Математика функционального программирования

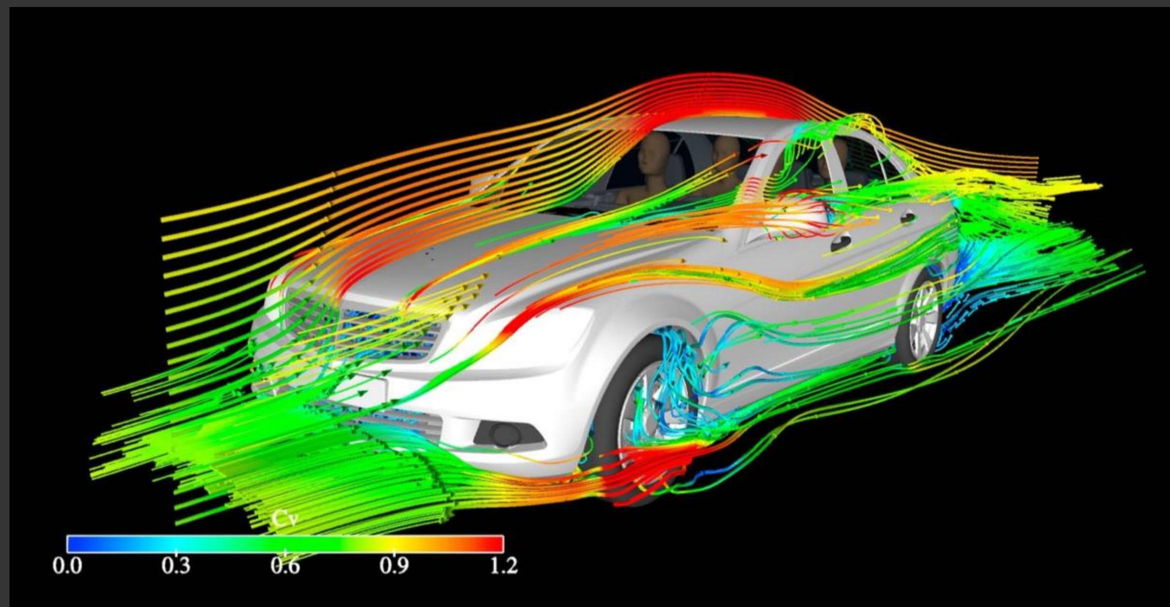
Дмитрий Сошников,

К.ф.-м.н., доцент НИУ ВШЭ/МАИ
Тех.рук. Лаборатории генеративного ИИ Школы дизайна НИУ ВШЭ, ex-Microsoft

@shwars – http://soshnikov.com

http://t.me/shwarsico

Программирование и математика

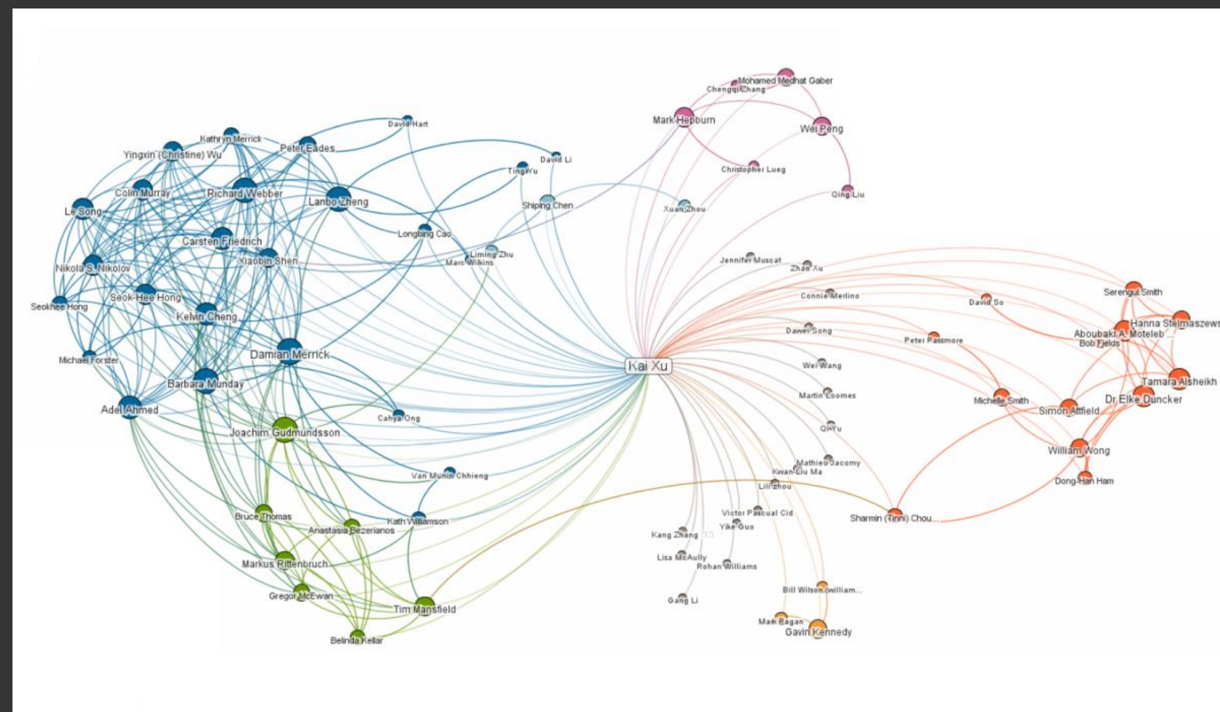


Графы

Машинное обучение / нейросети

Дифференциальные уравнения

Оптимизация



Математика в основе программирования

Машина Тьюринга

(λf.(λx.f(xx)) (λx.f(xx)))
(λf.λx.(λp.λt.λe.pte) ((λn.n(λc.λa.λb.b)
(λa.λb.a)) (x)) (λs.λz.sz)
((λn.λm.λs.n(ms)) (x)
(f((λn.λs.λz.(λp.p(λa.λb.b))
(n(λp.(λa.λb.λt.tab)
(s((λp.p(λa.λb.a)) (p))) ((λp.p(λa.λb.a))
(p))) ((λa.λb.λt.tab) (z) z)))(x)))))

Лямбда-исчисление

Математика в основе программирования

1936 г.

1930-е гг.

Основы лямбда-исчисления

$\lambda x \,.\, E(x)$ Аппликация

$(f\ x)$ Абстракция

$(\lambda x \,.\, (x+1))\ 5 \longrightarrow (5+1) \longrightarrow 6$

β-редукция η-редукция

Подробнее: чистое vs. прикладное λ–исчисление, нумералы

Пример на F#

$(\lambda x \,.\, (x+1))\ 5$

```
(fun x->x+1) 5

let incr = fun x->x+1 in incr 5

let x = E in F   ≡  (fun x -> F) E
```

Пример на F#

```
let f x = x+1 in
let t = 5 in
f t
```

```
(fun f t -> f t) (fun x->x+1) 5
```

Каррирование и циклы

$$e^x = 1 + x + \frac{x^2}{2} + \cdots + \frac{x^n}{n!} + \cdots$$

Рекурсия

```
let rec fact n =
    if n = 1 then 1
    else n*fact(n-1)
```

```
let rec fact = fun n -> if n = 1 then 1 else n*fact(n-1)

fact = (fun f -> fun n -> if n = 1 then 1 else n*f(n-1)) fact

fact = F fact

fact = fix F                let rec fix f x = f (fix f) x

fact = fix (
     fun f -> fun n -> if n = 1 then 1 else n*f(n-1))
```

Снова факториал

```
fact = fix (
     fun f -> fun n -> if n = 1 then 1 else n*f(n-1))
```

```
let s f g x = f x (g x)
let k x y = x
let c f a b = f b a
```

```
let cond p f g x =
   if p x then f x else g x
```

```
let fact = fix (fun f-> cond ((=)1) (k 1) (fun n->n*f(n-1)))

let fact = fix (cond ((=)1) (k 1) >> (fun f n->n*f(n-1)))
```

Подробнее: [комбинатор неподвижной точки](), [комбинаторная логика]()

Вывод №1

Любая программа может быть написано как одно большое выражение, содержащее аппликацию и абстракцию, либо даже только аппликацию. Выполнение такой программы – это последовательная редукция такого выражения до финального результата.

Рассуждать над большой программой можно как над функцией с входами и выходами, которая разбивается на композицию более простых функций.

Алгебра типов

```
type book = string * string * int // автор, название, год

type Source =
| Book of book
| Url of string

type download_func = Source -> Option<string>
```

Алгебра типов

```
type Option<T> =
| None
| Some of T
```

```
let x = Some(5)
let y = None
```

$o = 1 + t$

```
type book = string*Option<int>
```

$b = s*(1 + i)$

```
type book =
| Title of string
| TitleYear of string*int
```

$b = s + s*i$

Алгебраические типы данных

```
type List<T> =
| []
| Cons of T*List<T>
```

L = 1 + t*L

L = 1 + t*(1+t*(1+t*(....)))) = 1 + t + t*t + t*t*t + ...

Подробнее: алгебраические типы данных, изоморфизм Карри-Ховарда, coq

Линзы, оптика

```
let doc = Body([
    Header(1,String "Title");
    Paragraph(String "This is an introduction");
    Table([
      TableRow([String "Item 1";String "$1"]);
      TableRow([String "Item 2";String "$2"])
      ])
 ])
```

```
type Lens<'t,'a> = ('t -> ‘a) * ('t->'a->'t)
```

Вывод №2

Алгебраические типы данных позволяют легко конструировать предметно-ориентированные абстракции

Богатые возможности оперирования функциями приводят к созданию интересных абстракций и паттернов программирования

Теория категорий

id

f

g

A

B

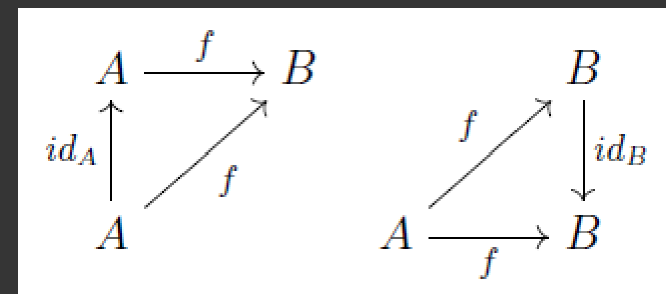
C

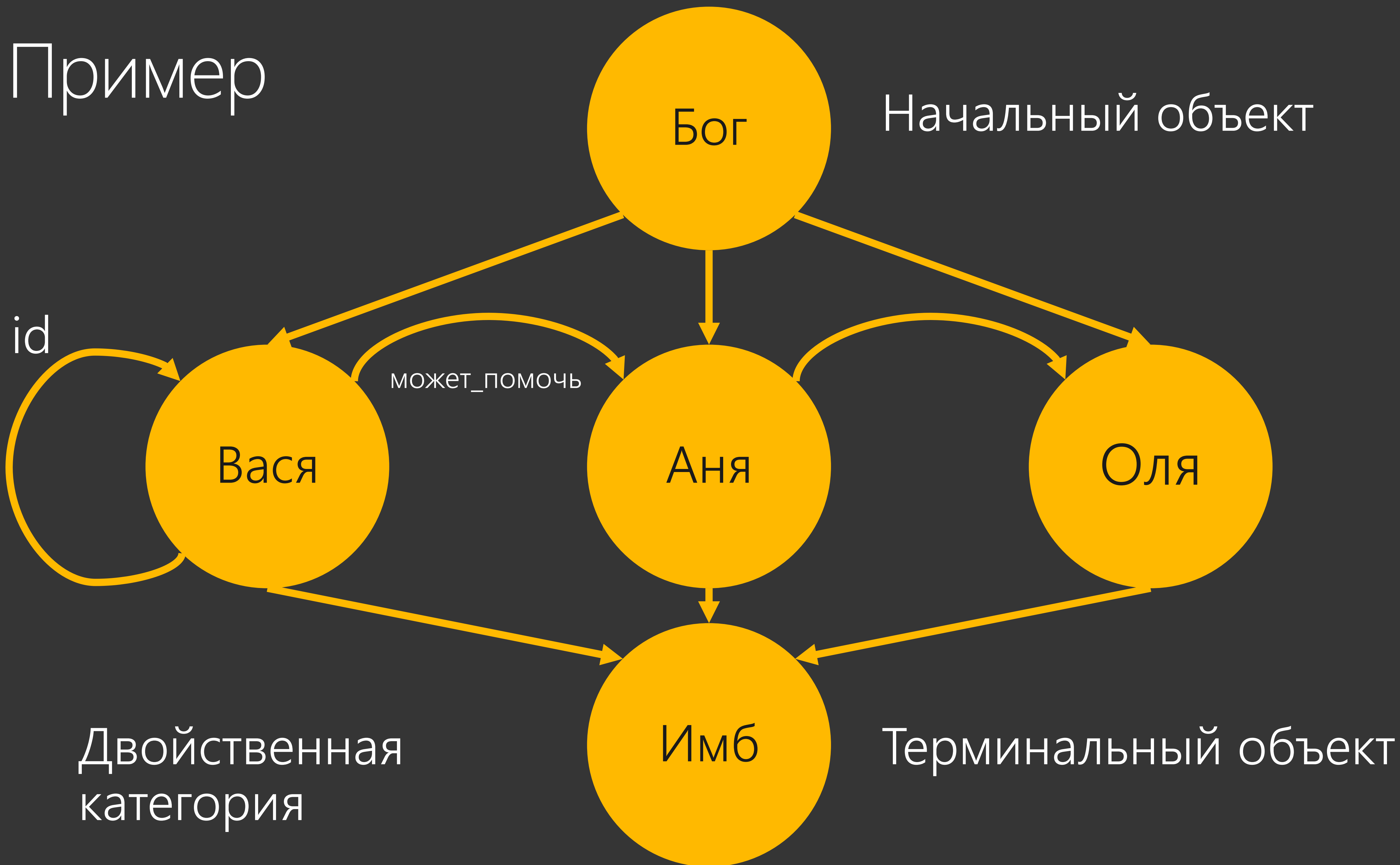
$\forall\ A \in Ob\ \exists\ id_A : A \to A$

$\forall\ f, g \in Mor\ \exists\ f \circ g \in Mor$

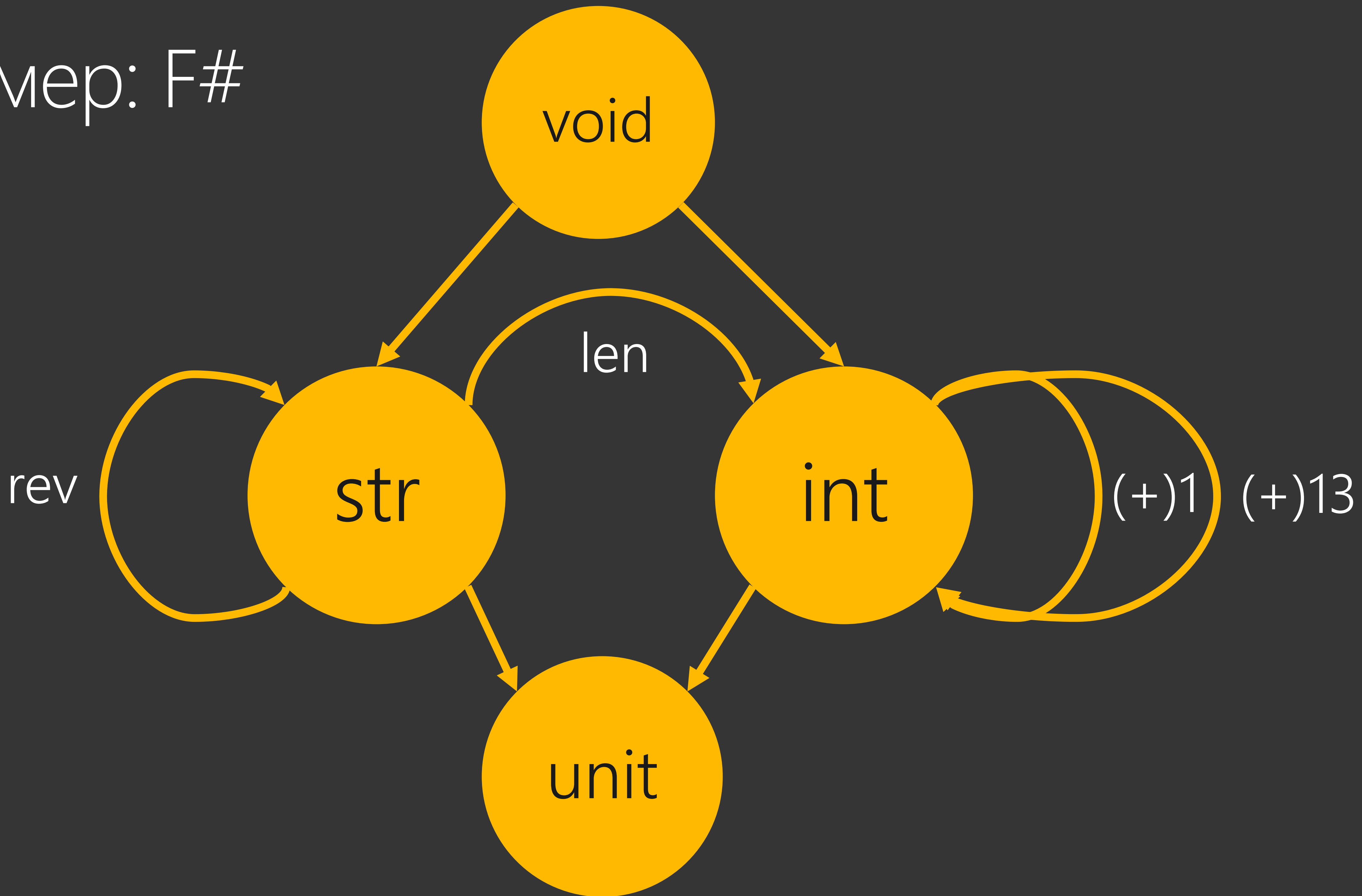
$\forall\ f, g, h \in Mor\ (f \circ g) \circ h = f \circ (g \circ h)$

$\forall\ f : A \to B \quad f \circ id_A = id_B \circ f = f$

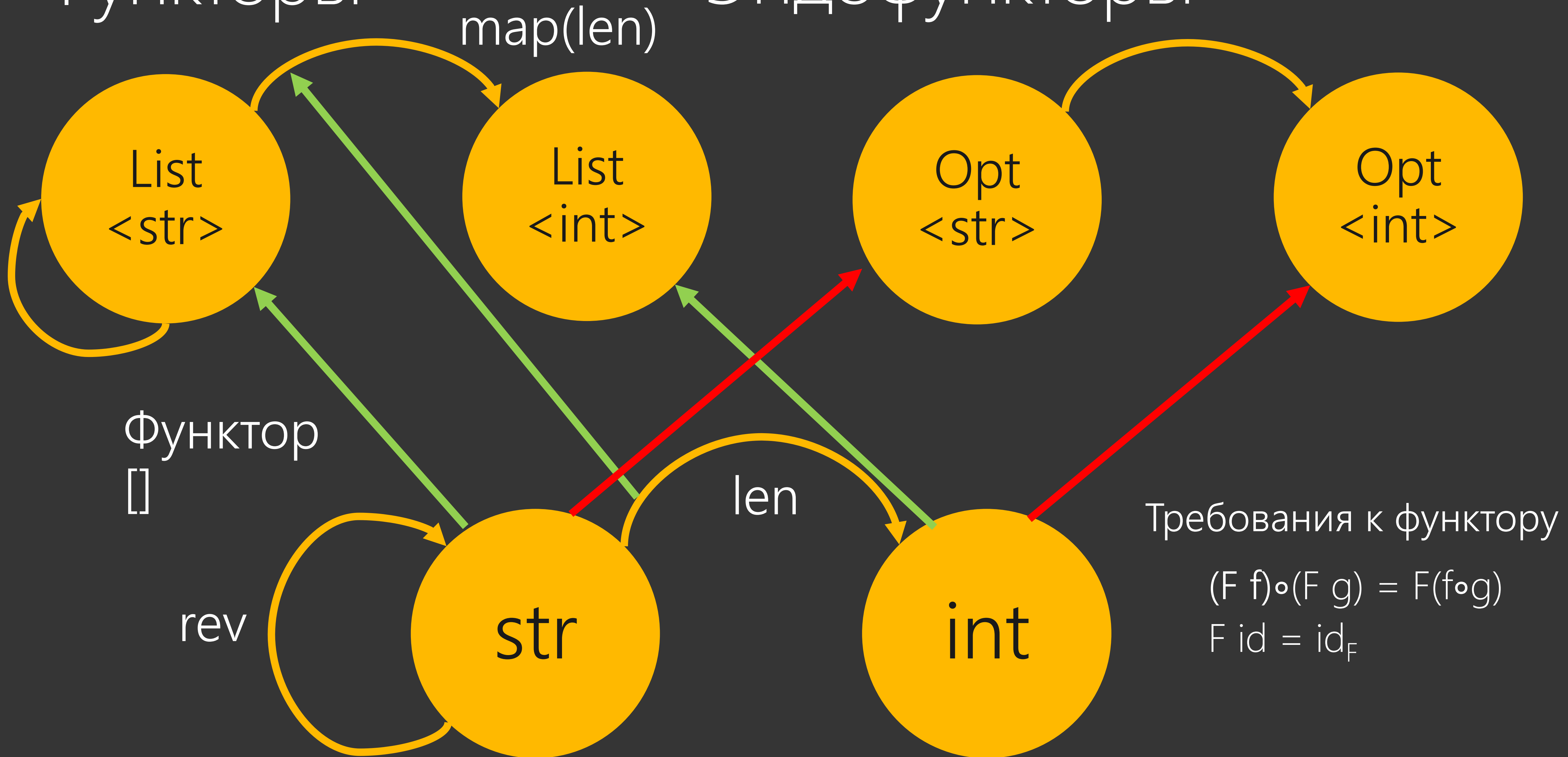



Пример
Бог
Начальный объект
id
может_помочь
Вася
Аня
Оля
Имб
Двойственная
категория
Терминальный объект

Пример: F#

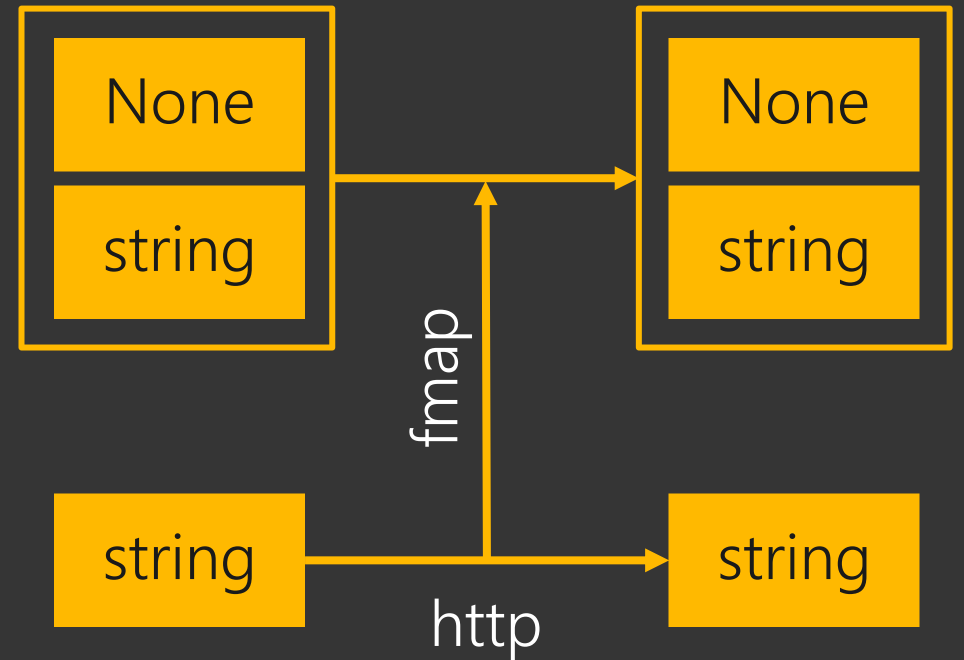


Функторы Эндофункторы



Функторы

```
let fmap f x = match x with
| None -> None
| Some(x) -> Some(f x)
```




```
[Book("Сошников","ФункПро",2011), Url("http://yandex.ru")]
|> List.map(JustUrl)
|> List.map(fmap http)
|> List.reduce(concat)
|> GptSummarize
```

```
|> List.map(JustUrl >> fmap http)
```

Монады

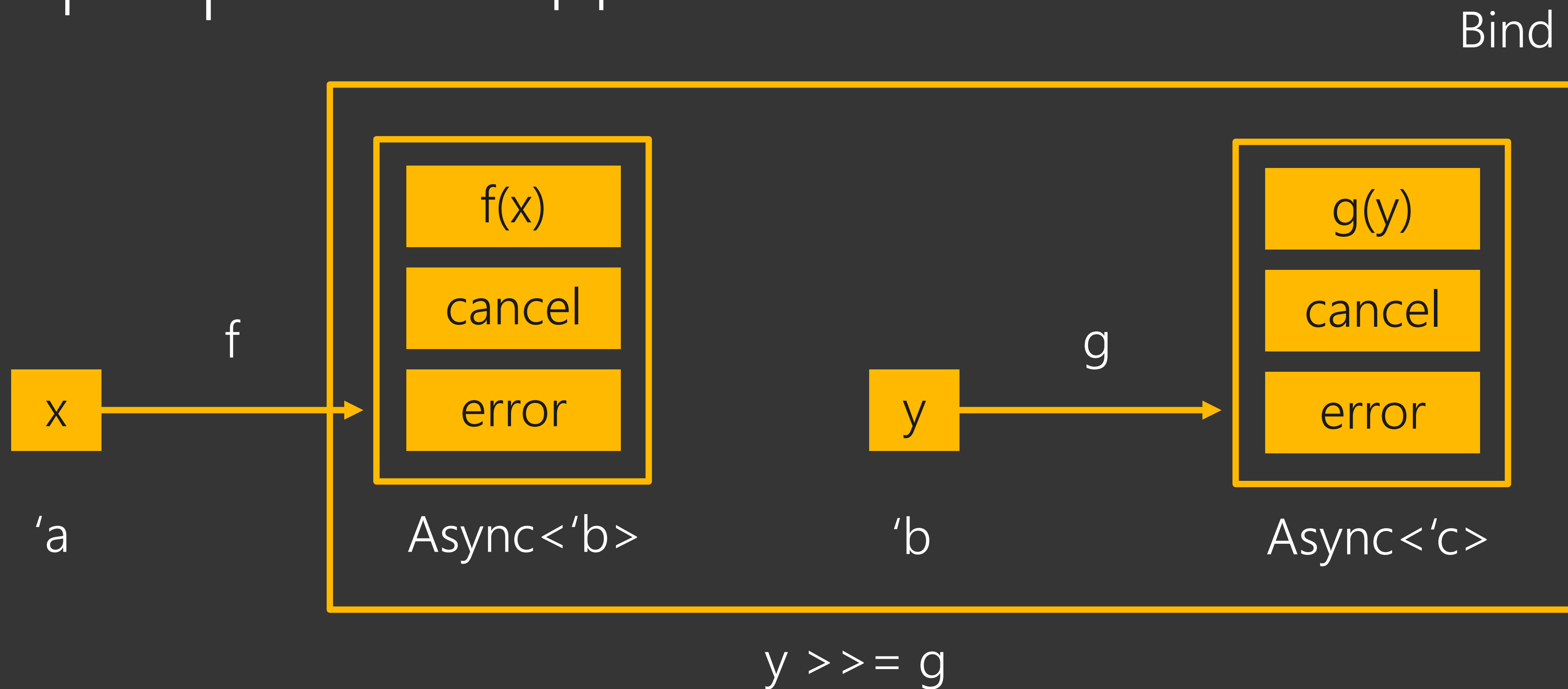
```
http : string -> Option<string>
```

```
[Book("Сошников","ФункПро",2011),
 Url("http://yandex.ru"),
 Url("http://localhost"]
|> List.map(unpack >>= http >>= grep "ФункПро")
|> List.reduce(concat)
|> GptSummarize
```

А если без вот этого всего?

```
string SummarizeDocs(List<Book> L)
{
    var res = new StringBuilder("Пожалуйста, суммаризуй документ:");
    foreach(var d in L)
    {
        switch(d.kind)
        {
            case "Book“: continue;
            case "Url" :
                try
                {
                    var c = http(d.url);
                    if c.contains("ФункПро") { res.append(c); }
                }
                catch {}
        }
        return GptSummarize(res);
    }
}
```

Ещё про монады



return : 'a -> M<'a>

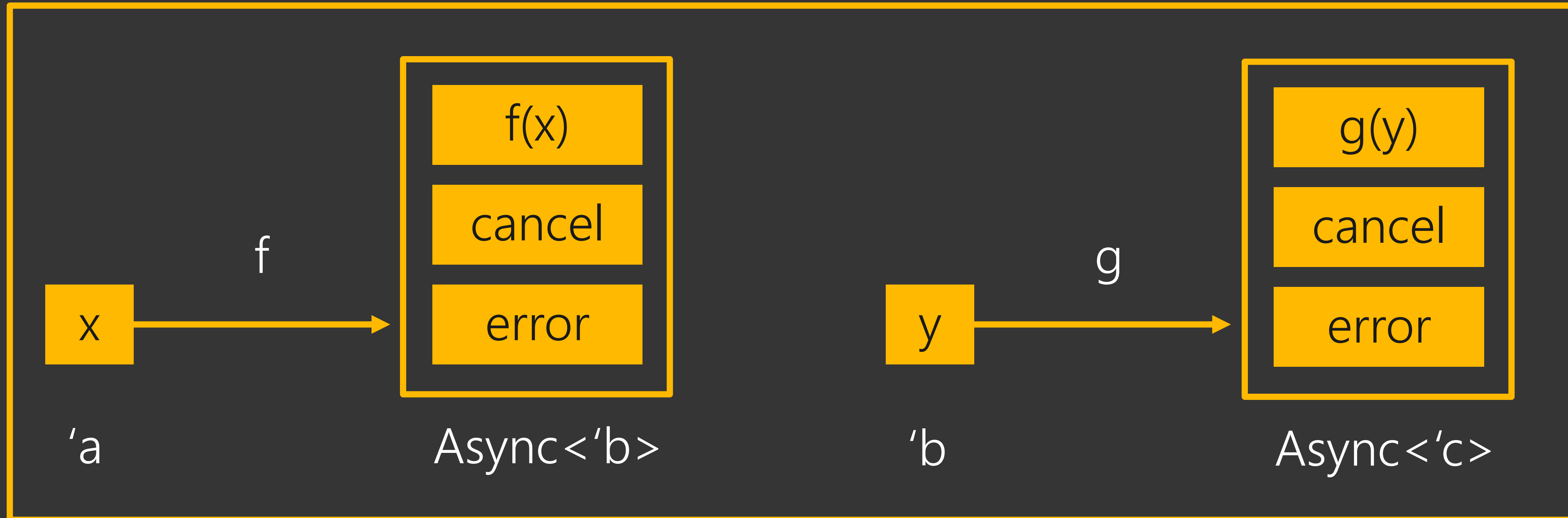
bind : M<'a> -> ('a -> M<'b>) -> M<'b>

Пример на F#

```
let get_images L =
   L
   |> map (fun x ->
              async {
                let! img = http_async(x)
                let res = resize(img)
                do! save(img)
              })
   |> Async.runSynchronously()
```

Монады и категории Клейсли

Стрелка Клейсли



f >=> g

return : ‘a -> M<‘a>

>=> : (‘a -> M<‘b>) -> (‘b -> M<‘c>) -> (‘a -> M<‘c>)

Моноид в категории эндофункторов

```
return >=> f   ≡  f
f >=> return   ≡  f
(f >=> g) >=> h   ≡   f >=> (g >=> h)
```

Моноид

$0+x = x$

$x+0 = x$

$(a+b)+c = a+(b+c)$

0 – единичный элемент

\+ - ассоциативная операция

Подробнее: Бартош Милевский, теория категория для программистов, англ

Вывод №3

Внезапно новое направление математики позволило рассуждать над оригинальными функциональными дизайн-паттернами.

Они позволяют делать код более модульным не за счет ОО-декомпозиции, а за счет оперирования функциями и рассуждений на более высоком уровне абстракции.

Мораль

- Математика развивает мозг
- Математика структурирует мышлен
  привести к более «чистым» и боле
  дизайн-паттернам
- Использовать эти паттерны можно
  императивных и ОО языках
- Но лучше всего – изучите Haskell в

Дмитрий Сошников

http://soshnikov.com
dmitri@soshnikov.com
http://t.me/shwarsico