Monad 與副作用

單中杰

2022-08

暖身

純遞迴 `Tree-1.hs`

- 型別 → 用途 → 範例 → 策略 → 定義 → 測試 (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

Accumulator passing

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)        = result := result + n;
                          result
sumTree (Branch t1 t2) = sumTree t1;
                          sumTree t2
```

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

State threading

$$
\begin{aligned}
&\mathit{next} := 0 \\
&\mathit{relabel}\ (\mathit{Leaf}\ \_) &&= \mathit{next} := \mathit{next} + 1; \\
&&&\quad \mathit{Leaf}\ \mathit{next} \\
&\mathit{relabel}\ (\mathit{Branch}\ t1\ t2) &&= \mathit{Branch}\ (\mathit{relabel}\ t1)\ (\mathit{relabel}\ t2)
\end{aligned}
$$

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

$$
\begin{aligned}
&\mathit{seen} := \mathit{S.empty} \\
&\mathit{unique}\ (\mathit{Leaf}\ n) &&= \textbf{if}\ \mathit{S.member}\ n\ \mathit{seen}\ \textbf{then}\ \mathit{False} \\
&&&\quad \textbf{else}\ \mathit{seen} := \mathit{S.insert}\ n\ \mathit{seen};\ \mathit{True} \\
&\mathit{unique}\ (\mathit{Branch}\ t1\ t2) &&= \mathit{unique}\ t1\ \&\&\ \mathit{unique}\ t2
\end{aligned}
$$

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）

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 → ()
```

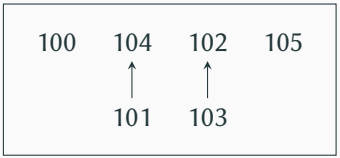
Pointers, references, file system

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") ]
```

Pointers, references, file system



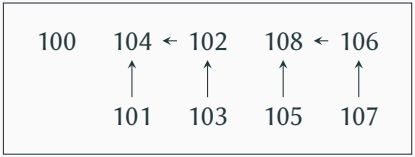

```
fresh :: Info → State → (Key, State)
find  :: Key → State → (Key, Rank, Info, State)
union :: Key → Key → State → State
```

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



State-threading interpreter

`ArithState-1.hs`

進階練習：調撥記憶體 `ArithState-2.hs`

$$\textbf{data}\ \mathit{Expr} = \mathit{Lit}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Expr}\ \mathit{Expr} \mid \mathit{Mul}\ \mathit{Expr}\ \mathit{Expr} \mid \mathit{New}\ \mathit{Expr} \mid \mathit{Get}\ \mathit{Expr} \mid \mathit{Put}\ \mathit{Expr}\ \mathit{Expr}$$

$$\textbf{type}\ \mathit{State} = [\mathit{Int}]$$

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*

data *Either b a* = *Left b* | *Right 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

正常產生的 *Just* 需要 “threading”

二十一點

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`

$$\begin{aligned}
&blackjack' :: Tree \rightarrow Int \rightarrow [Int] \\
&blackjack'\ (Leaf\ n)\ \ total = \mathbf{if}\ total + n > 21\ \mathbf{then}\ total \\
&\qquad\qquad\qquad\qquad\qquad\ \mathbf{else}\ amb\ [total, total + n] \\
&blackjack'\ (Branch\ t1\ t2)\ total = blackjack'\ t2\ (blackjack'\ t1\ total)
\end{aligned}$$

用 *blackjack′* 定義 *blackjack*

$$\begin{aligned}
&concatMap :: (a \rightarrow [b]) \rightarrow [a] \rightarrow [b] \\
&concatMap\ f\ as = concat\ (map\ f\ as)
\end{aligned}$$

Nondeterminism

覆面算

$$\begin{array}{r} X^2 \\ +\ Y^2 \\ \hline Z^2 \end{array} \qquad \begin{array}{r} \mathtt{SEND} \\ +\ \mathtt{MORE} \\ \hline \mathtt{MONEY} \end{array} \qquad \begin{array}{r} \mathtt{TO} \\ +\ \mathtt{GO} \\ \hline \mathtt{OUT} \end{array}$$

$$\mathit{concatMap} :: (a \to [b]) \to [a] \to [b]$$

$$\begin{aligned} &\mathit{concatMap}\ (\lambda x \to \mathit{concatMap}\ (\lambda y \to \mathit{concatMap}\ (\lambda z \to \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{if}\ x^2 + y^2 == z^2 \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{then}\ [\,(x, y, z)\,] \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else}\ [\,]) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [0\,..\,9])) \\ &\qquad\qquad\qquad\qquad\qquad\qquad [0\,..\,9]) \\ &\qquad\qquad\qquad [0\,..\,9] \end{aligned}$$

覆面算

$$\begin{array}{r} X^2 \\ +\ Y^2 \\ \hline Z^2 \end{array} \qquad \begin{array}{r} \texttt{SEND} \\ +\ \texttt{MORE} \\ \hline \texttt{MONEY} \end{array} \qquad \begin{array}{r} \texttt{TO} \\ +\ \texttt{GO} \\ \hline \texttt{OUT} \end{array}$$

type *Digit* = *Int*
digit :: (*Digit* → [*Answer*]) → *Answer*
digit (λ*x* → *digit* (λ*y* → *digit* (λ*z* → **if** $x^2 + y^2 == z^2$
then [(*x*, *y*, *z*)]
else [])))

可以把每一個 loop body 想成一個 *Digit* 的 continuation
所以「*digit* (λ*x* →」好像一個命令

覆面算

$$\begin{array}{r} X^2 \\ +\ Y^2 \\ \hline Z^2 \end{array} \qquad \begin{array}{r} \texttt{SEND} \\ +\ \texttt{MORE} \\ \hline \texttt{MONEY} \end{array} \qquad \begin{array}{r} \texttt{TO} \\ +\ \texttt{GO} \\ \hline \texttt{OUT} \end{array}$$

$$\begin{aligned} &\mathit{concatMap}\ (\lambda d \to \mathit{concatMap}\ (\lambda e \to \mathit{concatMap}\ (\lambda y \to \\ &\qquad \textbf{if}\ \mathit{mod}\ (d + e)\ 10 == y \\ &\qquad \textbf{then}\ \ldots \\ &\qquad \textbf{else}\ [\,]) \\ &\qquad ([0\,..\,9] \setminus\!\setminus [d, e])) \\ &\qquad ([0\,..\,9] \setminus\!\setminus [d])) \\ &[0\,..\,9] \end{aligned}$$

趁早檢查，免得做白工

覆面算

```
  X²          SEND        TO
+ Y²        + MORE      + GO
-----       ------      ----
  Z²         MONEY       OUT
```

```
type Chosen = [Digit]
digit :: (Digit → Chosen → [Answer]) → Chosen → [Answer]
digit (λd → digit (λe → digit (λy → if mod (d + e) 10 == y
                                    then ...
                                    else λ chosen → [])))
```

Crypta-1.hs

覆面算

$$\begin{array}{r} X^2 \\ +\ Y^2 \\ \hline Z^2 \end{array} \qquad \begin{array}{r} \texttt{SEND} \\ +\ \texttt{MORE} \\ \hline \texttt{MONEY} \end{array} \qquad \begin{array}{r} \texttt{TO} \\ +\ \texttt{GO} \\ \hline \texttt{OUT} \end{array}$$

type $Chosen = [\,(Char, Digit)\,]$
$digit :: Char \to (Digit \to Chosen \to [\,Answer\,]) \to Chosen \to [\,Answer\,]$
add `'D' 'E' 'Y' ...`

`Crypta-2.hs` 適合自資料檔讀取新題

Nondeterministic interpreter

`ArithNondet-1.hs`

data *Expr* = ⋯ | *Amb Expr Expr*

(McCarthy, 1963)

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/

John McCarthy. 1963. A Basis for a Mathematical Theory of Computation. In *Computer Programming and Formal Systems*, P. Braffort and D. Hirschberg (Eds.). Number 35 in Studies in Logic and the Foundations of Mathematics. Elsevier Science, Amsterdam, 33–70. http://jmc.stanford.edu/articles/basis/basis.pdf