Functor Zoo

Programming with functors in Python

Chaur Wu

Conf42, 2024

About this talk

- For Python programmers
- No prior knowledge of category theory required
- Ease of understanding over math rigor
- Examples are based on the funclift package

https://github.com/essentier/funclift

https://github.com/essentier/funclift-tutorials

About me

Chaur Wu (吳嘉二)

Software developer for over 20 years

Grew up in Taiwan

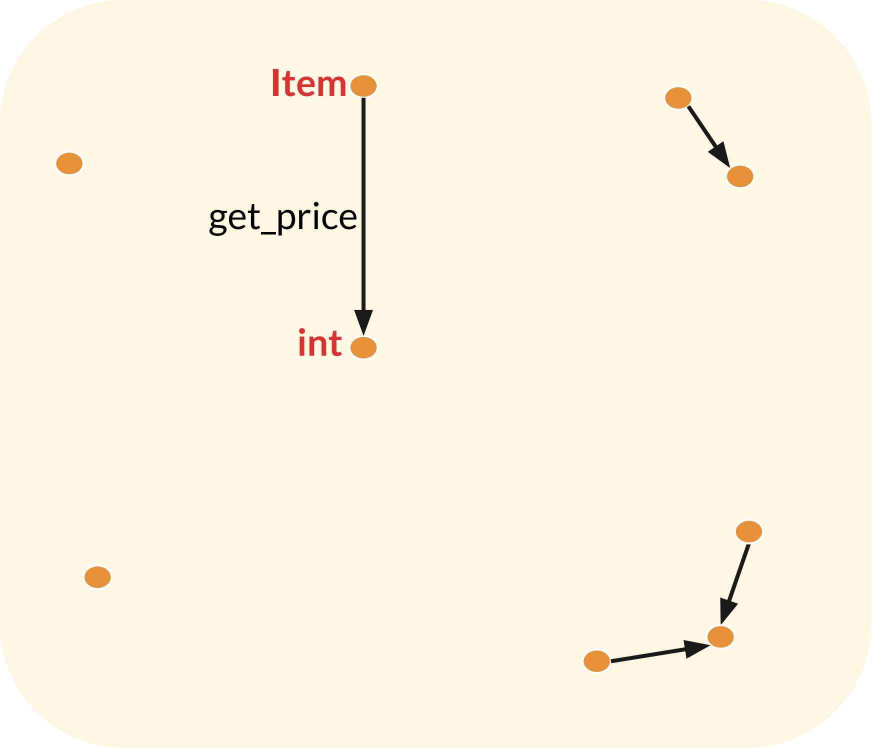
Based in the San Francisco bay area for the past 20 years

Agenda

- Motivating example
- Various functors and examples

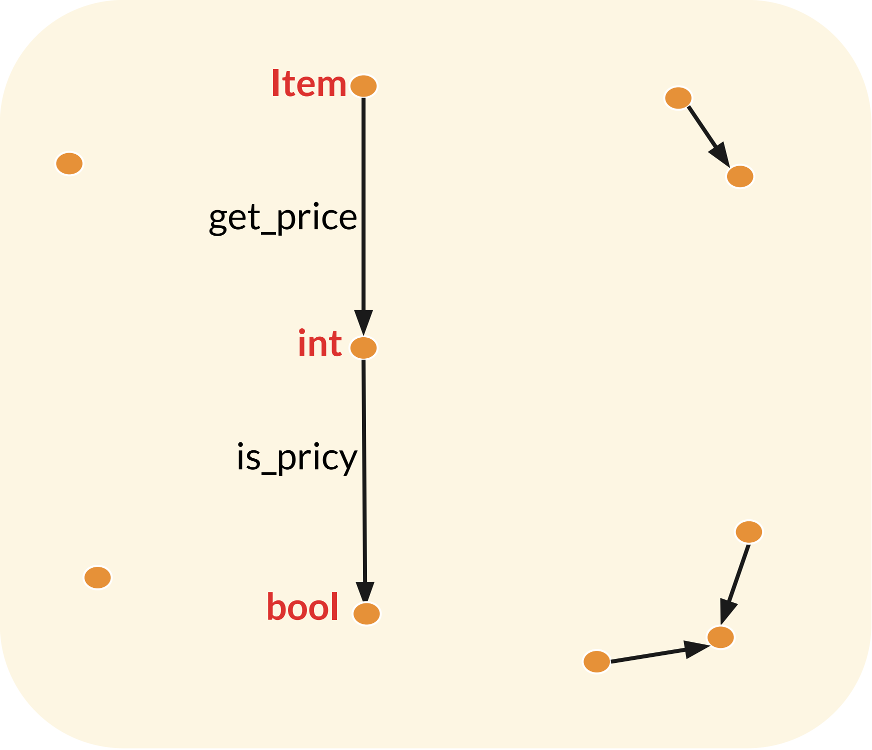
A function

```
def get_price(item: Item) -> int:
    return item.price
```



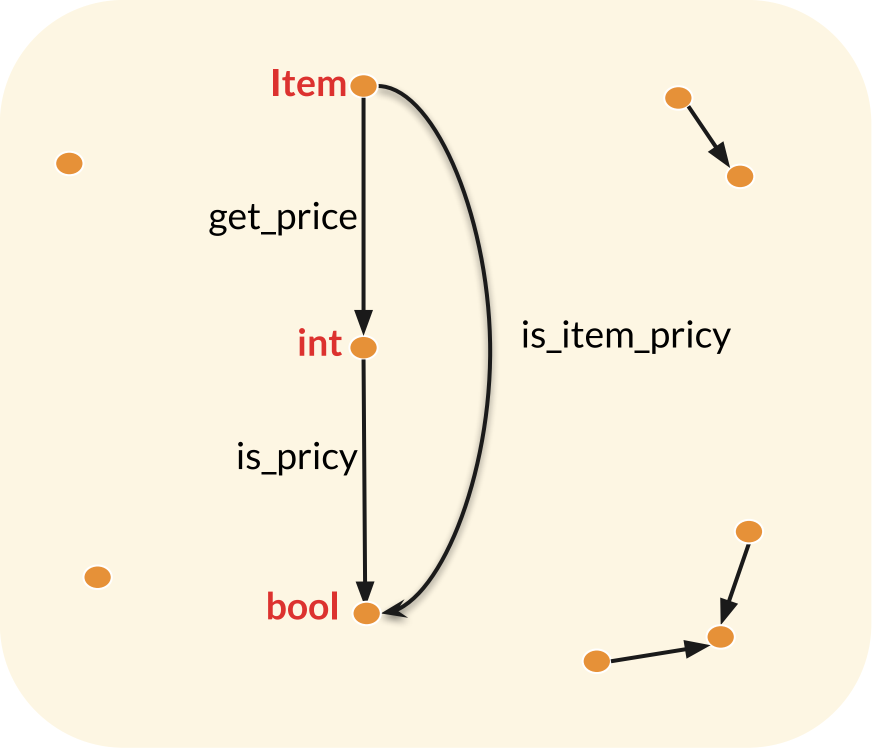
Another function

```
def get_price(item: Item) -> int:
    return item.price

def is_pricy(n: int) -> bool:
    return n > 100
```



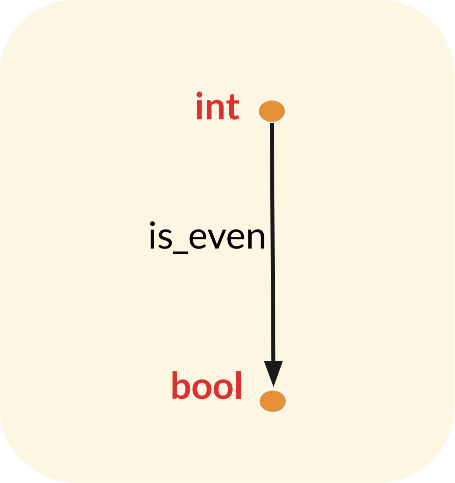
Category of types (T)

```
def get_price(item: Item) -> int:
    return item.price


def is_pricy(n: int) -> bool:
    return n > 100


def is_item_pricy(item: Item) -> bool:
    return is_pricy(get_price(item))
```



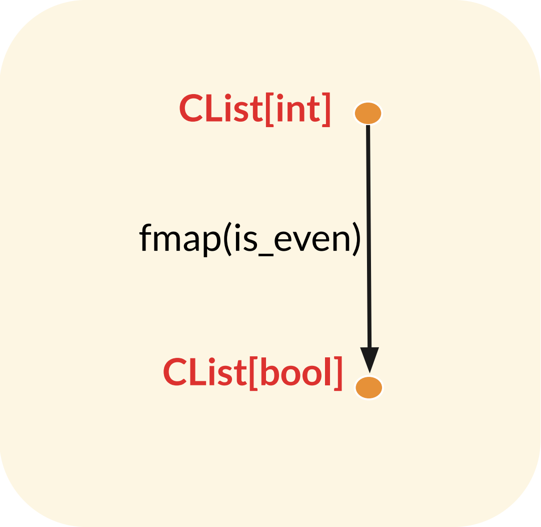
List of numbers

```
def is_even(n: int) -> bool:
    return n % 2 == 0


nums = [1, 2, 3, 4]
nums_even = [ is_even(n) for n in nums]
```

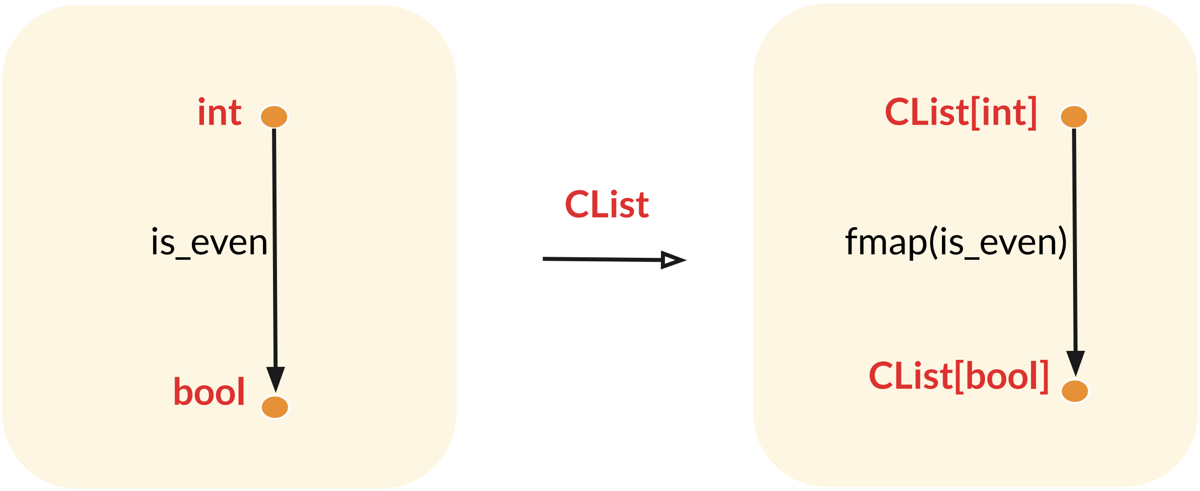


CList of numbers

```
from funclift.types.clist import CList

nums = CList([1, 2, 3, 4])
nums_even = nums.fmap(is_even)



# In Python, nums.fmap(is_even) is actually
fmap(self, is_even), which is equivalent to
(fmap(is_even))(nums)
```



CList is a functor



- A functor is a mapping between a source category and a target category.
- If the source and target categories are the same, the functor is called an endo-functor.
- CList is an endo-functor.

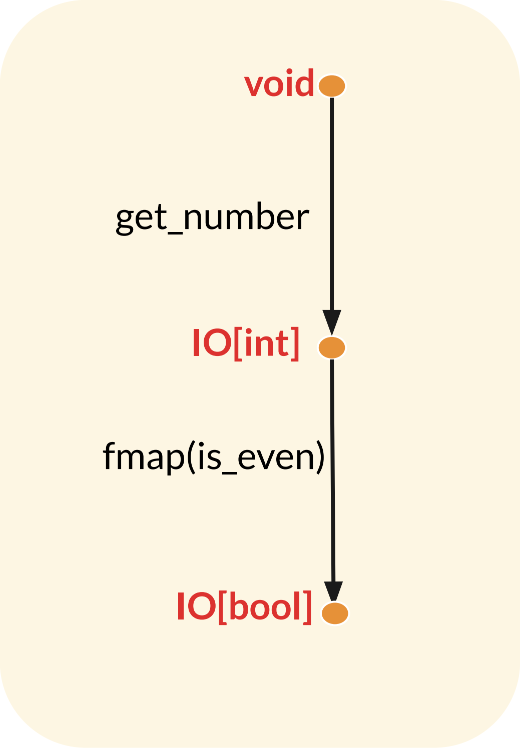
IO side effects

```python
def get_number_side_effect() -> int:
    return int(input('enter a number: '))
```

```python
from funclift.types.io import IO

def get_number() -> IO[int]:
    return IO(lambda: int(input('enter a number: ')))
```

IO functor

```
def is_even(n: int) -> bool:
  return n % 2 == 0

num = get_number()
num_even = num.fmap(is_even)
num_even.unsafe_run()
```



Partial function

```
def ten_mod_by(n: int) -> int:
  return 10 % n
```

```
def ten_mod_by(n: int) -> int | None:
  return None if n == 0 else (10 % n)
```

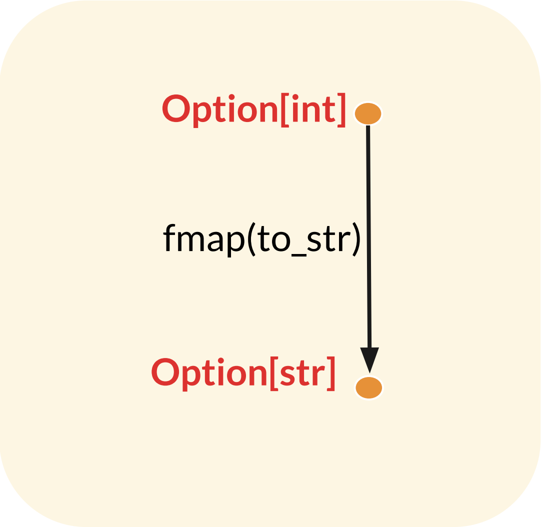
Not very composable

```
def ten_mod_by(n: int) -> int | None:
    return None if n == 0 else (10 % n)


def to_str(r: int) -> str:
    return 'remainder is ' + str(r)
```

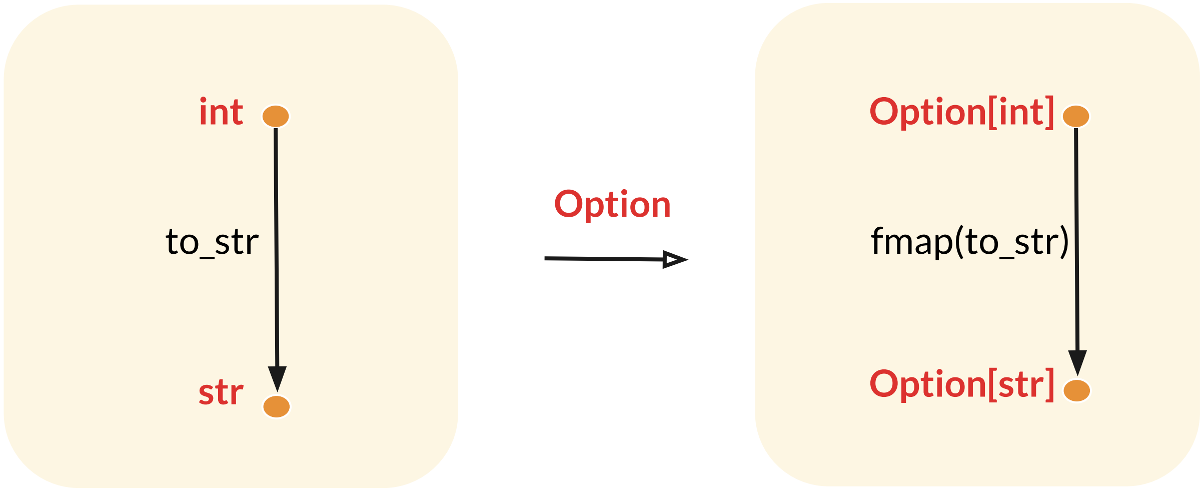
```
def ten_mod_by_in_text(x: int) -> str | None:
    r = ten_mod_by(x)
    if r:
        return to_str(r)
    else:
        return None
```

Option for better composability

```
from funclift.types.option import Option, Nothing, Some

def ten_mod_by(n: int) -> Option[int]:
  return Nothing() if n == 0 else Some(10 % n)

def to_str(r: int) -> str:
  return 'remainder is ' + str(r)

def ten_mod_by_in_text(x: int) -> Option[str]:
  r = ten_mod_by(x)
  return r.fmap(to_str)
```



Option is a functor



int

to_str

str

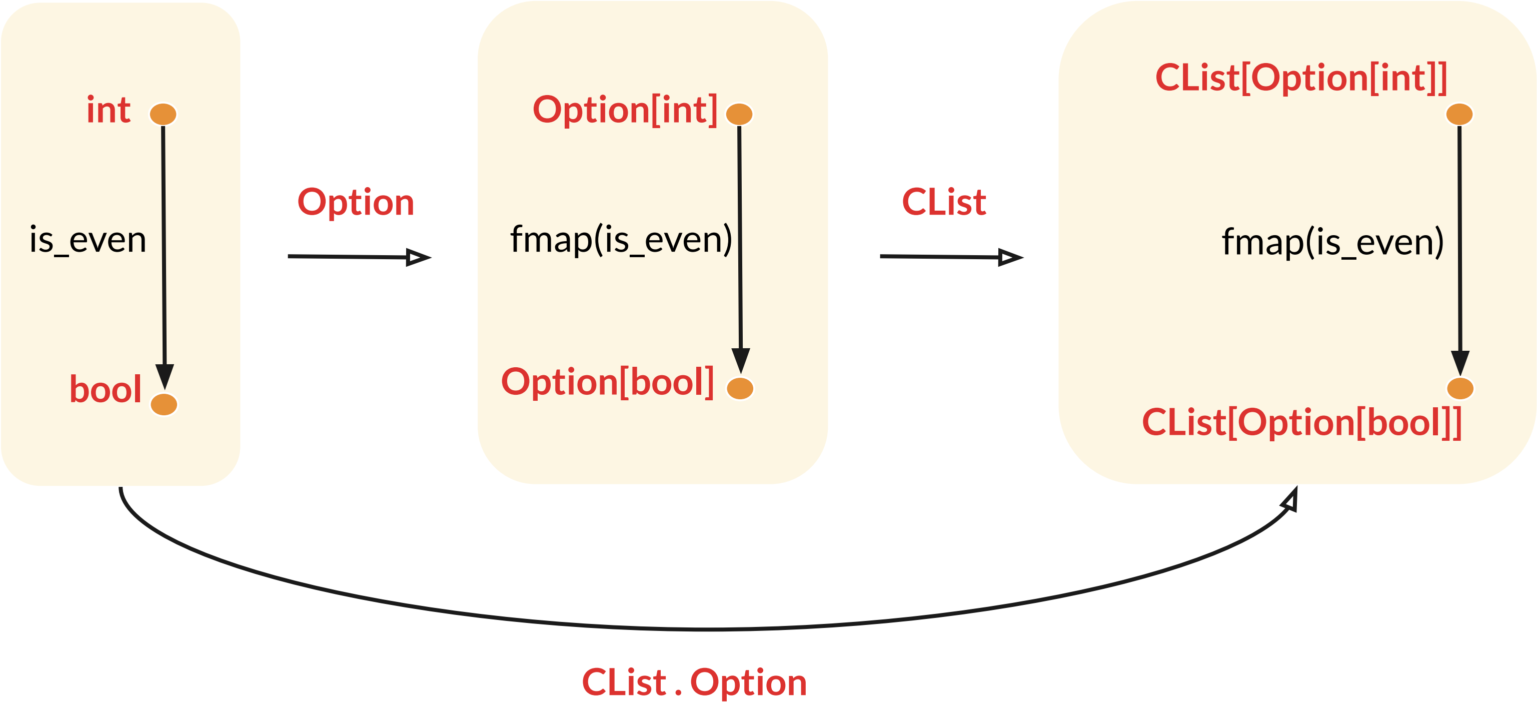
Option

Option[int]

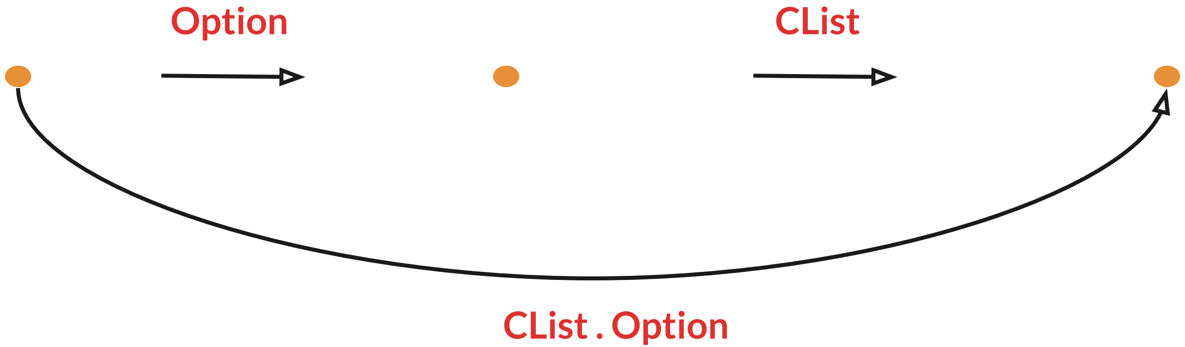
fmap(to_str)

Option[str]

Composing functors



Category of small categories



Option

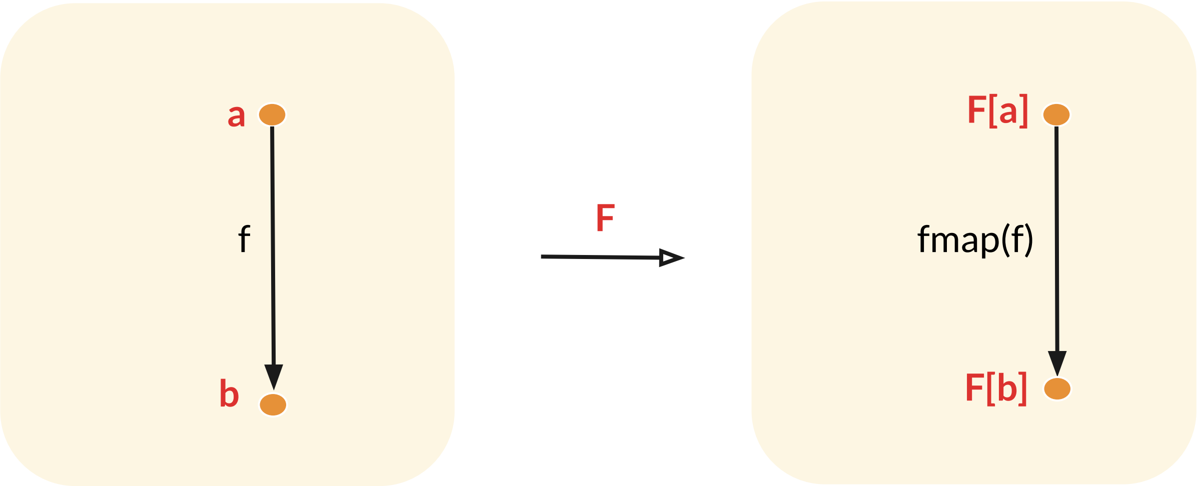
CList

CList . Option

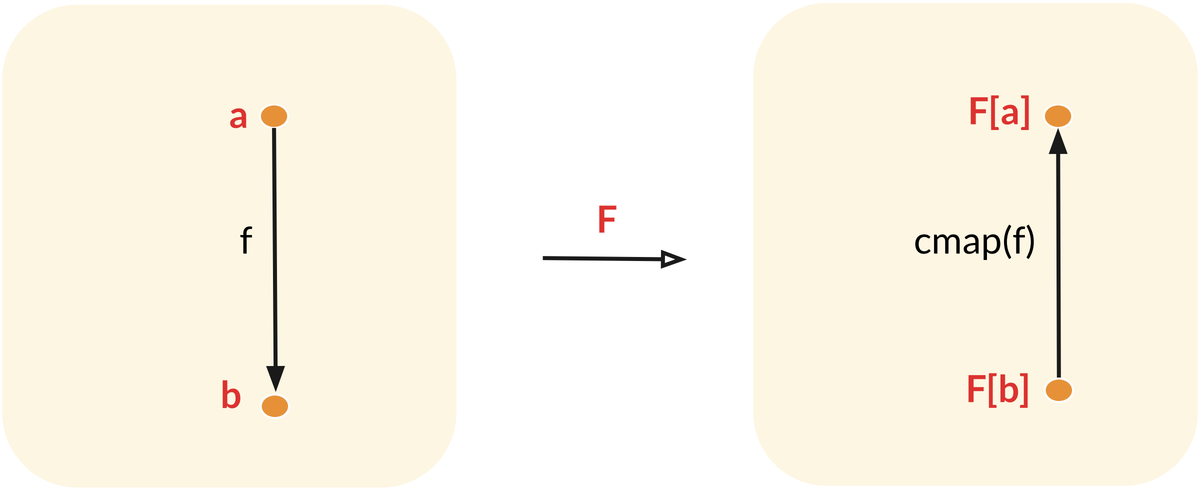
Example of CList . Option

```
from funclift.types.compose import Compose

def is_even(n: int) -> bool:
    return n % 2 == 0

def add3(n: int) -> int:
  return n + 3

nums = CList( [Some(1), Nothing(), Some(2)] )
composite = Compose(nums)
result = composite.fmap(add3).fmap(is_even)
assert result == Compose(CList([Some(True), Nothing(), Some(False)]))
```

Covariant Functor in general

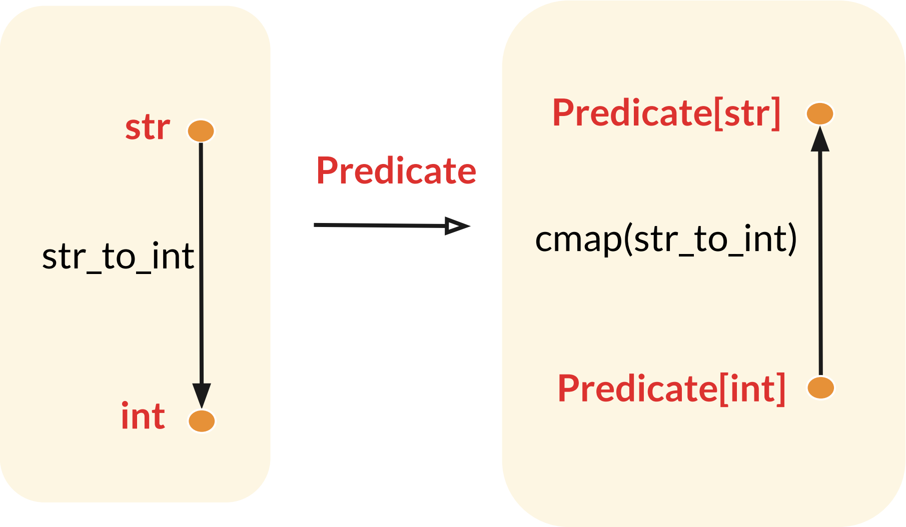


Contravariant Functor

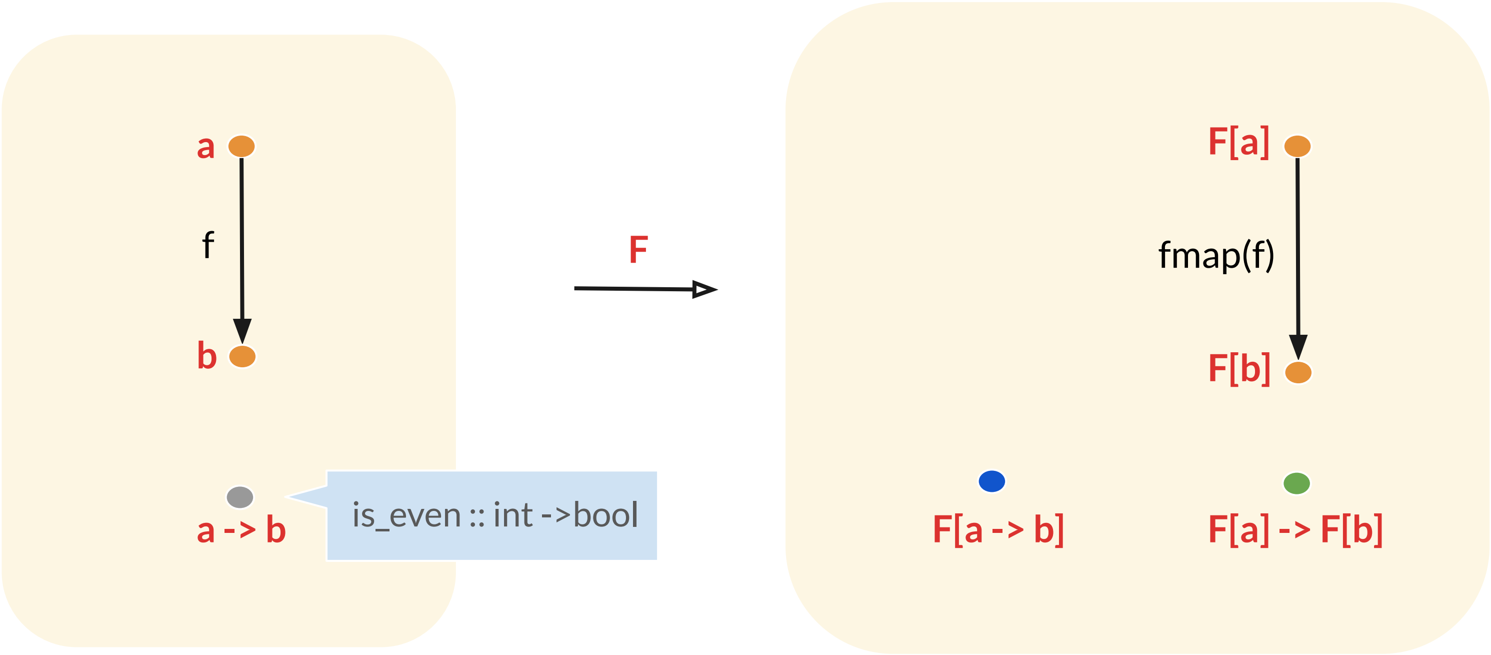


Predicate is a contravariant functor

```
from funclift.types.predicate import Predicate


def is_even(n: int) -> bool:
   return n % 2 == 0


def str_to_int(text: str) -> int:
   return int(text)


is_int_even = Predicate(is_even)
is_int_even(6)
is_str_even = is_int_even.cmap(str_to_int)
is_str_even('6')
```



closed category



a

f

b

a -> b

is_even :: int ->bool

F

F[a]

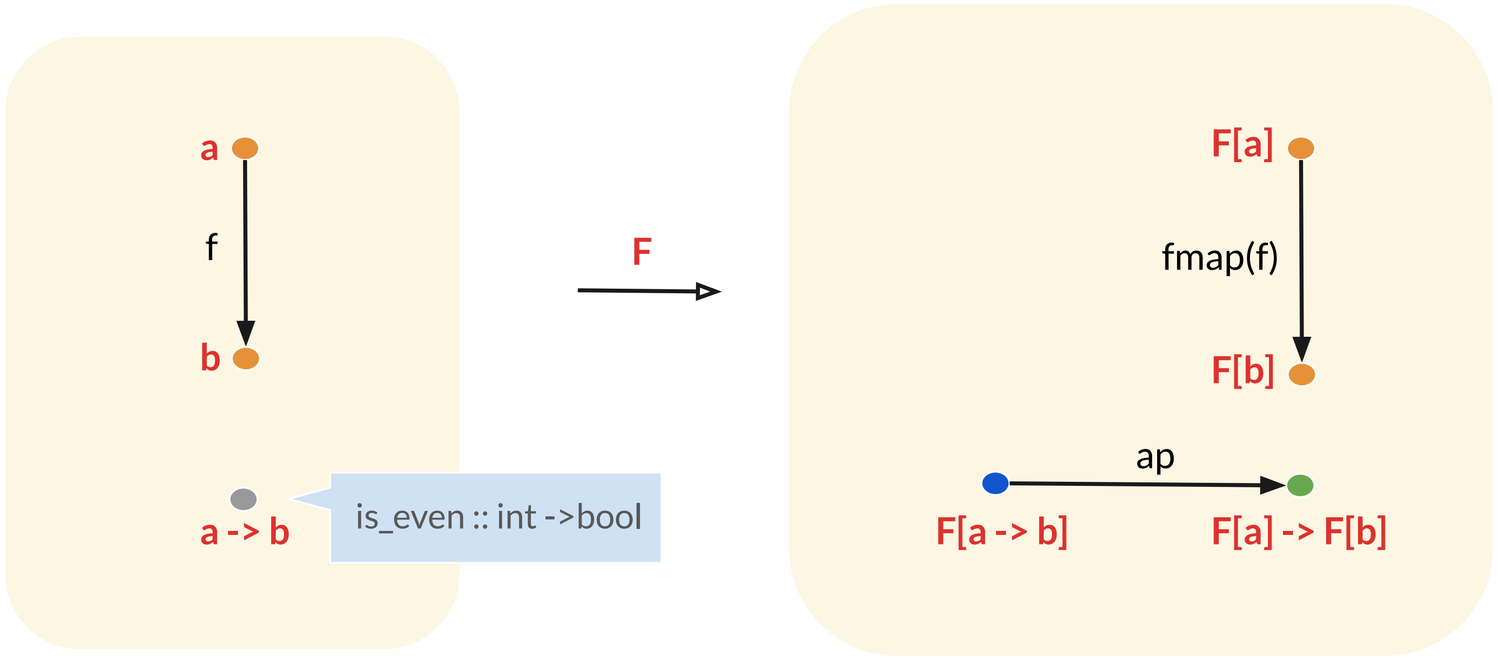
fmap(f)

F[b]

F[a -> b]

F[a] -> F[b]

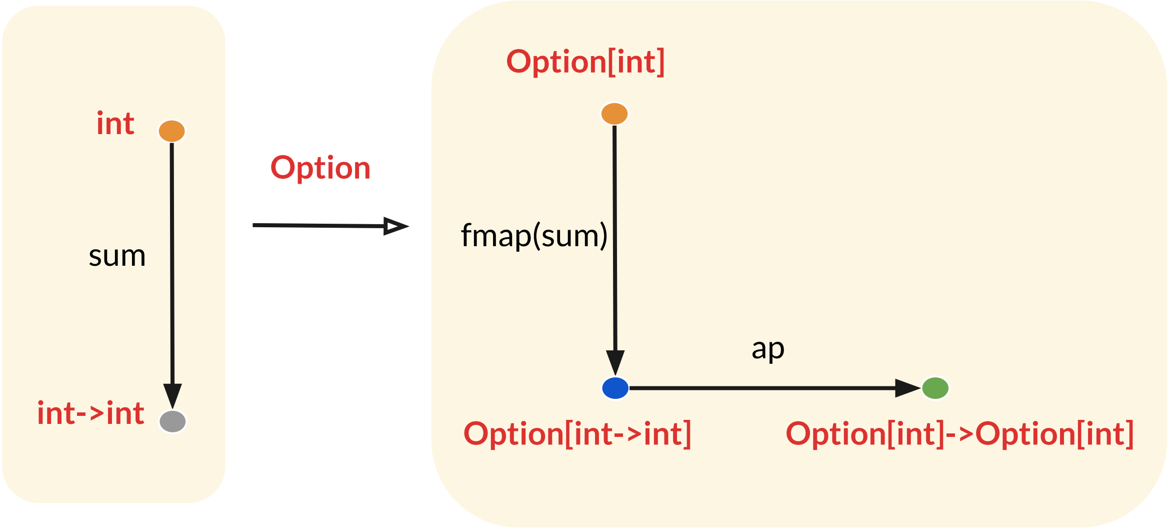
Applicative functor is lax closed functor



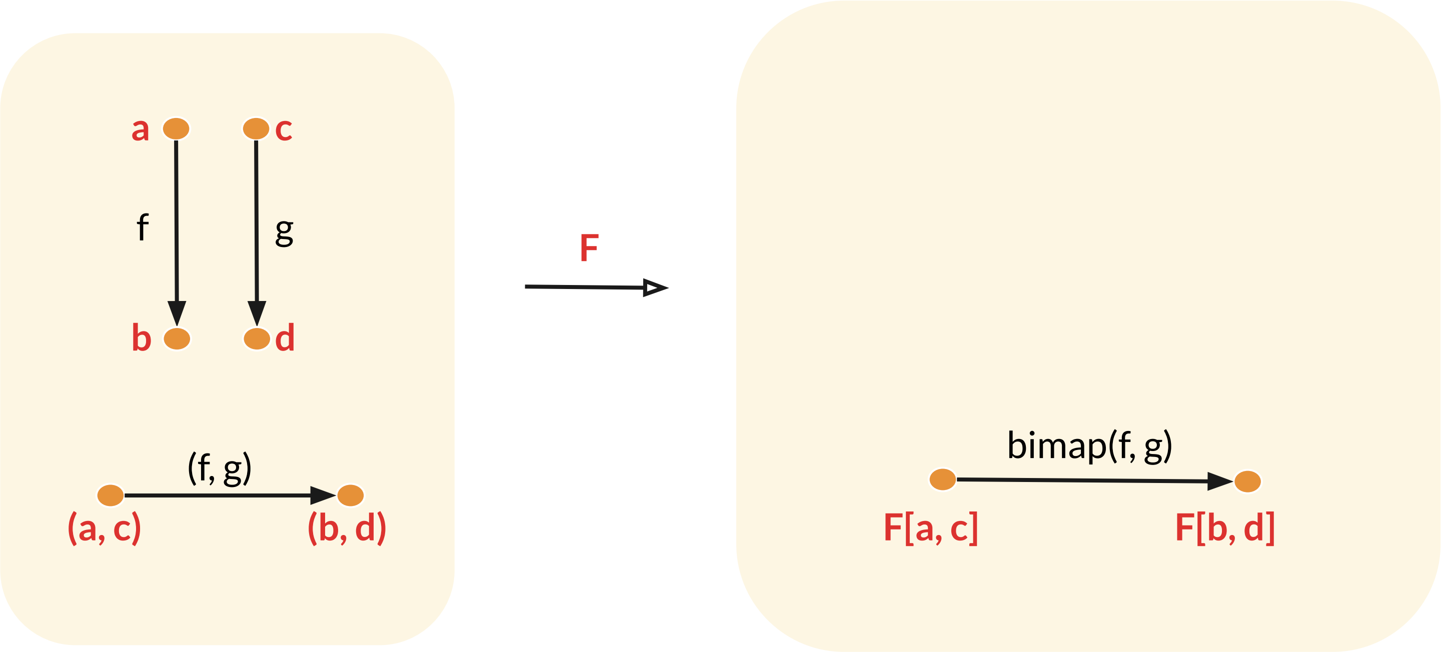
is_even :: int ->bool

Example of Applicative functor

```
@curry
def sum(a: int, b: int) -> int:
  return a + b

Some(20) \
   .fmap(sum) \
   .ap(Some(30))


# @curry turns (int, int) -> int
into int -> (int -> int)
```



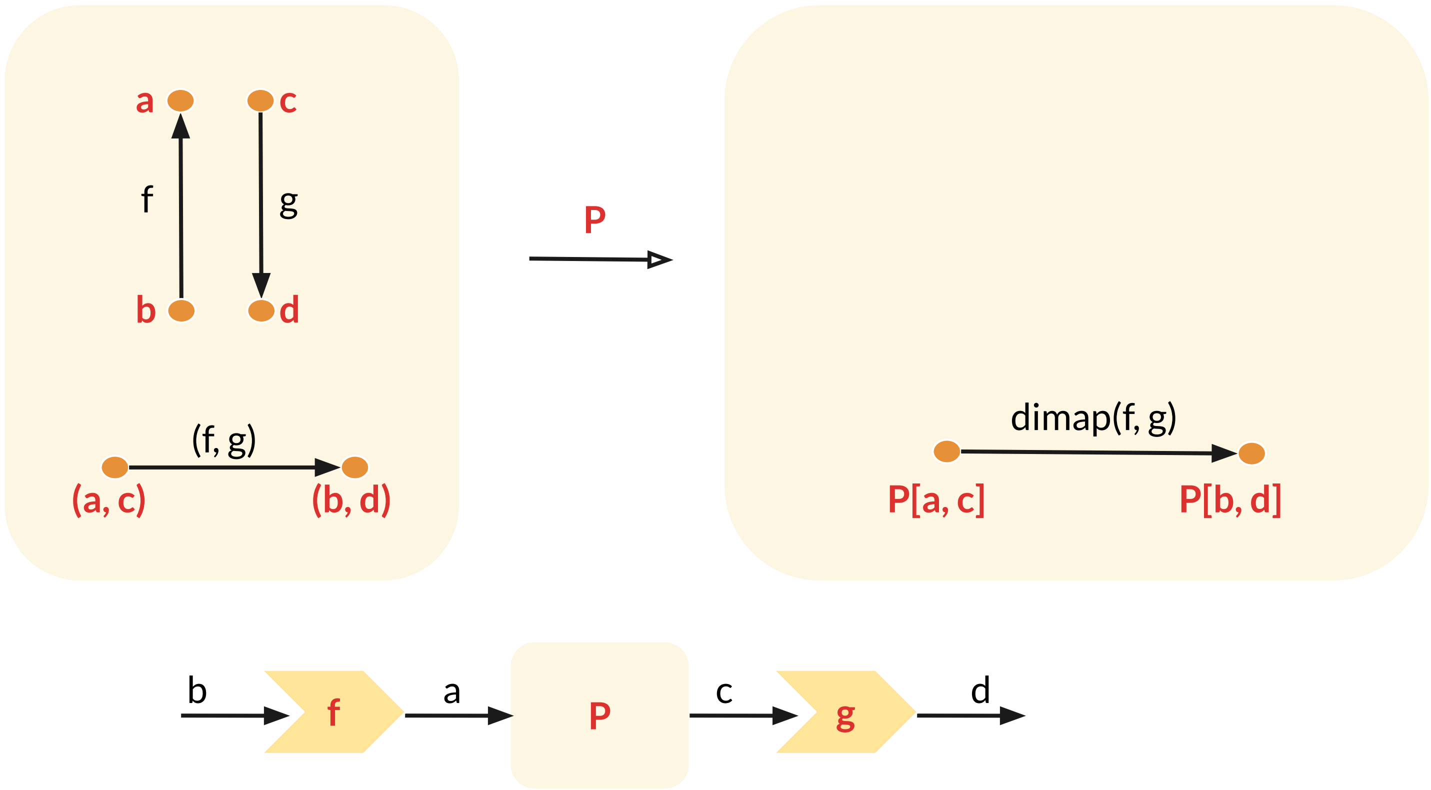
bifunctor



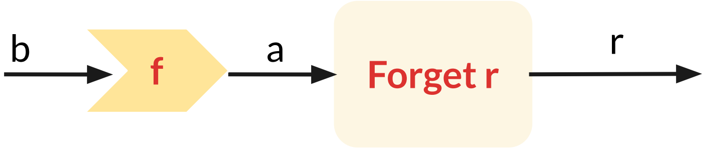
Example of bifunctor

```
from funclift.types.either import Right, Either, Left

def add1(n: int) -> int:
    return n + 1

def negate(b: bool) -> bool:
    return not b

v1: Either[bool, int] = Right(5)
v2 = v1.bimap(negate, add1)
assert v2 == Right(6)
```

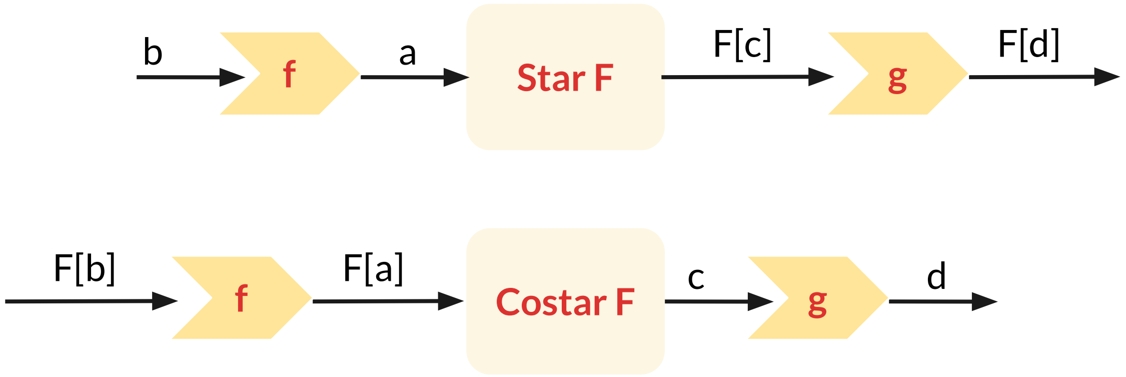
profunctor



Forget

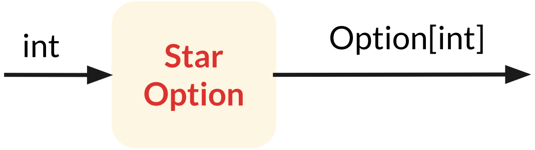


Star and Costar



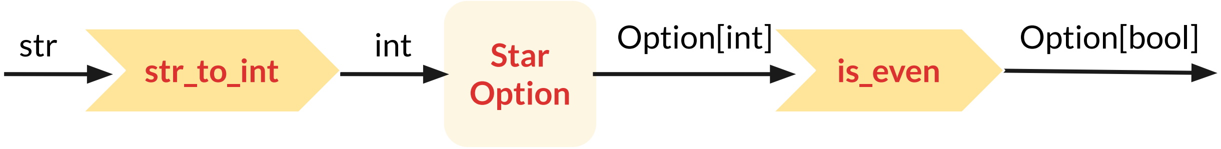
Example of Star profunctor

```
from funclift.types.option import Option, Some, Nothing
from funclift.types.star import Star

def ten_mod_by(n: int) -> Option[int]:
  return Nothing() if n == 0 else Some(10 % n)


star1 = Star(ten_mod_by)
assert star1.run(3) == Some(1)
assert star1.run(0) == Nothing()
```



Example of Star profunctor

```
def is_even(n: int) -> bool:
  return n % 2 == 0

def str_to_int(text: str) -> int:
  return int(text)

star2 = star1.dimap(str_to_int, is_even)
assert star2.run('6') == Some(True)
assert star2.run('0') == Nothing()
```



Summary

- Categories
- Covariant functors
- Composing functors
- Contravariant functors
- Closed functor
- Applicative functors
- Bifunctors
- Profunctors