Monad 與副作用

單中杰

2022-08

暖身

純遞迴 Tree-1.hs

- 型別 → 用途 → 測試 → 策略 → 定義 → 執行 (Felleisen et al., 2018)
- 先盡量把 *sumTree* 跟 *productTree* 寫得相似，
  然後才把它們抽象成更一般的、可重複利用的模組

解譯器 Arith-1.hs

- 隨機測試、property-based testing (Claessen and Hughes, 2000)
- 進階練習：定義變數 Arith-2.hs

個別的副作用

基本上副作用就是一段程式除了把傳進來的引數變成傳回去的結果以外做的事情。

我們寫程式有時候會直觀想要使用副作用。

最原始的、印象中最常想到的副作用是 state（狀態）：

$$result := 0$$
$$sumTree\ (Leaf\ n) \qquad = result := result + n;$$
$$\qquad\qquad\qquad\qquad result$$
$$sumTree\ (Branch\ t1\ t2) = sumTree\ t1;$$
$$\qquad\qquad\qquad\qquad sumTree\ t2$$

TreeState-1.hs 用 *sumTree'* 定義 *sumTree*

State threading

```
next := 0
relabel (Leaf _)        = next := next + 1;
                          Leaf next
relabel (Branch t1 t2)  = Branch (relabel t1) (relabel t2)
```

TreeState-2.hs 用 *relabel′* 定義 *relabel*

```
seen := S.empty
unique (Leaf n)         = if S.member n seen then False
                          else seen := S.insert n seen; True
unique (Branch t1 t2)   = unique t1 && unique t2
```

用 *unique′* 定義 *unique* （其實也可以用 *unique″* 定義 *unique*，那是比較不副作用、比較能平行化的作法）

3

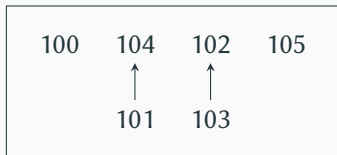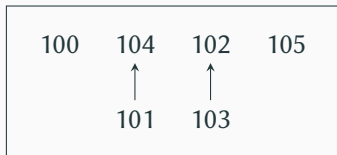把心目中的願望講出來，以便實現。所以如果心目中要的是副作用的話，就把副作用的意義講出來。

Local vs global state

UnionFind-1.hs

```
testState :: State
testState = M.fromList
  [ (Key 100, Root 0 "A")
  , (Key 101, Link (Key 104))
  , (Key 102, Root 1 "C")
  , (Key 103, Link (Key 102))
  , (Key 104, Root 1 "E")
  , (Key 105, Root 0 "F") ]

fresh  :: Info → Key
find   :: Key → (Key, Rank, Info)
union  :: Key → Key → ()
```

Pointers, references, file system

```
┌─────────────────────────────────┐
│  100    104    102    105        │
│          ↑      ↑                │
│          ┆      ┆                │
│         101    103               │
└─────────────────────────────────┘
```

Local vs global state

UnionFind-1.hs

Pointers, references, file system

```
testState :: State
testState = M.fromList
  [ ( Key 100, Root 0 "A")
  , ( Key 101, Link ( Key 104))
  , ( Key 102, Root 1 "C")
  , ( Key 103, Link ( Key 102))
  , ( Key 104, Root 1 "E")
  , ( Key 105, Root 0 "F") ]
```

```
100    104    102    105
        ↑      ↑
       101    103
```

```
fresh  :: Info → State → ( Key, State)
find   :: Key → State → ( Key, Rank, Info, State)
union :: Key → Key → State → State
```
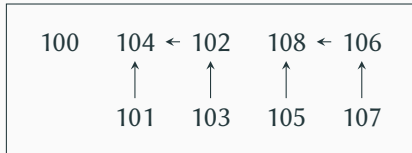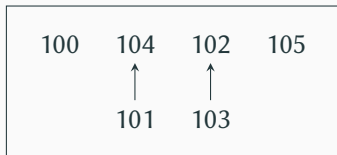
4

Local vs global state

UnionFind-1.hs

```
testState :: State
testState′ :: State
testState′ = M.fromList
  [ (Key 100, Root 0 "A")
  , (Key 101, Link (Key 104))
  , (Key 102, Link (Key 104))
  , (Key 103, Link (Key 102))
  , (Key 104, Root 2 "E")
  , (Key 105, Link (Key 108))
  , (Key 106, Link (Key 108))
  , (Key 107, Link (Key 106))
  , (Key 108, Root 2 "I") ]
```

Pointers, references, file system

4

ArithState-1.hs

進階練習：調撥記憶體 ArithState-2.hs

data *Expr = Lit Int | Add Expr Expr | Mul Expr Expr*
    | New Expr | Get Expr | Put Expr Expr

type *State = [Int]*

這怎麼會有用？

Exception (*Maybe*)

把中途跳脫的意義講出來

> **data** *Maybe a = Nothing | Just a*

TreeMaybe-1.hs

- *decTree* 碰到非正數是錯誤
- *productTree* 碰到零有捷徑

ArithMaybe-1.hs

- 除以零是錯誤

正常狀況下產生的 *Just* 需要 "threading"

每個數字遇到時都可以選擇要或是不要，但是一旦超過 21 就爆掉。
最後得分有哪些可能？

$$11, -1, 11 \quad \rightarrow \quad \{-1, 0, 10, 11, 21\}$$

TreeNondet-1.hs

```
blackjack′ :: Tree → Int → [ Int ]
blackjack′ ( Leaf n )      total = if total + n > 21 then total
                                    else amb [ total, total + n ]
blackjack′ ( Branch t1 t2 ) total = blackjack′ t2 ( blackjack′ t1 total )
```

用 blackjack′ 定義 blackjack

Nondeterminism

References i

Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *ICFP '00: Proceedings of the ACM International Conference on Functional Programming* (Montréal, Québec, Canada) *(ACM SIGPLAN Notices, Vol. 35(9))*. ACM Press, New York, 268–279. https://doi.org/10.1145/351240.351266

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2018. *How to Design Programs* (second ed.). MIT Press, Cambridge. http://www.htdp.org/2018-01-06/Book/