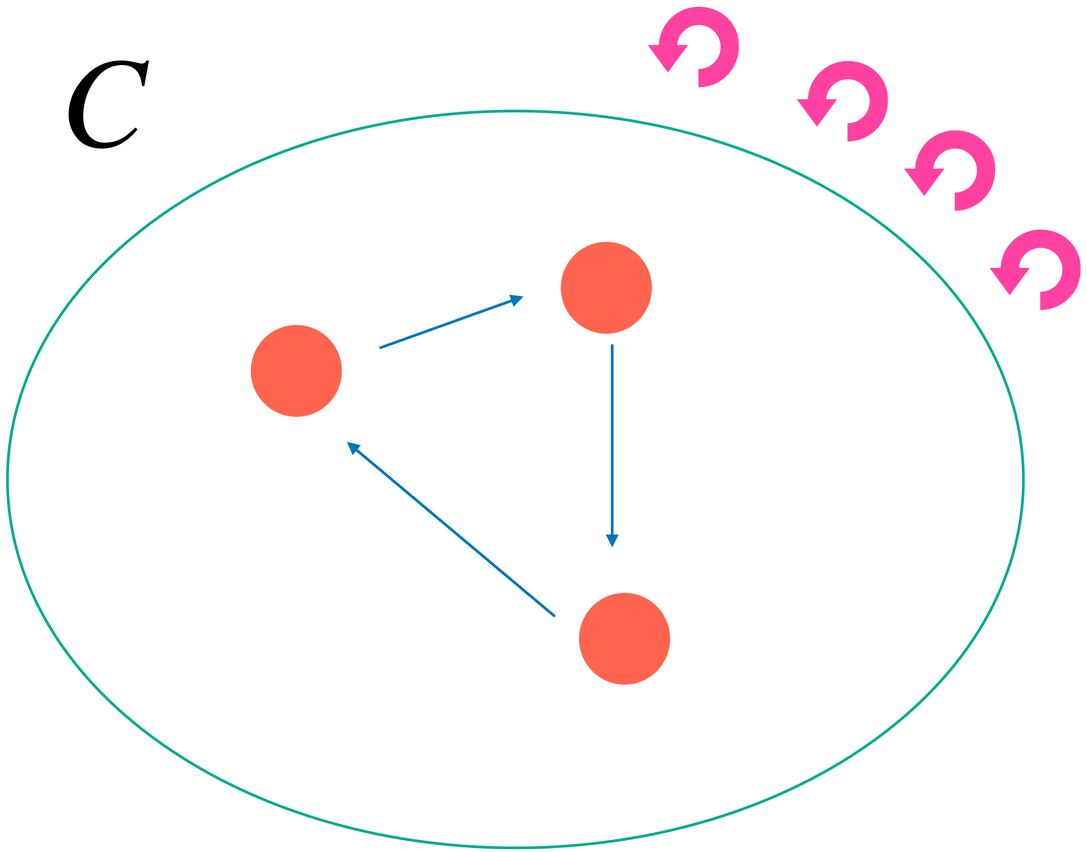
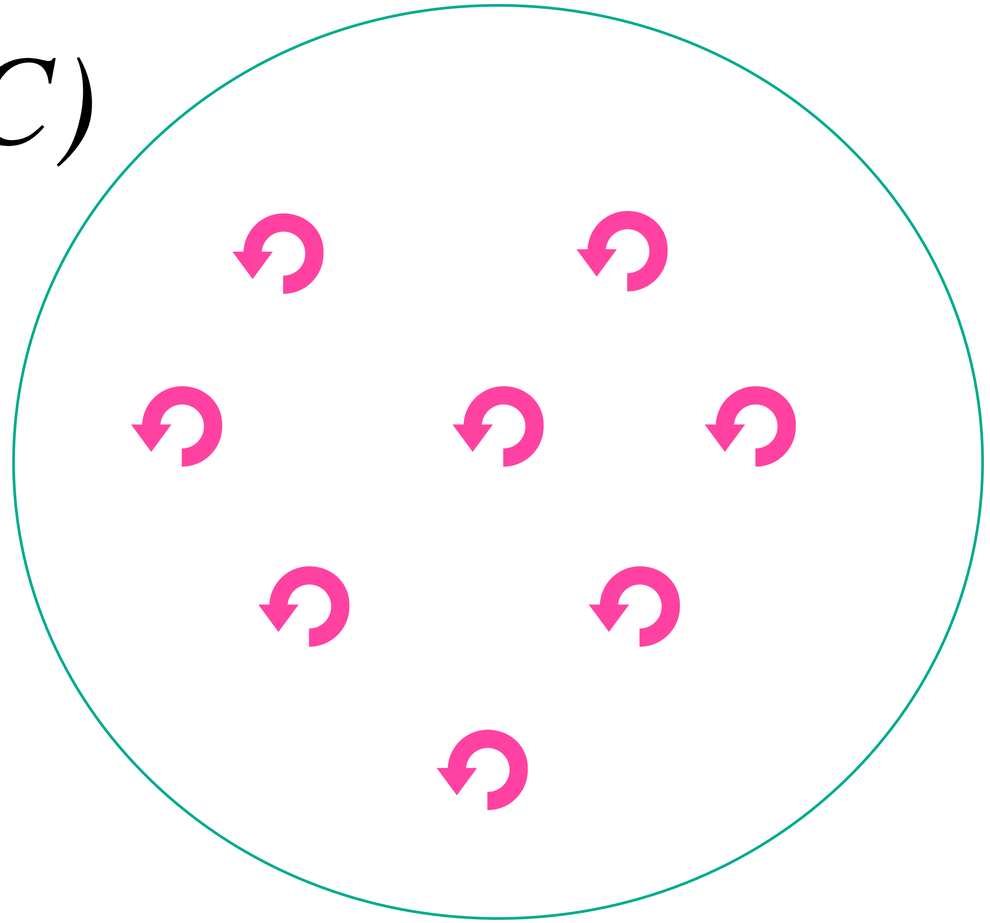
모나드 패턴

2023.09.02

모나드 (범주론)

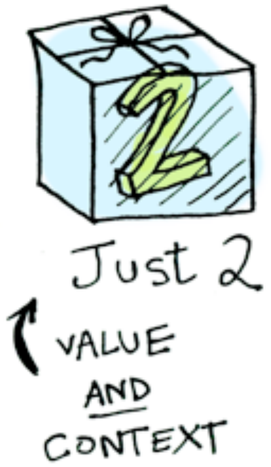
범주의 자기 함자로 이루어진 범주의 모노이드





> The curse of the monad is that once you get the epiphany, once you understand, you lose the ability to explain it to anybody else.
>
> Douglas Crockford – 'Monads & Gonads' at YUIConf 2012

모나드 (프로그래밍)



1. 함수형 프로그래밍에서 사용 되는 패턴이다

2. 특정한 값과 그 값을 감싸는 문맥으로 구성된다

모나드의 인터페이스

```
// Monad Interface
interface Monad<T> {
  unit(val: T): Monad<T>;
  bind<R>(f: (val: T) => Monad<R>): Monad<R>;
}
```

- Unit Function : 특정한 값을 받고 해당 값을 문맥(모나드 구조)로 감싸는 함수
- Bind Function : 모나드를 리턴하는 연산을 인자로 받고, 체이닝을 가능하게 하는 함수

모나드 구현체 (Class)

Maybe 모나드

```typescript
class Maybe<T> implements Monad<T> {
  constructor(private _value: T | null) {}

  public static unit<T>(value: T | null): Maybe<T> {
    return new Maybe(value);
  }

  bind<R>(f: (value: T) => Maybe<R>): Maybe<R> {
    if (this._value === null) {
      return new Maybe<R>(null);
    }
    return f(this._value);
  }

  get value(): T | null {
    return this._value;
  }
}
```

```typescript
const justFive = Maybe.unit(5); // Create Maybe monad with value 5
const none = Maybe.unit(null); // Create Maybe monad with no value

const addOne = (x: number) => Maybe.unit(x + 1);
const multiplyByTwo = (x: number) => Maybe.unit(x * 2);

const result = justFive // Chain operations using bind
  .bind(addOne)
  .bind(multiplyByTwo);

console.log(result.value); // 12
console.log(none.value); // null
```

모나드 법칙

- 좌측 항등법칙
- 우측 항등법칙
- 결합 법칙

- 위의 3가지 법칙을 만족해야 모나드의 정의에 부합한다

좌측 항등법칙

```ts
// Left identity
const a = 1;
const f = (x: number) => Maybe.unit(x * 2);

const left = Maybe.unit(a).bind(f);
const right = f(a);

console.log(left.value); // 2
console.log(right.value); // 2
```

- bind(unit(a), f) == f(a)
- unit 함수를 사용해서 모나드를 생성하고, f를 bind 한 결과와 f에 인자를 넣어서 바로 연산한 것과 값이 동일 해야함

우측 항등법칙

```
// Right identity
const m = Maybe.unit(1);

const left = m.bind(Maybe.unit);
const right = m;

console.log(left.value); // 1
console.log(right.value); // 1
```

- bind(m, unit) == m
- 모나딕한 값이 있고 이를 unit 함수에 전달하기 위해 bind를 사용하면 결과값도 동일한 값이어야한다

결합법칙

```ts
// Associativity
const m = Maybe.unit(1);

const f = (x: number) => Maybe.unit(x + 1);
const g = (x: number) => Maybe.unit(x * 2);

const left = m.bind(f).bind(g);
const right = m.bind((x) => f(x).bind(g));

console.log(left.value); // 4
console.log(right.value); // 4
```

- bind(bind(m, f), g) == bind(m, x => bind(f(x), g))
- bind에 어떻게 함수들 전달 하던지 결과 값이 동일해야한다

그런데 왜 쓰는거죠?

모나드를 쓰는 이유

- 공통 :
  값을 모나드로 감싸고, 여러 연산을 bind 해서 깔끔하게 pipeline을 작성

- 함수형 프로그래밍 :
  사이드 이팩트를 관리하기 위해서 사용한다. haskell 같이 함수형 프로그래밍을 강제하는 언어에서 I/O, 에러 처리 등을 구현하기 위해서 사용된다.

- 일반적인 프로그래밍 :
  보일러 플레이트를 줄여서, 선언적인 코드를 작성할 수 있다.

Array는 모나딕한 값이다

```ts
// Array as a monad
const unit = <T>(x: T) => [x];
const bind = <T>(arr: T[], f: (el: T) => T[]) => arr.flatMap(f);

// unit will wrap a value into an array
const a = unit(42);

// bind will apply a function to each element and return a new array
const arr = [1, 2, 3];
const f = (x: number) => [x * x];

const result = bind(arr, f);
console.log(result); // [1, 4, 9]
```

- 배열 선언을 unit function으로 볼 수 있다
- flatMap을 bind function으로 볼 수 있다

Array 모나드 법칙

```ts
// left identity
const x = 42;
const g = (x: number) => [x + 1];

const left = bind(unit(x), g);
const right = g(x);

console.log(left, right); // [43] [43]
```

```ts
// right identity
const m = [42];

const left = m.flatMap(unit);
const right = m;

console.log(left, right); // [42] [42]
```

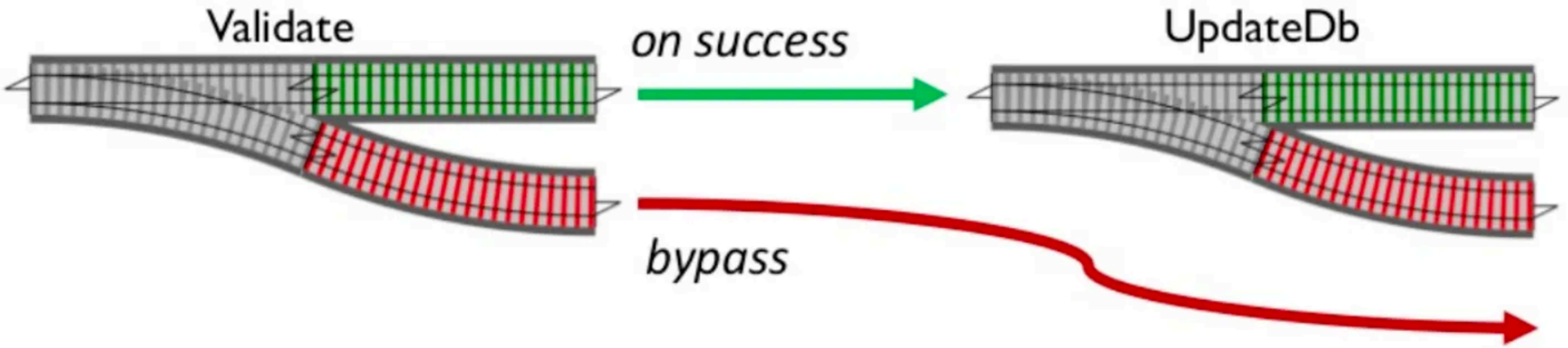
```ts
// associativity
const m = unit(42);
const f = (x: number) => [x + 1];
const g = (x: number) => [x * x];

const left = bind(bind(m, f), g);
const right = bind(m, (x) => bind(f(x), g));

console.log(left, right); // [1849] [1849]
```

Promise도 모나딕 한 값이다

```ts
// Promise as a monad
const unit = <T>(x: T) => Promise.resolve(x);
const bind = <T>(promise: Promise<T>, f: (el: T) => Promise<T>) =>
  promise.then(f);

export { unit, bind };

// unit function used to wrap a value into a monad
const a = Promise.resolve(42); // Promise(42)

// bind function used to provide operations on monads
const p = Promise.resolve(42);
const f = (x: number) => Promise.resolve(x + 1);

const res = bind(p, f); // Promise(43)
```

- Promise construction을 unit function으로 볼수 있다
- then을 bind function으로 볼 수 있다

분량 조절 실패...



- 다음 시간에 Railway-Oriented Programming에서 어떻게 모나드를 사용하는지 설명