一個外行對依值型別的理解

（始撰於 2023-07-14， 完稿 2023-07-16， 使用CC BY-NC-ND 授權）

依值型別 (dependent type) 是型別理論 (type theory) 的重要概念，也是 FP 的進階概念。因為概念很抽象，而且還要會點型別概念，方好入手。雖此係函數式程式語言的重要概念， 但是許多程式人， 會仰之彌高， 進而生畏， 降低學習意願， 遑論相關係的定理證明了 。

身為一個外行，之前筆者亦如是。對型別理論感到興趣之際，面對依值型別還是一頭霧水， 不知何以理解。 然最近稍理解些， 所以做了這份筆記 。

型別是一群變數的群體， 但型別自身有無群體 ？

大家應該都很清楚 ，1, 12, 1000, -1 都屬於 Int 這個「整數型別」; 3.1416、7.2、-11.22 都是 Float「浮點數型別」。就連一個函數輸入 Int 返回 Str 字串 (string) 的函數，也可以視為「輸入 Int 返回 Str」 的函數型別， 即 Int → Str 型別 。

型別內包許多東西（常數、變數等等），雖然型別和高中教的集合不一樣，但是也是「許多東西的合稱」。

這樣可以推論， 整數、 小數、 浮點數等型別， 甚至函數型別， 非常多元。 但是是否有「型別的型別」 ？ 就像一群村里組成鄉鎮一樣， 一群鄉鎮是否可以組成縣一樣 。

型別理論其實是有描述的， 稱為 “kind"（這個詞我不知道如何用華語表示）， 用星號 $*$ 表示。若是變數 x 屬於型別 T 時可以用 $x : T$ 表示， 同理， 型別 T 屬於 kind $*$ 時， 可以使用 $T :: *$（兩個冒號::用來和 「數 : 型別」 區分。 另外 kind 的群體叫做 “sort"， 但太複雜， 按下不表， 讀者現在只要知道 「type < kind < sort」 即可 。

該階層概念，係依值型別概念的先備知識。另一個概念是簡單型別 lambda 運算的型別指派 (typing)。

若有 apple， 且有 pen， 則……

以下簡單介紹一下簡單型別 lambda 演算的概念 。

假設我們有一個型別為Int的變數x，記為$x : \mathrm{Int}$，和旁邊的上下文（context，可理解為一群程式碼的集合，記為Γ）組合起來的新程式碼，即「$\Gamma, x : \mathrm{Int}$」之中，可推論另一數y屬於Str，即$y : \mathrm{Str}$。那麼，下列引數 (argument) 為x， 回傳y的匿名 lambda 函數（不會的可以看一下 JavaScript 或是 Python 對lambda 的講解） 的型別是什麼呢 ？

```
function(Int x){
    return y; // y is a "Str"
}
```

依上文以箭頭表示函數型別的方法， 是$\mathrm{Int} \rightarrow \mathrm{Str}$。

如果將整段話綜整描述的話，即：若從「上下文Γ和$x : \mathrm{Int}$」之組合中，可獲知$y : \mathrm{Str}$，則可推論出：

$$\text{從上下文}\Gamma\text{可知，}\texttt{function (Int x)\{return y;\}}\text{的型別是}\mathrm{Int} \to \mathrm{Str}\text{。} \quad [1]$$

用附帶型別的 lambda 運算中，可以表示為$(\lambda(x : \mathrm{Int}).\, y) : (\mathrm{Int} \to \mathrm{Str})$，其中括號可以省略，變成：

$$\lambda x : \mathrm{Int}.\, y : \mathrm{Int} \to \mathrm{Str} \quad [2]$$

將型別以代數 A, B 標記，context 之間，以逗號, 結合，並使用$A \Rightarrow B$（若A則B）的邏輯符號，[1]就變成：

$$\text{若從 }\Gamma, (x : A)\text{可推知}y : B \Rightarrow \text{從}\Gamma\text{可推知}\lambda x : \mathrm{Int}.\, y : A \to B \quad [3]$$

型別理論下「從A可得知B」以$A \vdash B$表示；「A可以推論出B」可以用$\dfrac{A}{B}$表示，所以[3]寫成：

$$\frac{\Gamma, (x : A) \vdash y : B}{\Gamma \vdash \lambda x : A.\, y : A \to B} \quad [\text{T-ABS}] \quad [4]$$

以上是如何建構 lambda 函數型別的方法，但是我們假設有一個函數$f : A \to B$，輸入一個變數$a : A$，那$f(a)$的型別是什麼呢？

回到現實的程式碼。假設 `f = function (A x){return y;} // y 型別是 B`，那`f(a)`直觀來看，回傳`y`，型別就是`B`。

因此，用型別理論的角度，我們可以這樣說：

$$\begin{gathered}\text{在上下文（一串程式碼）}\Gamma\text{中，若可知}f : A \to B\text{，}\\ \text{且據}\Gamma\text{亦可知}a : A\text{，}\\ \text{則自}\Gamma\text{可知}f(a) : B \quad [5]\end{gathered}$$

用型別理論的語言表示為：

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f \;\; a : B} \quad [\text{T-APP}] \quad [6]$$

其中$\dfrac{A \wedge B ...}{C}$的$\wedge$不標，以空格區分；$f(a)$寫成$f \;\; a$。

除了[4] T-ABS 和[6] T-APP 兩個 rule 外，簡單型別 lambda 運算還有其他的 rule：

1. 若在上下文Γ中，包含變數x，其型別為A，則從Γ可得知$x : A$。亦即：

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \quad \text{[T-VAR]} \quad [7]$$

2. 若c是型別為A的常數， 則在上下文Γ中， 可得知$c : A$。 亦即 ：

$$\frac{\text{c 是型別為A 的常數}}{\Gamma \vdash c : A} \quad \text{[T-CONST]} \quad [8]$$

和邏輯學的關係

把[4]移除程式變（函） 數的話， 會得到 ：

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \quad [9]$$

若是會命題邏輯 (sentence logic) 的話，可發現這個表達式和「若有串邏輯表達語句（邏輯論述）Γ和命題A，可得知命題B 的話，則可推論該論述可知若A則B」的表達方式一樣 。

[6]T-APP 若一樣移除變數， 可得到 ：

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \quad \text{[T-APP]} \quad [10]$$

和命題邏輯的「若一串邏輯表達語句Γ可知若A則B，且亦可知A，則Γ亦可知B」 一樣 。

這種型別推論和邏輯推論如出一轍的情況，叫做 Curry-Howard 同構 (Curry-Howard Correspondence)。 而簡單型別 lambda 運算的推論方法， 對應到命題邏輯 。

擴充到依值型別

現在我們可以推衍到簡單的依值型別形式λLF。

但什麼叫「依值型別」？ 可以想成「型別裡面內嵌著數」， 若一個屬於某型別S的數x被包在型別T裡面裝變數的「容器」，則T 這個型別，實際鑲嵌x，不論其型別S是浮點數、 整數、 或是樹狀結構、 函數， 只要是有 type 的都行 。

為了方便理解， 打比方 ：

我們假設有個型別，屬於 kind $*$，叫做$\text{DepType}(x)$，且有一個函數$f(x)$，輸入一個Int，輸出的型別是$\text{DepType}(x)$。則$f(x)$的型別不是固定的一個詞句，更非一成不變，而是取決於x的「表達式」， 隨x改變。 比如$f(42)$， 型別就叫做$\text{DepType}(42)$；$f(101)$， 型別就叫做$\text{DepType}(101)$。

但要如何在程式中， 生出（建構出） 依值型別的函（變） 數？ 推論過程比較複雜些 。

我們從[7]開始 ：

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \quad [11]$$

這是型別的推論法， 但是我們引入「型別需要屬於 kind」 的概念， 所以前提需要加一條 ：「 自上下文Γ， 可得知型別A屬於 kind $*$， 即$A :: *$」， 因此補充如下 ：

$$\frac{x : A \in \Gamma \qquad \Gamma \vdash A :: *}{\Gamma \vdash x : A} \quad \text{[T-VAR]} \quad [12]$$

然後為了建構依值型別， 我們需要引入 Pi 型別 (Π-type) 的概念 。

其為函數型別，輸入$x : S$，輸出值型別為T—— 但T除了是常型別如 ：Str, Int...，或型別變數$A, B, C, ...$外，也可以是包含可以被值套用 (applicable) 的依值型別，如$P(x), Q(x)$（型別內的括號 () 書寫時可忽略 ），甚至是另一個 Pi 型別。這種狀況可以用下列程式碼的函數類比 ：

```
// S 是Type
Type PiType1(S x){
    return T; /* T是 Type， 可以是P(x)、Q(x)、Array(x)
    PiType2(A y){...}等*/
}
```

Pi 型別表達如下 ：

$$\prod_{x:S} T \quad [13]$$

下列是合規的 Pi 型別 ：

1. $\prod_{x:\text{ Int}} \text{Str}$
2. $\prod_{x:\text{ Int}} (\text{Foo } x)$
3. $\prod_{x:\text{ Int}} \left(\prod_{x:\text{ Str}} (\text{Foo } x \;\; y) \right)$

為了建構一個 Pi 型別， 需要擴充[4]：

$$\frac{\Gamma, (x : A) \vdash y : B}{\Gamma \vdash \lambda x : A.\, y : A \to B} \quad [14]$$

因為引入型別屬於 kind 的概念， 所以前提仍加$A :: *$， 但是建構出的函數可不只是$\Gamma \vdash \lambda x : A.\, y : A \to B$如此簡單了，要考慮$B$是$\Pi$型別的情況，所以需要將$\Pi$型別的規定，套用到產出函數的型別， 修改如下 ：

$$\frac{\Gamma, (x : A) \vdash y : B \qquad \Gamma \vdash A :: *}{\Gamma \vdash \lambda x : A.\, y : \prod_{x:A} B} \quad \text{[T-ABS]} \quad [15]$$

要推論函數f套用一個變數a的返回值$f \;\; a$， 型別是什麼？ 需要擴充[6]：

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f \ \ a : B} \ [16]$$

先把函數f的型別改成 Pi 型別：

$$\frac{\Gamma \vdash f : \Pi_{x:A} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f \ \ a : B} \ [17]$$

因為$f(x)$這種 Pi 型別函數，回傳值的型別受到x的值改變，也就是套用變數a之後，B的型別裡面所有的x改成a（沒有的話不管它），才能得到正確的輸出型別。我們把「B裡面的x取代成a」寫成$[a \mapsto x]B$，因此推論的 rule 最後寫如下：

$$\frac{\Gamma \vdash f : \Pi_{x:A} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f \ \ a : [a \mapsto x]B} \ \text{[T-APP]} \ [17]$$

因此，若f型別是$\Pi_{x:\mathrm{Int}}\mathrm{Str}$時，套用 10 這個整數後，$f(10)$的型別是$[10 \mapsto x]\mathrm{Str}$，還是Str。這點和簡單型別 lambda 運算不變。但若是f 的型別是$\Pi_{x:\ \mathrm{Int}}\mathrm{foo}\ x$時，則$f(42)$型別是$[42 \mapsto x]\mathrm{foo}\ x$，即$\mathrm{foo}\ 42$，至$f(8)$的型別是$\mathrm{foo}\ 8$。

和邏輯的關係？

如之前說的 Curry-Howard 同構，依值型別的 typing 可類比謂詞邏輯 (predicate logic) 推論原則。說明如下：

假設P代表人 (Person) 這個類別，且我們從一串邏輯論述（上下文）Γ中可以得知「P屬於類別的類別」，且Γ和P組合在一起，可以得知$C \ \ x$，括號略，C指 Creature（生物）。可以推演出「在該論述下，對於所有x屬於人的時候，x是生物」。這句話用形式邏輯寫如下：

$$\frac{\Gamma, P \vdash C \ \ x \qquad \Gamma \vdash P :: *}{\Gamma \vdash \forall x \in P.\, Cx} \ [18]$$

和[15]移除變數並改寫型別符號的長相很像：

$$\frac{\Gamma, P \vdash C \ \ x \qquad \Gamma \vdash P :: *}{\Gamma \vdash \prod_{x:P} C \ \ x} \ [19]$$

而 1.「自邏輯論述Γ可知，$\forall x \in P.\, C \ \ x$」且2.「自$\Gamma$可知$m \in P$」，則3.「自$\Gamma$可知$C \ \ m$」這種三段論邏輯，可表達如下：

$$\frac{\Gamma, \vdash \forall x \in P.\, C \ \ x \qquad \Gamma \vdash m \in P}{\Gamma \vdash C \ \ m} \ [20]$$

[20]和[17]更改型別名稱，移除一些項目的形式很像：

$$\frac{\Gamma \vdash \Pi_{x:A} C\ \ x \qquad \Gamma \vdash m : A}{\Gamma \vdash [m \mapsto x] C\ \ x}$$ [21]

其中$[m \mapsto x]C\ \ x = C\ \ m$ 。

所以有說依值型別對應謂詞邏輯，Pi 型別對應全稱量詞∀。

接下來可以學什麼呢？

依值型別還有其他的變體，且上面僅介紹 Pi 型別，沒介紹 Sigma 型別（Σ-type，對應存在量詞∃）。另依值型別和電腦輔助數學（邏輯）證明息息相關，若是想要踏入這些大坑，還有其他書籍可以參考。使用依值型別的程式語言，包含 Agda、Coq、Lean 等，想瞭解的話可以另外學習。

但我還沒入坑，所以就留給各位看官探究了。