函数式编程概念梳理

E. W.

2025 年 2 月 21 日

目录

1 Purity & Side Effects

1.1 The Pros of Purity

- Cachable
- Portable & Self-documenting
- Testable
- Reasonable
- Parallelizable

2 Currying

$$f(x, y, z) \implies f(x)(y)(z) \tag{1}$$

```
const add = x => y => x + y;
const increment = add(1);
const addTen = add(10);

increment(2); // 3
addTen(2); // 12
```

3 Composing

$$f(x) \quad g(x) \implies f(g(x)) \tag{2}$$

Reference Equation (2).

```
const compose = (f, g) => x => f(g(x));

const add1 = x => x + 1;
const mult2 = x => x * 2;

const add1Mult2 = compose(mult2, add1);
add1Mult2(5); // 12
```

Point-free style

```
// not pointfree because we mention the data: word
const snakeCase = word => word.toLowerCase().replace(/\s+/ig, '_');

// pointfree
const snakeCase = compose(replace(/\s+/ig, '_'), toLowerCase);
```

3.1 Category Theory

Definition of Category: A collection with the following components:

- A collection of *Objects*
- A collection of *Morphisms*
- A notion of *Composition* on the morphisms
- A distinguished morphism called *Identity*

In the context of programming, the *Objects* are types, the *Morphisms* are functions, *Composition* is function composition, and the *Identity* is the identity function.

```
// identity
compose(id, f) === compose(f, id) === f;
// true
```

In Haskell, the compose operator is represented by a dot:

```
(.) :: (b -> c) -> (a -> b) -> a -> c
```

4 Hindley-Milner Type System

The explanation of *Free as in Theorem*: Write down the definition of a polymorphic function on a piece of paper. Tell me its type, but be careful not to let me see the function's definition. I will tell you a theorem that the function satisfies.

5 Functor

Definition of Functor: A type that implements `map` and obeys some laws. And the laws are:

1. Identity: `fx.map(x => x) === fx`
2. Composition: `fx.map(f).map(g) === fx.map(x => g(f(x))`

6 Example Figure

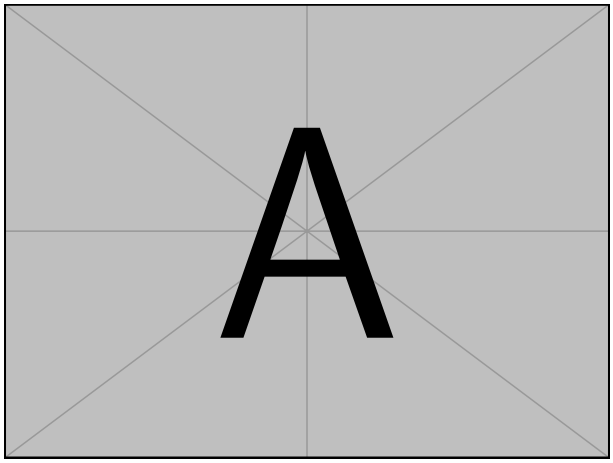

图 1: An example figure.

See Figure 1 for an illustration.