Clo: another typesetter

一個排版器的實作心得

陳建町

版權聲明

目　錄

序言

以前從國中時候試用 Linux 以及架站以後，就開始想用 LaTeX 排版些自己所寫的東西，其中包含覺得 LaTeX 的語法不好想要重造輪子。就算後來大學沒有走上資訊工程這條路，還是希望有天至少能夠完成個能用的雛形。

但是這是涉及字體檔案的處理、PDF 的處理、語法分析，後來自己因為不知道如何開發，所以一直停擺。不是遇到很多蟲，就是效能問題有缺失。因為時間繁忙很少更不消說了。甚至買了 Knuth 教授的 *Digital Typography*，想要瞭解斷行演算法，結果粗估五、六十頁，所以幾乎沒有讀。

另外筆者一個分支興趣是編譯器的相關知識，所以開始讀和王垠的編譯器思想系出同門的 Jeremy G. Siek 所著作之 *Essential of Complication: An Incremental Approach in Racket*（編譯之要素：Racket 語言的遞增的方法）。我想到：既然編譯器這種複雜的軟體，可以一層一層的用 pass 來遞增功能，就像水彩從背景、大物體一直由少漸多的完成。而排版軟體也是把使用者輸入的排版之領域特定語言(DSL)轉換成文字、圖形和二維座標對應關係（最後匯出成 PDF 或 SVG 等等）的編譯器，若是能夠用層層遞增的方法來完成，相信也能夠避免結構的複雜化導致錯誤容易發生的挫折。

然而排版語言不只是輸入文字轉圖形而已，更重要的是還要有因應美觀的自動斷行(justification)和斷字(hyphenation)等等的演算法、還有 PDF 的基本知識、字型函式庫的取用、排版要求（多欄）、甚至還牽涉到語言特有的特性：比如東亞全形文字（漢字、諺文、日文

假名、注音符號）和非全形文字中間要加空白，以及從左寫到右的文字（希伯來字母和阿拉伯字母等）的排版方法，不一而足。

為了簡化起見，且目標讀者是臺灣的受眾，本書僅涉及到 ASCII 英文字母——頂多加些一些附加符號(diacritics)和漢字的排版。其他的功能希望讀者可以漸次由少漸多的附加。另外這邊會使用到一些 LISP 的表達式來表達抽象語法樹，若是不懂的話，可以看一點教 Lisp 或是 Scheme 的書，如 SICP。另外這本書不是編譯原理和描述 PDF 規格的書，不涉獵底層的知識，有需要的可以參考相關領域的書。

致謝

感謝 Donald Knuth 教授開發出這麼一套排版系統以及排版的演算法，除了造福科學排版的諸多用戶外，也間接鼓舞我想要研究排版軟體如何實作；感謝 Jeremy G.Siek 老師的 *Essential of Complication: An Incremental Approach in Racket*，讓我獲得排版語言編譯器設計的啓發。感謝王垠讓我對編譯器相關的技術有興趣，從而斷斷續續學習相關的資訊。

感謝愛爾蘭語，除了讓我對語言和語言復興的知識打開新的世界以外，這個軟體的名字 Clo 也是從這裡來的（cló 有「活字」的意思，因為技術限制抱歉沒辦法輸入長音符號）。

感謝我的父母，雖然專長不是電腦資訊科技，但是要感謝他們讓我讓我有餘力能夠在中學的時候研究這種興趣，這條路才能走下去。

感謝這本書閱讀的人們，讓我知道筆者不是孤單的。

Siōng-āu Kám-siā góa ê Siōng Chú, nā-bô i ê hû-chhî kap pó-siú, chit-pún chheh iā bô-hó oân-sêng.（最後感謝上主，若無扶持保守，這本書也很難完成）

1 先備知識

這不是教一位入門使用者如從零知識撰寫排版軟體的書，讀者應該有知道如何使用靜態型別語言的經驗，比如一點 C、或是 Rust 等等。另外抽象語法樹為求方便，使用 LISP 撰寫，所以需要會 LISP 和 Scheme 的知識（知名教科書 SICP 的開頭可以讀一讀）。

這本書也不教編譯理論和 tokenizing、parsing、狀態機等等的，頂多只會帶到一些很基礎的知識，有需要的請另外再讀。所以使用者需要會有使用正規表達式(regex)的能力。

1.1 抽象語法樹

C 語言、Python 語言就算有許多的關鍵字、操作符、符號或是常數變數，在編譯器分析語法以後，最後會轉成編譯器可以操作的樹結構，然後再轉成我們想要的另一個語言的樹，最後輸出另一個語言的程式碼。

但是什麼叫做抽象語法樹呢？我們先從一點句法知識來談。

學過中學國文文法的課程，會背一堆類似「主詞+動詞+受詞」、「主詞+(有/無)+受詞」的結構。可以換個說法，是句子=「主詞+動詞+受詞」或是「主詞+(有/無)+賓詞」的形式。我們將「=」寫成「::=」，「/」（或是）寫成「|」，動詞擴充變成「動詞片語」，就變成:

```
句子 ::= (主詞 動詞片語 受詞) | (主詞 (有 | 無) 受詞)...
```

用這種形式表示的語言句法，叫做「BNF 文法」。這種句法看起來很語言學，但是我們想: 受詞和主詞可以為名詞、專有名詞或是「形容詞+名詞」; 動詞片語可以為動詞或是「副詞+動詞」。因此這樣之規則，就可以生成許多句子，比如「我有筆」、「張三養貓」、「小芳

慢慢移動檯燈」等等的句子。然後句子可以用上述規則，分析成語法的樹狀結構，如圖 1 把「我曾旅居新竹」寫成語法樹。

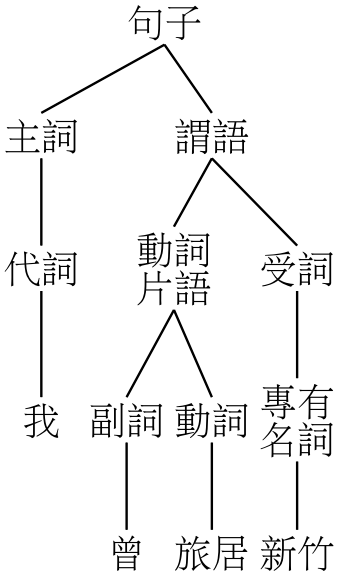

圖 1: 「我曾旅居新竹」的語法樹

同理，程式語言通常也有更嚴謹的這樣生成文法，可以用幾個簡單規則生出繁多的程式碼，而且合乎語法規定。這種生成文法也可檢查輸入的程式碼有沒有符合句法的規定。而這種語法生成的程式碼，去掉不需要的逗號等等符號，當然也可以做成語法樹，就是抽象語法樹 (abstract syntax tree, AST)，如圖 2 所示。

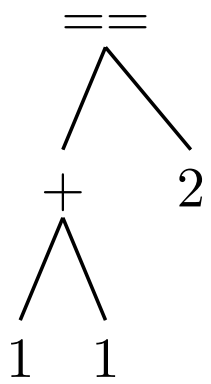

圖 2: `(2+2) == 4` 的語法樹。注意括號已經刪除。

而上文的抽象語法樹，可以是我們把程式經過編譯器分析之後，用「樹」儲存的資料結構。而樹形結構我們可以使用 Lisp 語言的 S 表達式(S-expressiom; S-exp)來表示，本文採用這樣的表示方法。所以上文的`(2+2)==4` 即`(== (+ 2 2) 4)`；`let baz = foo("bar")`，若是把 foo(“bar”) 這種函數套用(apply)寫成`(APPLY foo "bar")`，則其 S-exp 語法樹可寫為`(let baz(APPLY foo "bar"))`。