \,.\, x\,B\,(\lambda u : A \,.\, y\,u\,B)(\lambda v : B \,.\, v) \;:$ $(A \to \bot) \to B$	$(abst)$ on (8)
(10)	$\lambda x : (\Pi C : * \,.\, ((A \to C) \to (B \to C) \to C)) \,.$ $\lambda y : A \to \bot \,.\, x\,B\,(\lambda u : A \,.\, y\,u\,B)(\lambda v : B \,.\, v) \;:$ $(\Pi C : * \,.\, ((A \to C) \to (B \to C) \to C)) \to$ $(A \to \bot) \to B$	$(abst)$ on (9)

Figure 7.1 A derivation of the logical tautology $(A \vee B) \Rightarrow (\neg A \Rightarrow B)$