How FP Deals With Effects

PoolC 양제성

Source: 2024/01/24 (Wed)

목차

Overall Structure

FP is all about **composing pure functions**.

```
int main(void) {
  f(); g(); h(); ..
}
```

VS

```
f(g(h(..)))
```

[Procedural Promramming]

[Functional Programming]

Overall Structure

How?

FP is all about **composing pure functions**.

```
int main(void) {
  f(); g(); h(); ..
}
```

VS

```
f(g(h(..)))
```

[Procedural Promramming]

[Functional Programming]

Overall Structure

Sum all. [`stdin <- "5\n1 2 3 4 5"`]

```
int main() {                C++ (imperative)
  int n, result;
  std::cin >> n;
  for (size_t i = 0; i < n; ++i) {
    int a;
    std::cin >> a;
    result += a;
  }
  std::cout << result << '\n';
  return 0;
}
```

```
main =                 Haskell (declarative)
  interact
    (show . sum .
      map read . drop 1 . words)
```

```
                       Python3 (declarative)
from sys import stdin

print(sum(map(
  int, stdin.read().split()[1:]
)))
```

[Procedural Promramming]

[Functional Programming]

Overall Structure

1. Purity
   - Side Effect
   - Referential Transparency
   - Significance of ...

2. Immutability
   - Recursion (feat. Tail Call Optimization)
   - C vs Haskell in File IO

3. First Class Function
   - Currying
   - Linked List

Overall Structure

1. Purity
   - Side Effect
   - Referential Transparency
   - Significance of …
2. Immutability
   - Recursion (feat. Tail Call Optimization)
   - C vs Haskell in File IO
3. First Class Function
   - Currying
   - Linked List

<Let's code!>

Historical Review (CS + Math)

Lambda Calculus

1. Very Basics
2. Boolean in Action

Category Theory

1. Very Basics
2. Functor in Action

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

$\Leftrightarrow$ `Turing Machine`

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

$\Leftrightarrow$ `Turing Machine`

$$\lambda x.fx$$

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

$\Leftrightarrow$ `Turing Machine`

$$\underline{\lambda x.fx}$$

Lambda Abstraction

`JS ver. (x) => f(x)`

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

$\Leftrightarrow$ `Turing Machine`

Function Signifier $\longleftarrow$ $\underbrace{\lambda x . f x}_{\text{Lambda Abstraction}}$

`JS ver. (x) => f(x)`

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

$\Leftrightarrow$ `Turing Machine`

$$\text{Function Signifier} \longleftarrow \underset{\text{Lambda Abstraction}}{\underline{\lambda \overset{\overset{\text{Parameter Variable}}{\uparrow}}{x} . f x}}$$

`JS ver. (x) => f(x)`

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

$\Leftrightarrow$ `Turing Machine`

$$\begin{tikzcd} & \text{Parameter Variable} & \\ \text{Function Signifier} & \underline{\lambda x . f x} \arrow[l] \arrow[u] \arrow[r] & \text{Return Expression} \end{tikzcd}$$

Lambda Abstraction

`JS ver. (x) => f(x)`

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

$\Leftrightarrow$ `Turing Machine`

$$\begin{aligned} \text{expression} ::= \ & \text{variable} && \mathit{identifier} \\ | \ & \text{expression expression} && \mathit{application} \\ | \ & \lambda \ v_1 v_2 \cdots \ . \ \text{expression} && \mathit{abstraction} \\ | \ & (\ \text{expression} \ ) && \mathit{grouping} \end{aligned}$$

Lambda Calculus

Function Encoding

1. Variables (Immutable)
2. Functions (Curried)
3. Application

⇔ Turing Machine

ex) Church Encoding: Boolean JS