Verification of Correctness and Time Complexity of Skew Binomial Heap via CALF

高文君

摘要：非形式化的演算法證明不管是正確性還是時間複雜度上都有出錯的可能性，目前存在 Calf 這個用於演算法證明的邏輯框架讓我們能利用它訂定的規則在 Agda 中表示演算法、並證明它的性質。此專題為一個案例研究，將 Skew Binomial Heap 的定義利用Calf的語言表示在 Agda 中並證明它的正確性與時間複雜度，最終此專題證明了 Skew Binomial Heap 的關鍵性質，即 insert 操作所花的時間為常數而獨立於資料結構的大小。

問題動機

沒有經過 proof assistant 驗證過的數學證明都有可能出錯，演算法相關的證明也不例外，有個2008年被發表的一篇有關圖論的定理[Addario-Berry et al., 2008]經過了12年後才被發現有一句推論是錯的[Addario-Berry et al., 2020]，而這個錯誤推翻了這篇論文給出的結論，雖然是圖論的論文，但是由於圖論和演算法之間有很強的關係，所以那段時間還是有很多演算法的論文有引用它；1991年一篇圖論演算法的論文[Xue-Hou Tan et al., 1991]也在8年後被發現有錯誤[Xue-Hou Tan et al., 1999]，使得正確的時間複雜度應為 $O(n^4)$ 而非聲稱的 $O(n^3)$ 。這兩個事件說明了演算法的正確性與複雜度有被 proof assistant 驗證的必要性

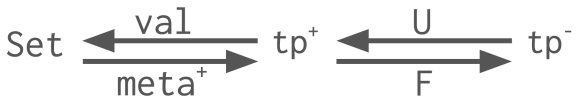
CALF (Cost-Aware Logical Framework)

Calf 是一種可以同時做正確性與複雜度分析且被實作在Agda中的邏輯框架[Niu et al.,2022]，它的用法簡單來說就是在 Agda 中用它提供的 metalanguage 來撰寫演算法，也就是寫程式，並在中間註記時間開銷的「標籤」，最後再把整個程式的標籤收集起來。

Calf 裡能寫的程式中的會出現的型別可以被分成兩個類，`tp⁺` 和 `tp⁻`，前者代表「數值」，後者代表「計算」[Levy, 2012]。以演算法可能會出現的型別來舉例的話，「自然數」這個型別屬於 tp^+，而「一條程式碼」或是一個「函數」的型別則是屬於 tp^-。

`U` 和 `F` 兩個函數會聯繫 tp^+ 和 tp^- 之間的關係，`U(X)` 代表將某個計算X擱置，即可把它當作數值以傳給某個函數當輸入；而 `F(A)` 則代表一個運算完的結果型別為 A 的計算

為了把 calf 實作在 Agda 中，需要建立起 `Set` 和 metalanguage 中的 tp^+ 與 tp^- 之間的關係。Calf 中的所有數值型別，包括 `nat`，皆是利用 $meta^+$ 從 Agda 中移植到 calf 的 metalanguage 中的，只要利用 `val` 將 tp^+ 轉換回 `Set`，我們就能利用 Agda 中定義過的運算或是已經證明過的性質了。同樣地，我們可以以 `val∘U` 來將 tp^- 轉成Set，由於實作時很常使用，所以定義 `cmp := val∘U` 。



Calf 中要寫程式最基本的兩個語法是：

```
ret: val A → cmp(F A)
bind: (cmp (F A)) → (val A → cmp X) → cmp X
```

`ret` 和 `bind` 讓我們可以做 monadic programming，而我們想要的 computation effect 就正是時間的開銷，利用 writer monad 可以很方便地把一個演算法的時間開銷紀錄起來。

Calf 中可以用`step`來標記某個計算的時間開銷

```
step : (X : tp⁻) → ℂ → cmp X → cmp X
```

`ℂ` 代表時間開銷的單位，最常用的就是自然數。`step X c e` 在行為上和 `e` 一樣，但是有 `step` 的版本附加了開銷的標記。

有了附上標記的函數了以後我們就能用一些 deduction rules 來證明一個計算的總時間開銷有某個上界。

Return: $\frac{}{isBounded\ (A; ret(a); 0)}$ Step: $\frac{isBounded\ (A; e; d)}{isBounded\ (A; step^{c}(e); c+d)}$

Bind: $\frac{isBounded\ (A; e; c)\ \ \forall a{:}A.isBounded\ (B; f(a); d(a))}{isBounded\ (B; bind(e; f); bind(e; \lambda a.c+d(a)))}$ Relax: $\frac{isBounded\ (A; e; c)\ \ c \leq c'}{isBounded\ (A; e; c')}$

前三條規則針對三個語法，`ret`, `step` 和 `bind` 要如何計算他們的時間開銷，最後一個規則則說明了上界可以被放寬。

Skew Binomial Heap

這次實作的目標，Skew Binomial Heap 是 Meldable Priority Queue 的其中一種實作方式 [Okasaki, 1999]，Meldable Priority Queue 這類資料結構需要維護一個集合，集合中的元素可以比大小，並且有四個操作：`insert`, `find-min`, `delete-min` 與 `meld`，分別可以插入一個元素進入集合、找到集合中的最小元素、刪除集合中的最小元素和將兩個 queue 合併。

Skew Binomial Heap 和其他實作方式比的特色有：

1. 較具直覺性並好理解的設計方式
2. 保持 `meld` 的效率($O(logn)$)的同時讓 `insert` 成為一個花費常數時間的操作
3. 將此資料結構視為一個抽象介面以後（不用修改內部運作方式），經過一點資料結構上的轉換可以成為就時間複雜度而言最佳的 Meldable Priority Queue [Brodal, Okasaki, 1996]

研究規劃與成果

我將完整的研究規劃分成了五個階段

1. 使用 Haskell 實作 Skew Binomial Heap
2. 在 Agda 中定義出 Skew Binomial Heap
3. 利用 calf 的語言實作有關這個資料結構的函式
4. 利用 calf 的工具證明3.中定義的函式的時間複雜度
5. 證明正確性

最後，第一和第二階段全部完成，三至五階段皆部分完成，各個函式對應的進度會以表格的方式放入附錄

後續問題

多輸入函數的時間複雜度

Calf 目前的實作只有支援單變數輸入的函數能用 Big-O Notation 來表示其時間複雜度，但是很多演算法可能有多個變數輸入。解決這個問題很直接的想法就是給 calf 的函數擴充 uncurrying 的功能，不過至今為之能找到的實作例子中還沒有人嘗試對 calf 做這個擴充。

如何讓函數定義與性質證明同時進行

在 calf 中遞迴地定義某個函數f時，可能會遇到f需要自己的性質才能定義的情況，例如 f(n) 的定義或許會需要 f(n-1)>0 的證明等，當實際狀況比較簡單時，可以利用Σ型別來讓函數同時輸出本來的輸出與需要的證明來解決問題，但是當狀況較複雜（邏輯分支較多、需要多個性質的證明）時，這個做法就會變得很冗長。

演算法的實作用與證明用表示方式

一般演算法論文中都會用 pseudocode 來描述他們提出的演算法，這類語言的設計目的是為了讀者能快速了解演算法的流程，並能輕鬆用他們常用的程式語言實作該演算法。這類用於實作的語言有一個與 proof assistants 合不來的點是，由於實作時不一定會需要在乎定義，表示資料結構的方式常常會選擇越簡單、越有效率越好，例如以這次實作中遇到的例子而言，一個除了前兩項可以相等以外嚴格遞增的序列在實作中會被以一個一般的序列表示，但是在證明時還是會需要將後者的表示方式修改成前者的表示方式，這個步驟是否是多餘的？在演算法論文提出前者表示方式的定義對嚴謹性有無幫助？再者，同一個資料結構可能被以不同的方式來實作，若證明中的資料結構不是以概念式的方式定義的話，就沒辦法寫出不依賴於實作方式的證明了，反而是抽象的證明比較好修改成各個實作方式的證明，我認為甚至有自動化的可能性。

參考資料

Louigi Addario-Berry, Maria Chudnovsky, Frédéric Havet, Bruce Reed, and Paul Seymour. 2008. Bisimplicial vertices in even-hole-free graphs. *Journal of Combinatorial Theory, Series B* 98, 6 (2008), 1119-1164. DOI:https://doi.org/10.1016/j.jctb.2007.12.006

Louigi Addario-Berry, Maria Chudnovsky, Frédéric Havet, Bruce Reed, and Paul Seymour. 2020. Corrigendum to “Bisimplicial vertices in even-hole-free graphs”. *Journal of Combinatorial Theory, Series B* 142, (2020), 374-375. DOI:https://doi.org/10.1016/j.jctb.2020.02.001

Gerth Stølting Brodal and Chris Okasaki. 1996. Optimal purely functional priority queues. *Journal of Functional Programming* 6, 6 (1996), 839-857. DOI:https://doi.org/10.1017/s095679680000201x

Harrison Grodin, Yue Niu, Jonathan Sterling, and Robert Harper. 2024. Decalf: A Directed, Effectful Cost-Aware Logical Framework. *Proceedings of the ACM on Programming Languages* 8, (2024), 273-301. DOI:https://doi.org/10.1145/3632852

Harrison Grodin, Yue Niu, Jonathan Sterling, and Robert Harper. 2024. **calf**: A Cost-Aware Logical Framework. *Artifact Digital Object Group*(2024). DOI:https://doi.org/10.1145/3580425

P.B. Levy. 2012. *Call-By-Push-Value*. Springer Science & Business Media.

Runming Li, Harrison Grodin, and Robert Harper. A Verified Cost Analysis of Joinable Red-Black Trees. *Arxiv*. Retrieved August 21, 2024 from https://arxiv.org/abs/2309.11056

Yue Niu, Jonathan Sterling, Harrison Grodin, and Robert Harper. 2022. A cost-aware logical framework. *Proceedings of the ACM on Programming Languages* 6, (2022), 1-31. DOI:https://doi.org/10.1145/3498670

Chris Okasaki. 1999. *Purely Functional Data Structures*. Cambridge University Press.

Xue-Hou Tan, Tomio Hirata, and Yasuyoshi Inagaki. 1991. An incremental algorithm for constructing shortest watchman routes. *Lecture Notes in Computer Science* 557, (1991), 163-175. DOI:https://doi.org/10.1007/3-540-54945-5_60

Xue-Hou Tan, Tomio Hirata, and Yasuyoshi Inagaki.. 1999. CORRIGENDUM TO "AN INCREMENTAL ALGORITHM FOR CONSTRUCTING SHORTEST WATCHMAN ROUTES". *International Journal of Computational Geometry & Applications* 9, 3 (1999), 319-323. DOI:https://doi.org/10.1142/s0218195999000212

附錄

Priority Queue: 一個 priority queue 是一個附帶五個操作的集合，已知集合中的元素有比大小的運算，由於其中一個操作(`decrease-key`)被提出 Skew Binomial Heap 的作者列為 future work，這裡列出其他四個我們考慮的操作

`Find-Min`	回傳 priority queue 中最小的元素
`Delete-Min`	刪除 priority queue 中最小的元素
`Insert`	插入一個元素進 priority queue
`Meld`	將兩個 priority queue 合併成一個 priority queue，新的 priority queue 中的元素為原本兩個的聯集

Skew Binomial Heap: 參見[Brodal, Okasaki, 1996]

進度表格

函式名稱	用Calf實作	證明時間複雜度
link	100%	100%
skewLink	100%	100%
insertTree	100%	100%
uniqify	100%	0%
meldUniq	100%	0%
insert	100%	100%
meld	5%	0%
findMin	5%	0%
deleteMin	5%	0%

藍色標注的為資料結構內部使用的函數，其餘的四個函數為 priority queue 這個資料結構類別需要的函數，由於 agda 中的定義本來就包含一部分的正確性證明，正確性的進度較難估計做了多少，所以表格中只有 calf 實作和時間複雜度證明的部分。
成果目前公開在 : https://github.com/YunXiuRZ/Brodal-Okasaki/tree/main

個人化背景故事

大約一年前修系上的進階演算法課時，比起基礎的演算法課多出了很多較為複雜的證明，例如 Fibonacci Heap 的證明，使我無法像上基礎課的時候一樣只要大概看過證明的論述就會覺得很顯然是對的，甚至上課提供的教材中也有直接出現教授認為很顯然的事情實際上存在反例，

那個教授甚至本來就是以演算法為研究專業的，這也讓我在乎起了「如何確保演算法證明沒錯」以及「付出的心力是否值得」。

由於已經有團隊提出了在 Agda 上的（函數式）演算法證明框架，所以此專題著重於實作一些例子並提出問題，而不是從零開始設計一個框架。我挑選的是 Binomial Heap（一種Priority Queue）的一種變種，Skew Binomial Heap。挑選此資料結構的原因是第一次知道 Binomial Heap 這個資料結構時，被它優美的定義以及操作上的直覺性吸引，因此即便它的效率比不上其他進階的 Priority Queue，我仍想透過它來做到一些事。而經過了一番搜尋，我找到了基於 Binomial Heap 的思想來設計的 Skew Binomial Heap，這個資料結構也成為了 Optimal Priority Queue (without decrease-key)的基礎。