

The Trek through Pure Reason Begins: A Justification for Forms

國立台灣大學 / 2005 年 9 月 – 2009 年 6 月

柯向上 Josh H-S. Ko

完稿於 2009 年 7 月 5 日

說這一切正式始於蔡聰明老師的大一微積分，我相信並不過份——這門課帶來的遠不只是影響，而是衝擊。第一週上課並不是用正規微積分課程的實數、極限等基本概念介紹開場，而是跟隨 Leibniz 和 Newton 的腳步「發現」微積分學根本定理（Fundamental Theorem of Calculus）。蔡聰明老師先從求算

$$\sum_{k=1}^{100} \frac{1}{k(k+1)}$$

的例子開始，抽取出「差和分學根本定理」：

若能找到數列 b_k 使得 $\Delta b_k \equiv b_{k+1} - b_k = a_k$ ，則 $\sum_{k=1}^n a_k = b_{n+1} - b_1$ 。

這是藉著「求兩項的差」達成「求多項的和」。接著我們從有涯飛躍無涯，把求和記號 \sum 直覺地推廣為積分符號 \int ，並接受「無窮小的小精靈」 dx 。當我們從差和分學根本定理類推得到

$$\int_a^b dF(x) = F(b) - F(a)$$

並把它和函數 $f(x)$ 下所圍面積

$$\int_a^b f(x) dx$$

兩個式子陳列在一起時，蔡聰明老師問道：「究竟怎麼求算這個面積呢？」偉大定理即將誕生前的偉大時刻重現了！一陣屏息之後，T-H-E-O-R-E-M 一個一個字母重重地寫在黑板上，微積分學根本定理輝煌地出現在我們面前：

若可找到函數 $F(x)$ 使得 $dF(x) = f(x) dx$ ，則 $\int_a^b f(x) dx = F(b) - F(a)$ 。

這就是我的第一個大定理。

現在回想起來，蔡聰明老師對我的啟發是非常深刻的。「從數學發展脈絡敘說定理故事」的獨門風格搭配上蔡聰明老師特有的四射熱情，很容易就牢牢吸引住我這種上課熱切地盯著老師的學生，進而領會老師企圖傳達過來的意念。儘管日後我某種程度壓抑了這種參雜感情的觀點，但我仍然懷念這種觀點，也認為這樣的教學風格對於初學者是相當有效的。除了數學史的直覺脈絡，蔡聰明老師也不忘強調數學的嚴謹理路，向我們宣導「數學成熟度」的重要性：每個定理都要能用清楚無歧義的數學語言講出其前提、結論、和證明，定理與定理之間的邏輯網（logic net）應該了然於胸。這種要求對於只受過一般高中數學訓練的大一學生而言是很新鮮的，對於死心塌地崇拜老師的學生而言更是必須達成的理想目標。在我對型式的追求當中，或許這番關於數學成熟度的論述扮演了重要角色。

大一就要結束了，我對數學的興趣愈來愈明顯。看到數學家質精量多的成果之後，我也對編程方法產生懷疑。當時的焦點仍然放在 C++/Java OOP 和 C++ STL 上，OO 理論在型式化的標準衡量下完全無法和數學相比，而令我感到有意思的 STL 又似乎和代數血緣相近，讓我興生「往數學系取經」的動機。雖然我已經見到 Haskell，但我只著迷於它作為實用語言的簡潔和獨特思維，對 functional programming 所促成的強大推論能力仍一無所知。然而用比較浪漫的說法，Algebra of Programming 已經透過 STL 隱隱向我招手，而我因為經過大一微積分的洗禮有了一探究竟的勇氣，於是以「輔修數學系」作為回應。

接著登場的就是陳金次老師的大二高等微積分了。大一的初等微積分只是讓我們窺視數學分析理論的重要節點和附近的脈絡、學會微積分的演算規則，而高等微積分則要在我們面前把這個體系的根基建造出來。一開始的重點自然是紮實打造出微積分基本定理。微積分基本定理的「任督二脈」在陳金次老師的親切板書中清楚地浮現出來：從實數完備性開始，任脈通過區間套定理（長度趨近於 0 的閉區間嵌套序列交於一點）、Bolzano-Weierstrass 定理（閉區間內的數列必有收斂子數列）、連續函數的最大最小值定理（連續函數在閉區間上必能取到最大、最小值）和中間值定理（連續函數若在閉區間的兩端點上取異號值，則必在此區間內取得 0 值）接上初等微積分的均值定理（包括微分版和積分版），最後流入微積分基本定理的第一部份再抵達第二部份（Newton-Leibniz 公式）；督脈亦發源自實數完備性，從 Heine-Borel 定理（閉區間為緊緻集）導出連續函數在閉區間上均勻連續，進而保證連續函數在閉區間上為 Riemann 可積，最後得以宣稱微積分基本定理中所列的積分值確實存在。和「程式編寫」演對手戲的正是像這樣的“mental constructions”。我

在這一年的演練中逐漸體會到兩者的緊密連結，但要等到 FLOLAC '07 前夕才知道這連結早已經有了名字 —— Curry-Howard correspondence。

同一時間進行的是孫效智老師的倫理學，這是在哲學系的第一門正式課程。一方面我欣賞倫理學企圖以邏輯思辨方法處理道德這種複雜議題，一方面哲學不夠型式化的討論又不讓我滿意。哲學用詞和日常用詞重疊部份相當多，同一個詞往往每個哲學家有各自的定義，於是哲學家常常得花很多力氣澄清語詞意義，或因為辯論雙方的同一個語詞其實指涉不同意義而使整場辯論沒有實質內容。這和高等微積分理論建構的高效率和確定性立成對比，很清楚地展示型式化的威力。

大二還讀了基本的 computability theory。Turing 1936 年的論文¹ 是這麼引介 Turing machines：

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions [...]

由此句可看出 Turing machines 是為了模映「人類能執行的（機械）計算」所定義的。Church, Kleene, Rosser, Turing 等人證明 Turing machines 和 λ -calculus 以及 general recursive functions 的計算能力等價，Church-Turing thesis 進一步宣稱這些等價的計算模型完全模映「計算」這個概念，因此研究任一個模型就等同於研究人類的計算能力。一開始我覺得 Church-Turing thesis 令人不太舒服，這種不適感主要來自 Turing machines 的 finiteness，我原本認為這層限制過於嚴格。但隨著數學經驗增多，我發現我們處理的 infinity 其實都是 finite representations，真實的無窮不是我們能夠掌握的。接納 Church-Turing thesis 之後，computability theory 便在我心中佔據了特殊地位：這套理論研究的對象是我們自己的重要能力，研究的方法則是我信任的數學方法。如果我們連自己都不了解，何必急著把知識的疆域拓展到外在世界呢？

大二結束得十分戲劇化。才剛上完整年最後一堂課，馬上就在校園內摔車。吊著一隻左手在好友 Yen3 陪伴協助下，我參加期待已久的 FLOLAC '07 —— 首次舉辦的「邏輯、語言與計算」暑期研習營。似乎打從一開始，一切就已經在那裡靜靜等著我：出現在某篇 C++0x 論文² 的神祕 typing symbols、描述 imperative programs 正確性的老祖宗 Hoare logic、Algebra of Programming

¹On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, Series 2, 42, pp 230–265.

²G. Dos Reis, B. Stroustrup. A formalism for C++. ISO/IEC/JTC1/SC22/WG21 N1885.

的通識版本、優雅的 λ -calculus、intuitionistic logic 和終於正式見面的 Curry-Howard correspondence。 λ -calculus 模映的是數學上常見的代換演算。先以 untyped λ -calculus 為例，假設有個 countably infinite set of variable names 並以 x 指涉其中成員，那麼合法的 λ -expression (λ -term) 定義是

$$E := x \mid \lambda x. E \mid E E$$

其中 $\lambda x. E$ 相當於一個變數為 x 的未具名函式 (anonymous function)， E 裡可能提及 x 。這樣的函式右邊出現另一個 λ -term E' 時，就相當於函式套用 (function application)，把 E 裡面出現的 x 代換成 E' 作為結果。這相當於下面這個 β -reduction 規則

$$(\lambda x. E) E' \rightarrow_{\beta} E[E'/x]$$

λ -calculus 下的「計算」就是從一個 λ -term 開始，不斷套用 β -reduction 直到不能套用為止，最後得到的 λ -term 即為計算結果。這樣單純的定義就已經是 Turing-complete。我們也可以為 λ -calculus 加入型別 (types)，排除一些我們不希望有的 λ -terms。例如下面這個 simply-typed λ -calculus 的 typing rule

$$\frac{\Gamma \vdash F : \alpha \rightarrow \beta \quad \Gamma \vdash E : \alpha}{\Gamma \vdash F E : \beta}$$

說的是「若 F 是個把 α -typed object 映射到 β -typed object 的函式，同時 E 是個 α -typed object，那麼 $F E$ 就是個 β -typed object」，符合我們對於函式套用的直覺。此時若把型別以外的符號抹去：

$$\frac{\alpha \rightarrow \beta \quad \alpha}{\beta}$$

我們立刻發現這正是邏輯上的 *modus ponens* inference rule —— 「如果 α 蘊含 β 而且 α 成立，那麼 β 成立」！類似地，整套 simply-typed lambda calculus 的 typing rules 和 intuitionistic logic 的 inference rules 全部自然地對應起來，程式的型別是邏輯上的命題，而程式就相當於那個命題的證明。Curry-Howard correspondence 即泛指這一類演算系統和邏輯系統之間的正式對應關係。因為數學以邏輯為語言，Curry-Howard correspondence 點明了「數學理論建構」和「程式編寫」至少在型式上是同一件事情。此類風味的理論深深打動了我——這才是有意義的研究進路！稍後穆信成老師來信邀我加入他的 research group，我毫不猶豫地答應了，一腳踏入這個姑且以 programming language theory 概稱的未知領域。

很快地在大三，我為我的知識論找到最重要的一塊基石。哲學系梁益培老師的知識論以 Descartes 的第一篇和第二篇沈思錄 (*Meditations on First Philosophy*) 開場。在第一篇沈思錄裡，Descartes 敘述他意圖建構完全確定的知識體系，接著以三段論證拒絕一切外在世界知識的可靠性：

1. 感官獲得的資訊有時候是錯誤的。
2. 在夢境中 Descartes 常夢見荒誕不經的事情並相信它們，而因為夢境和現實無法區分（你怎麼證成你現在不是在做夢？），所以現實中也可能相信錯誤的事情。
3. Descartes 所感知的外在世界可能根本是某個 “evil genius” 刻意造出來欺騙他的。這個論證並不是說 evil genius 存在，而是說因為這個可能性無法被排除，外在世界知識的可靠性也就無法確立。

這種「方法上的懷疑論」（methodological scepticism）立刻吸引住我。這是「方法上」的懷疑論，因為終極目的是構造完全可靠的知識，懷疑論只是過程中協助檢驗的方法而已。懷疑論盡其所能排除不可靠的知識，留下的就是那些完全可靠的部份。Descartes 繼續摧毀他既有的知識，拋棄天地萬物乃至於他的身體，終於在第二篇沈思錄中找到第一個無可置疑的事實——我思故我在。當他懷疑一切、甚至懷疑自己是否存在時，他就立刻知道自己是存在的，因為存在才能懷疑。

至此，我先前的一切偏好似乎都獲得解釋了：我所企求的是那些**完全可靠、無可置疑的知識**。雖然 Descartes 的 evil genius 論證並非強到能夠完全拒絕關於外在世界的知識，但已足以指出這些知識並不穩固。若要讓這些知識在邏輯上站得住腳，我們無可避免必須引進一些假設（如歸納法的合理性），而且得為「這些假設出錯」的潛在情況作好準備。這不是我所想的可靠知識。於是有了知識論上的正當理由不投入研究關於外在世界的一切事物，包括各種自然科學以及建造於上的所有東西——因為我註定無法獲得關於它們的完全確定知識。

那麼究竟什麼是**完全可靠、無可置疑的知識**？即使是看似無可置疑的數學理論，搬起這些雄偉建築就會看到下面的基石集合論，而集合論這個型式理論之下又有非型式化的根基。繼續下探，我最終看到了邏輯。我的推論能力是可以懷疑的嗎？透過模仿「我思故我在」的論證，我判定邏輯是我的本質中無法排除的一部份。當我問任何一個型式是「為什麼」的問題、甚至「邏輯為何必要」這個問題時，我是因為想要理由而問問題，而探求理由正是邏輯的表現。

找到穩固的根基之後，自然應該從這根基開始發展。因此若想建造確定知識，當從邏輯開始。

於是我對型式邏輯的讚嘆就不令人意外了。型式邏輯試圖以符號捕捉邏輯概念，透過操作這些具體符號，我們得以清晰地研究邏輯的某個面向。型式邏輯成為數學理論的基礎，甚至使「型式化」成為數學理論的特徵，而經由 Curry-Howard correspondence，型式邏輯下的命題與證明也對應回計算學的类型別和程式！哲學、數學、計算學，我對這三個學門的興趣源頭其實是同一個：我希望探索我的“*mental construction*”的能力。我不急著運用這能力向外擴張自己的知識疆域，而是把這能力當作我試圖理解的對象，因此這番探索是純粹「向內」的。換個語境來講，在數學上我最有興趣的是各式各樣的構造與推論型式，在計算上我則希望對「程式」這種材料本身有更深入的理解——我不急著用數學或程式解決外在世界的問題。

型式系統的穩固也使它成為回應懷疑論的利器。原則上，一套理論型式化的部份就不再有歧義，推論的有效性也無可置疑，懷疑論者無法挑剔理論內容，只能把攻擊焦點轉移到在下面支持的型式系統。實務上，型式化是辛苦的工作，因為任何細節都不容許出差錯。自動計算的機器發明以前，Whitehead 和 Russell 的巨著 *Principia Mathematica* 花了兩千頁將數學的基礎概念型式化。而現在我們有了電腦，*Principia* 所需要的嚴格正確性驗證能夠讓機器自動完成。這讓型式化有了實務上的重大意義——我們終於有實用的方法可以確定理論或程式（的核心部份）完全正確無誤。

再來，型式化或許是解決「低效率溝通」問題的唯一途徑。自然語言本身是相當不精準的溝通媒介，未型式化的理論往往因此難以傳達，哲學上的辯證因為語言的模糊而容易失焦。相對地，數學以符號為主、口語解釋為輔的進路則相當成功。即便以最嚴格的標準而言，這仍不能保證意念的傳達是精確的，但至少雙方（或自己）對此概念的理解確切與否有了一個客觀判準，即必須符合型式的操作規則。隨著型式化的程度愈來愈高、手法愈來愈精緻，被刻劃的概念也就愈來愈能夠毫無歧義地被「凝結」在符號之中。Algebra of Programming 就是個完美的例子。整套 Algebra of Programming 建立在 inductive datatypes 上面，以 category theory 的簡潔語言描述。以自然數為例，我們首先用一個 functor $FX = 1 + X$ 描述自然數的形狀，說自然數的構造是一個“base case”和一個“inductive case”。然後在所有 F-algebras（型式為 $A \leftarrow FA$ 的 arrows）和 F-homomorphisms（尊重 F-algebra 運算結構的 arrows）所形成的 category 裡面找到 initial F-algebra $\alpha : \mathbb{N} \leftarrow F\mathbb{N}$ 。就是我們想要的自然數集，而 α 則是 \mathbb{N} 和 $F\mathbb{N} = 1 + \mathbb{N}$ 之間的 isomorphism，意味

著自然數同構於「一個 base case 加一個 inductive case」的結構。「 α 是個 isomorphism」其實是導自 α 的 initiality：從 α 到任意的 F-algebra $f : A \leftarrow FA$ 都有唯一的 F-homomorphism $((f)) : A \leftarrow \mathbb{N}$ 使下圖流通 (commute)：

$$\begin{array}{ccc}
 \mathbb{N} & \xleftarrow{\alpha} & F\mathbb{N} \\
 \downarrow ((f)) & & \downarrow F((f)) \\
 A & \xleftarrow{f} & FA
 \end{array}$$

在 functional programming 上我們稱 $((f))$ 為一個 fold。上圖暗示 α 能夠被 $((f))$ 代換為 f ，而這其實就是 mathematical induction。從這樣的構造立即可以看出 mathematical induction 對於自然數有多麼關鍵，因為這來自 α 的 initiality，後者又正好是當初尋找 \mathbb{N} 的條件。另外從這構造也可看出「自然數是造出來計數的」，因為每個自然數是藉著逐次套用 α 所造出，保持著「逐次」的結構，等著被 $((f))$ 具現成「逐次套用 f 」的動作。和傳統的 Peano postulates 相比，這樣的 formulation 緊緻地包裝了更多更清楚的資訊，同時相當容易推廣。沿著這條進路，Algebra of Programming 接著將 relations 視為“potentially nondeterministic and partial functions”並發展一套 relations 的算術，然後解掉數類最佳化問題。我們把問題的規格寫成一個 relation，然後根據這個 relation 的性質套用適當的定理，從問題規格逐步演變為高效率的演算法。例如最簡單的一類問題型式是

$$\min R \cdot \Lambda(S)$$

意思是以一個 fold 產生所有 feasible solutions，然後在 R 這個 order 之下取一個最佳解。若我們能證明「比較差的 partial solutions 不會導致最佳解」

$$S \cdot FR^\circ \subseteq R^\circ \cdot S$$

此時我們就可以把「取最佳解」的步驟 $\min R$ 分發到 fold 裡面

$$\min R \cdot \Lambda(S) \supseteq ((\min R \cdot \Lambda S))$$

亦即在產生解答的每一步直接選一個當下看起來最好的解。不意外地，這個定理稱作 Greedy Theorem。Greedy Theorem (和 Algebra of Programming 其他關於最佳化問題的定理) 的貢獻不在於解決新一類問題，而是示範如何把某一

類 greedy algorithms 的關鍵思考模式凝結於符號之中，反映直覺又有堅固基礎，甚至可供具體操作。這樣的進路值得效法。

然而，型式化往往被視為相當嚴重的限制，使得研究型式看似沒什麼潛力。例如一個常見問題是：型式系統真的成功模映我們的計算方式（思考）嗎？神經科學的發展很容易讓人提出一些反對論點，例如大腦的複雜運作機制很可能超過型式系統所能掌握的範疇。這常被視為對 Church-Turing thesis 的質疑，但其實並非如此。Church-Turing thesis 所意圖定義的「有效計算」不是指那麼廣義的計算——型式系統從來就不意圖模映完整的思考，而是模映「機械的」型式推理，因此根本不是為了回答「何謂思考」而提出來的。那只考慮型式推理就夠了嗎？我的看法是，我們必須要能嚴謹地把一件事情寫下來，才算真正了解那件事情。一旦要求嚴謹寫下來，事情就落入型式系統的領域了。換個角度講，無法寫下來的事情就不是我關注的焦點，因為我無法保證能有效地操作它。我們很可能永遠無法了解思考的本質，但至少我們可以研究那些能夠寫下來細細省視的推理型式。總結來說，能嚴謹寫下來的東西才是我們可能精確掌握的，而只鑽研那些我們可能精確掌握的東西，在我看來相當合理。型式系統的侷限或許正好反映我們處理精確事物時必須承認存在的限制。

所以我選擇把我的焦點集中在幅員相對狹小的型式部份，但這不代表我必須放棄其餘的疆域。懷疑論拒絕後者的絕對可靠性，但也沒能將它們完全排除。只舉一個單純例子：音樂對我的影響是如此之大，讓我不得不認為它對於人性有特殊意義，但其影響看來幾乎不可能以邏輯方法解釋。這趟旅程我將以型式為目的地，沿途觀賞其他風景，或許找到更多值得探勘的景點，試試能看得多廣、走得多遠。