Монады в C#. Элементы теории категорий

Автор: Кирилл Сухоруких

Об авторе

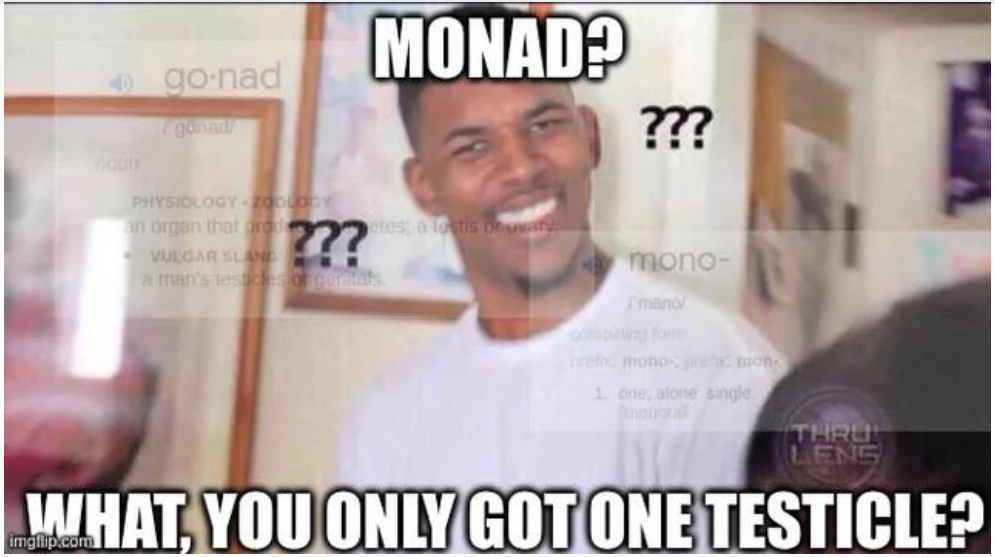
Кирилл Сухоруких

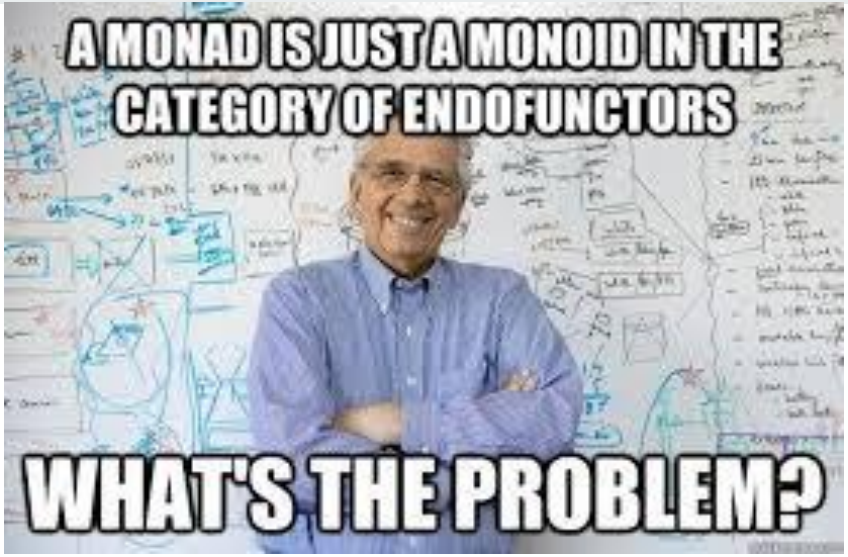
Компания: Softlight.
https://career.habr.com/companies/softlight

Опыт работы: 9 лет

Должность: Senior fullstack-developer React/.NET

Контакты: www.linkedin.com/in/kirill-sukhorukikh-226b3a191





A MONAD IS JUST
A MONOID IN THE
CATEGORY OF
ENDOFUNCTORS

Проблематика

Композиция операций с «эффектами» в императивном ООП-коде приводит к:

- взрыву сложности проверок (null, исключения)
- «пирамидам ада» из async/await + try/catch
- дублированию логики обработки ошибок и побочных эффектов

Решение из **теории категорий:**

Монада — абстракция последовательной композиции вычислений с контекстом

Было

```
public static string? GetUserEmail_Classic(int userId)
{
    var user = Database.GetUser(userId);        // может быть null
    if (user == null) return null;

    var profile = Database.GetProfile(user.Id); // может быть null
    if (profile == null) return null;

    return profile.Email;                       // и тут тоже может быть null!
}
```

Стало

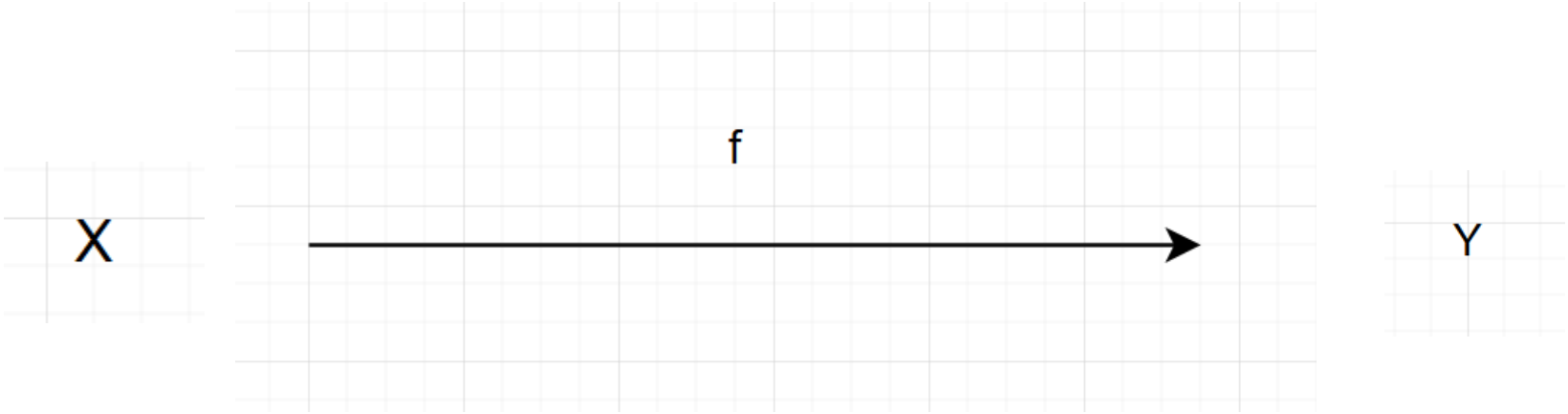
```
public static Option<string> GetUserEmail_Monadic(int userId)
{
    return Some(userId)                          // Option<int>
        .Bind(Database.GetUser)                  // Option<User>
        .Bind(user => Database.GetProfile(user.Id)) // Option<Profile>
        .Map(profile => profile.Email);          // Option<string>
}
```

Определение

Тео́рия катего́рий — раздел математики, изучающий свойства отношений между математическими объектами, не зависящие от внутренней структуры объектов.

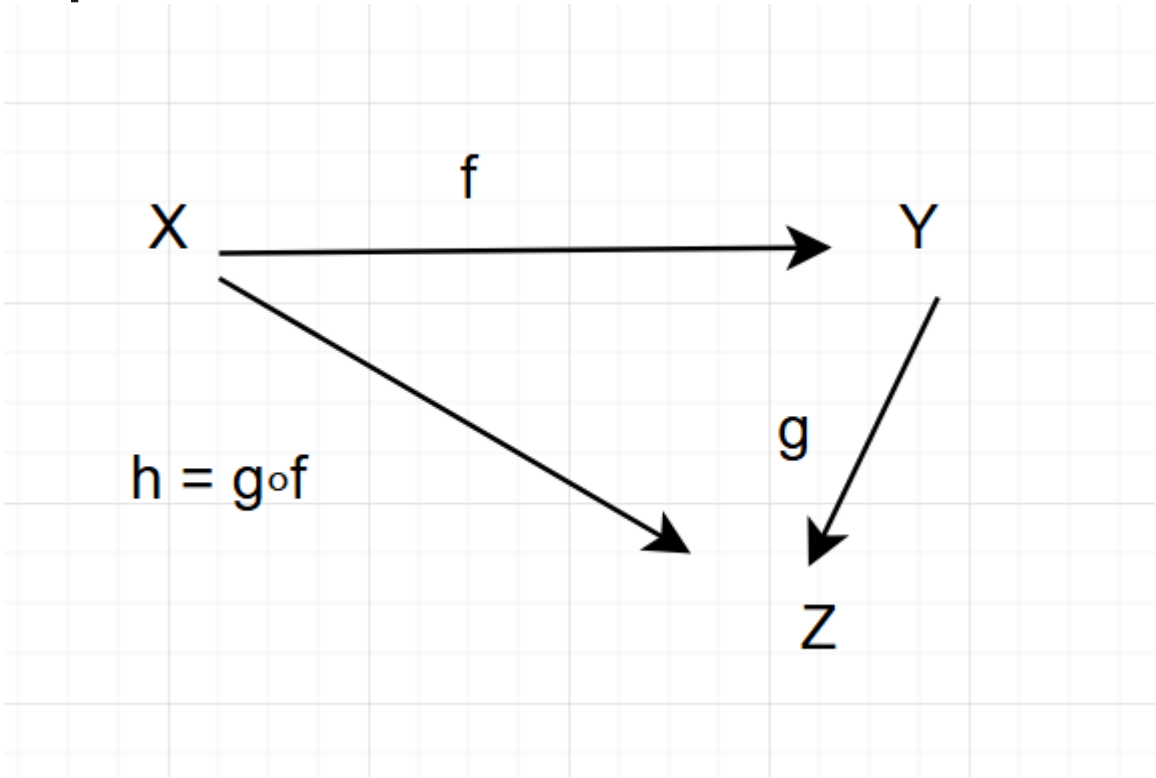
Категория (объекты и морфизмы)

Категория состоит из объектов и морфизмов между ними



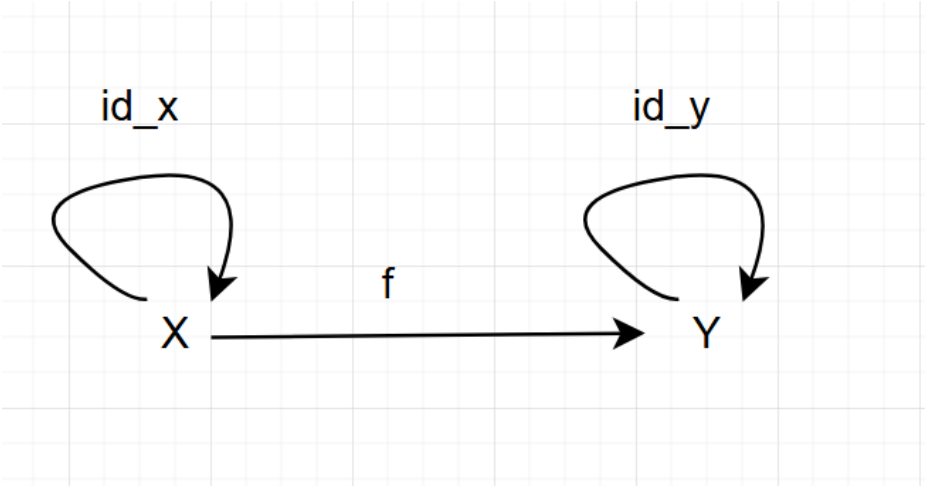
Категория (композиция)

1. Для **f: X → Y** и **g: Y → Z** существует **h: X →Z** композиция **h: g ∘ f**, что эквивалентно **g(f(X)): X →Z**



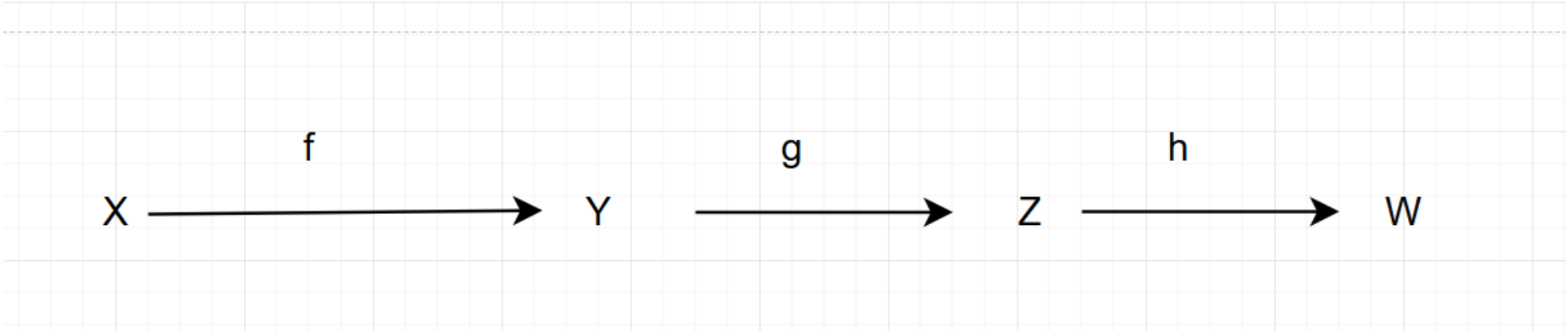
Категория (тождественный морфизм)

2. Для любого объекта x существует единичная стрелка (тождественный морфизм) **id_x: X→X** такая, что для любого **f: X→ Y** верно, что **f∘id_x = id_y∘f =f**



Категория (ассоциативность композиции)

3. Композиция ассоциативна f∘(g∘h) = (f∘g)∘h



Пример категории №1

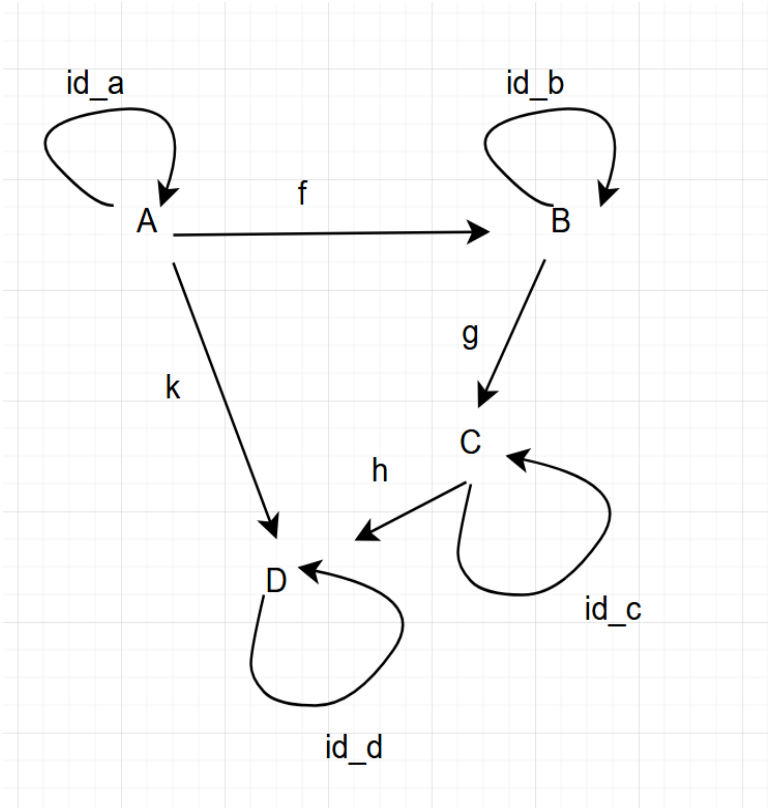
Множества **(объекты категории)**

A = {1,2}

B = {x,y,z}

C = {a,b}

D = {true,false}

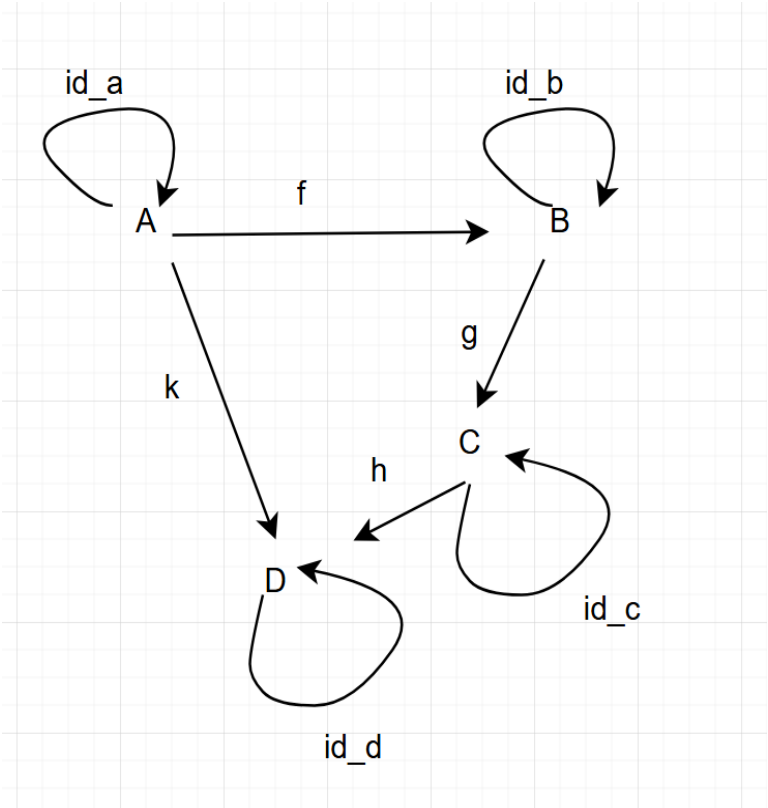


Пример категории №1

Стрелки

f: A→B | f(1)=x, f(2)=y
g: B→C | g(x)=a, g(y)=a, g(z)=b
h: C→D | h(a)=true, h(b)=false
k: A→D | k(1)=true, k(2)=false

Тождественные
id_a : A→A | id_a(1)=1, id_a(2)=2
id_b : B→B | id_b(x)=x, id_b(y)=y, id_b(z)=z
id_c : C→C | id_c(a)=a, id_c(b)=c
id_d : D→D| id_d(true)=true,
id_d(false)=false



Проверка аксиом

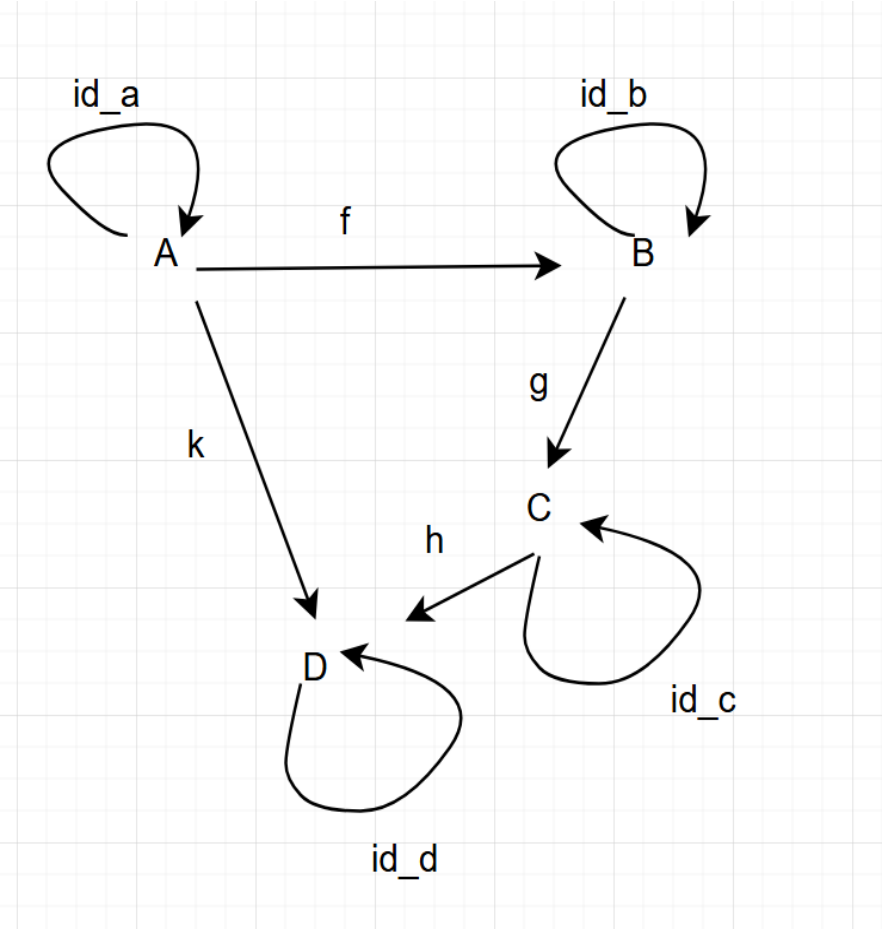
Композиция существует

g∘f: A→C где g∘f(1) = g(f(1)) = g(x) = a;
g∘f(2)=g(f(2))=g(y)=b

h∘g: B→D где h∘g(x)=h(g(x))=true

…

и так далее



Проверка аксиом

Единичные стрелки

f∘id_a : f∘id_a(1) = f(id_a(1))=f(1)=x (левая единица)

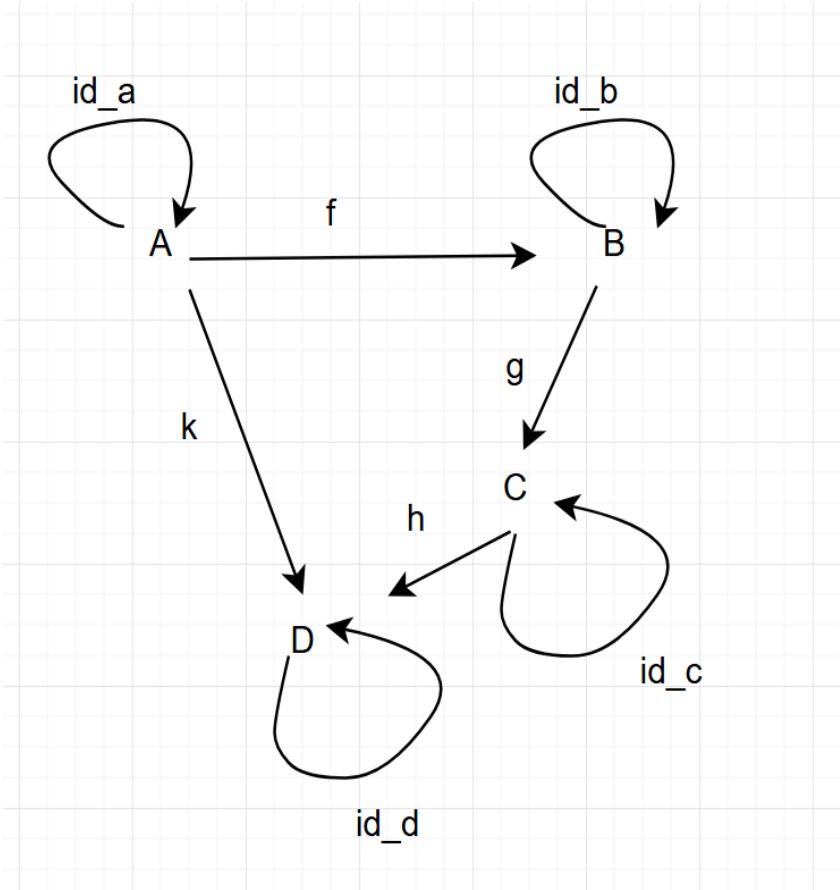
id_b∘f : id_b∘f(1)= id_b(f(1))=id_b(x)=x (правая единица)

g∘id_b : g∘id_b(x) = g(id_b(x))=g(x)=a (левая единица)

id_c∘g: id_c∘g(x) = id_c(g(x))=id_c(a)=a (правая единица)

...

и так далее



Проверка аксиом

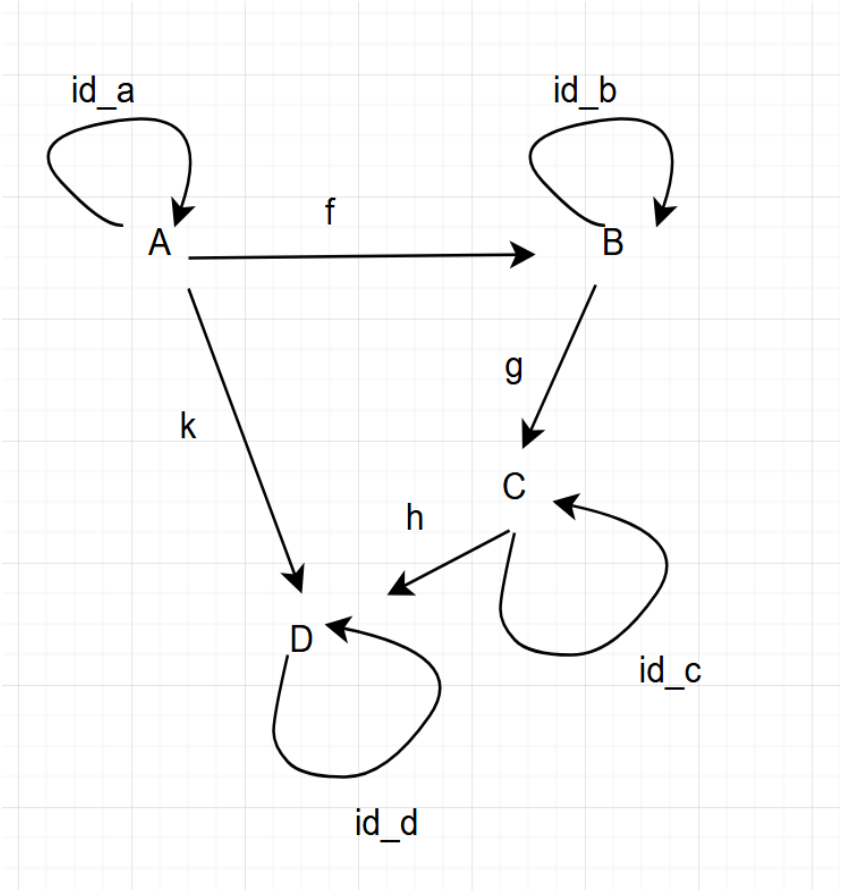
Ассоциативность

$(h \circ g) \circ f = h \circ (g \circ f)$

$((h \circ g) \circ f)(1) = (h \circ g)(f(1)) = (h \circ g)(x) =$

$= h(g(x)) = h(a) = true$

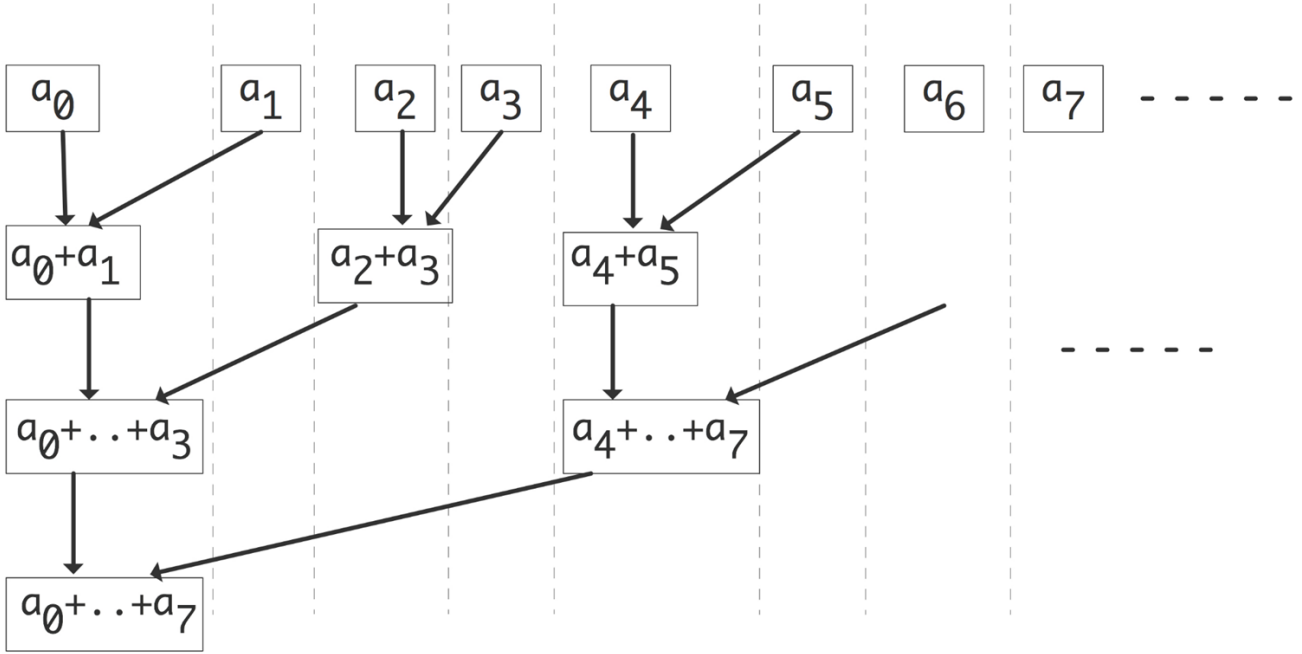
$h \circ (g \circ f)(1) = h \circ (g(x)) = h(a) = true$



Ассоциативность

Ассоциативность упрощает вычисления

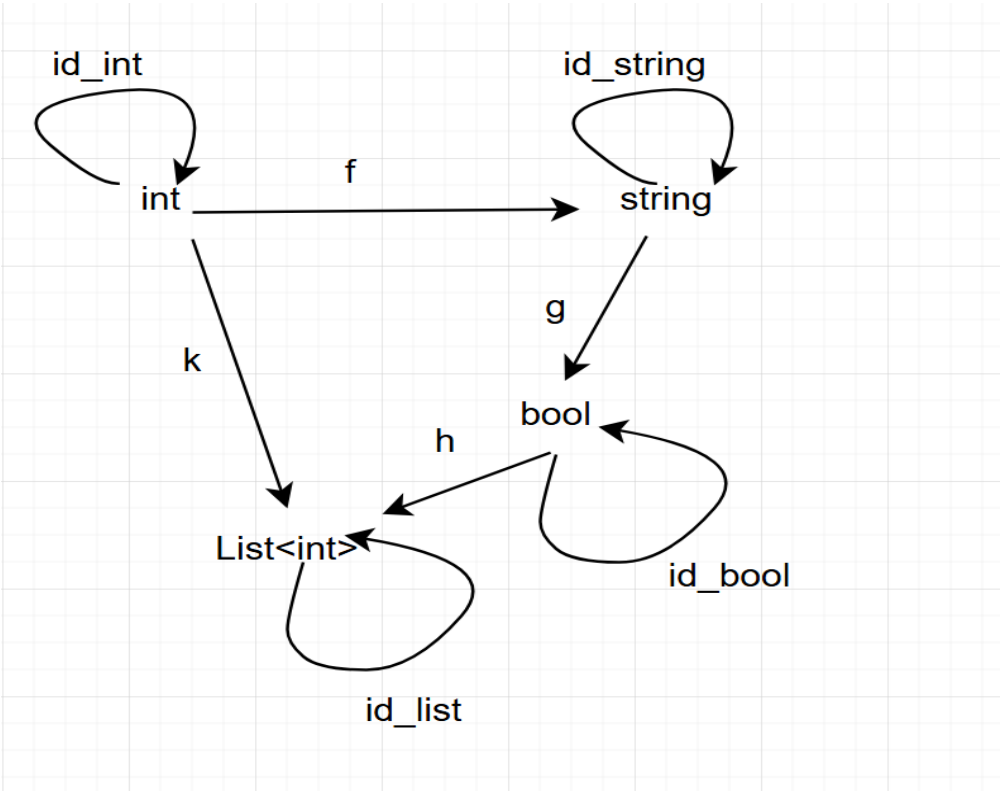
a+b+c+d = (a+b) + (c+d)



Пример категории №2

Объекты: типы данных языка C# (int, string, bool, List<int>)

Стрелки: функции



Пример категори №2

```
using System;
using System.Collections.Generic;

// Обычные функции
Func<int,    string>       f = x => x.ToString();
Func<string, bool>         g = s => s.Length > 0;
Func<bool,   List<int>>    h = b => b ? new List<int> { 1, 2, 3 } : new List<int>();

// Композиция "вручную": h ∘ g ∘ f
Func<int, List<int>> k = x => h(g(f(x)));

// === Идентичности (identity) для разных типов ===
Func<int,    int>          id_int    = x => x;
Func<string, string>       id_string = x => x;
Func<bool,   bool>         id_bool   = x => x;
Func<List<int>, List<int>> id_list   = x => x;
```

Проверка аксиом (композиция)

```
Func<int, string> f = x => x.ToString();
Func<string, bool> g = s => s.Length > 0;
Func<bool, List<int>> h = b => b ? new() { 1, 2, 3 } : new();

var compose_g_f = f.Then(g);               // int → bool
var compose_h_g = g.Then(h);               // string → List<int>

Console.WriteLine(compose_g_f(100));       // True
Console.WriteLine(compose_h_g("hello").Count); // 3
Console.WriteLine(compose_h_g("").Count);  // 0

public static class FuncExtensions
{
    public static Func<T, TResult> Then<T, TMiddle, TResult>(
        this Func<T, TMiddle> first,
        Func<TMiddle, TResult> second)
        => x => second(first(x));
}
```

Проверка аксиом

Единичные стрелки

```
Func<T, T> Id<T>() => x => x;

var f_left  = Id<int>().Then(f)(1);       // левая единица: id ∘ f
var f_right = f.Then(Id<string>())(1);    // правая единица: f ∘ id
Console.WriteLine(f_left == f_right);     // True

var g_left  = Id<string>().Then(g)("hello");     // id ∘ g
var g_right = g.Then(Id<bool>())("hello");       // g ∘ id
Console.WriteLine(g_left == g_right);            // True

var h_left  = Id<bool>().Then(h)(true);          // id ∘ h
var h_right = h.Then(Id<List<int>>)(true);       // h ∘ id
Console.WriteLine(h_left.SequenceEqual(h_right)); // True

var k_left  = Id<int>().Then(k)(1);              // id ∘ k
var k_right = k.Then(Id<List<int>>)(1);          // k ∘ id
Console.WriteLine(k_left.SequenceEqual(k_right)); // True
```

Проверка аксиом (ассоциативность)

```
Func<int,    string> f = x => x.ToString();
Func<string, bool>   g = s => s.Length > 0;
Func<bool,   List<int>> h = b => b ? new() { 1, 2, 3 } : new();

var leftComposition  = h.Then(g.Then(f));     // (h ∘ g) ∘ f
var rightComposition = h.Then(g).Then(f);     // h ∘ (g ∘ f)

var testValues = new[] { 0, 1, 5, 100, -10 };

Console.WriteLine("=== Тест ассоциативности композиции ===");
Console.WriteLine("(h ∘ g) ∘ f  =  h ∘ (g ∘ f)\n");

foreach (var x in testValues)
{
    var leftResult  = leftComposition(x);
    var rightResult = rightComposition(x);

    Console.WriteLine($"x = {x,3}");
    Console.WriteLine($"  (h ∘ g) ∘ f(x) = [{string.Join(", ", leftResult)}]");
    Console.WriteLine($"  h ∘ (g ∘ f)(x) = [{string.Join(", ", rightResult)}]");
    Console.WriteLine($"  Равны: {leftResult.SequenceEqual(rightResult)}\n");
}
```

Проверка аксиом (ассоциативность)

```
=== Тест ассоциативности композиции ===
(h ∘ g) ∘ f  =  h ∘ (g ∘ f)

x =  0
  (h ∘ g) ∘ f(x) = [1, 2, 3]
  h ∘ (g ∘ f)(x) = [1, 2, 3]
  Равны: True

x =  1
  (h ∘ g) ∘ f(x) = [1, 2, 3]
  h ∘ (g ∘ f)(x) = [1, 2, 3]
  Равны: True
...
```

Моноид

Моноид это алгебраическая структура состоящая из:

Множества: **M**

Бинарной операции: * **: M*M→M** (операция замкнута на **M**)

Аксиомы:

1) Ассоциативность **(a*b)*c=a*(b*c)** для всех **a,b,c** из **M**
2) Существует единичный элемент **e*a=a*e=a** для всех **a** из **M**

Моноид в категории

M - объект категории (множество)

Морфизмы

*** : M*M→M**

e: 1 -> M

Пример

Категория: Type
Объекты: типы языка C#
Морфизмы: функции между типами

Моноид StringMonoid

```
1 usage
public class Unit
{
}

public class StringMonoid
{
    public static Type StringType => typeof(string);

    //Бинарная операция
    public static Func<(string, string), string> Concatenation = pair => pair.Item1 + pair.Item2;

    //Морфизм единичного элемента
    public static Func<Unit, string> EmptyString = _ => string.Empty;

}
```

Моноид StringMonoid

StringMonoid удовлетворяет всем аксиомам моноида:

✓Замкнутость — string × string → string

✓Ассоциативность — (a + b) + c = a + (b + c)

✓Единичный элемент — "" + a = a + "" = a

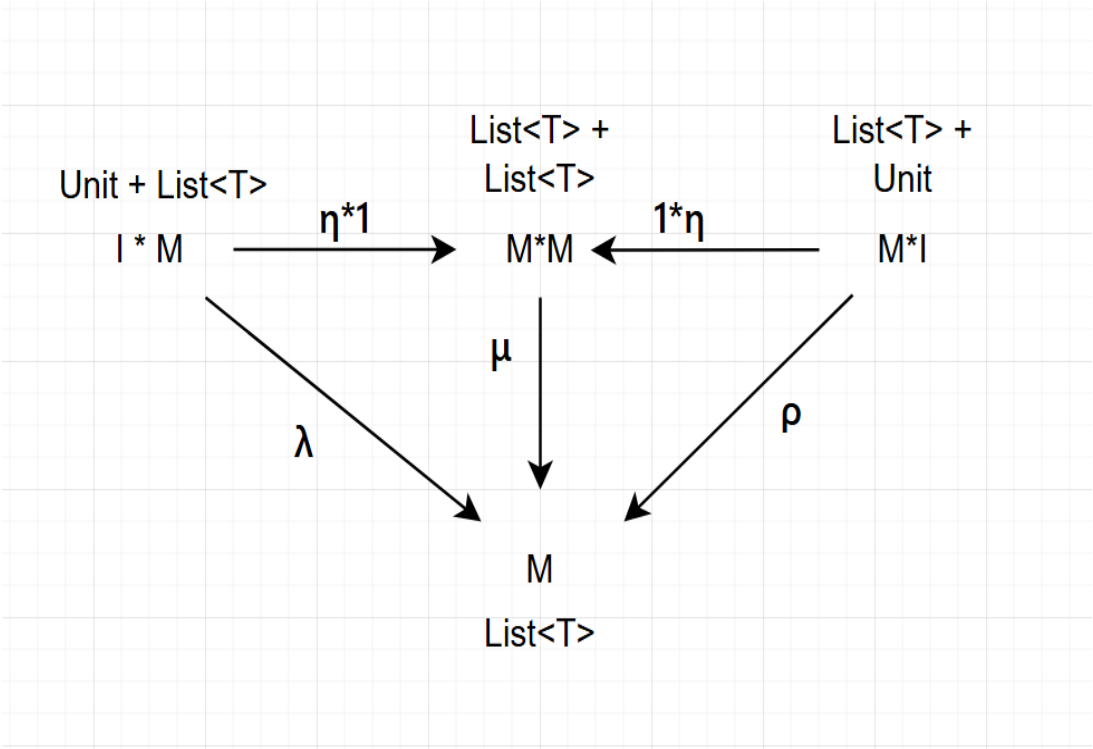
Моноид ListMonoid

```
19  public class ListMonoid<T>
20  {
21
22      public static Func<(List<T>, List<T>), List<T>> Concatenation = pair =>
23      {
24          var result = new List<T>(pair.Item1);
25          result.AddRange(pair.Item2);
26          return result;
27      };
28
29      public static Func<Unit, List<T>> EmptyList = _ => new List<T>();
30  }
```

Моноид(теория категорий)

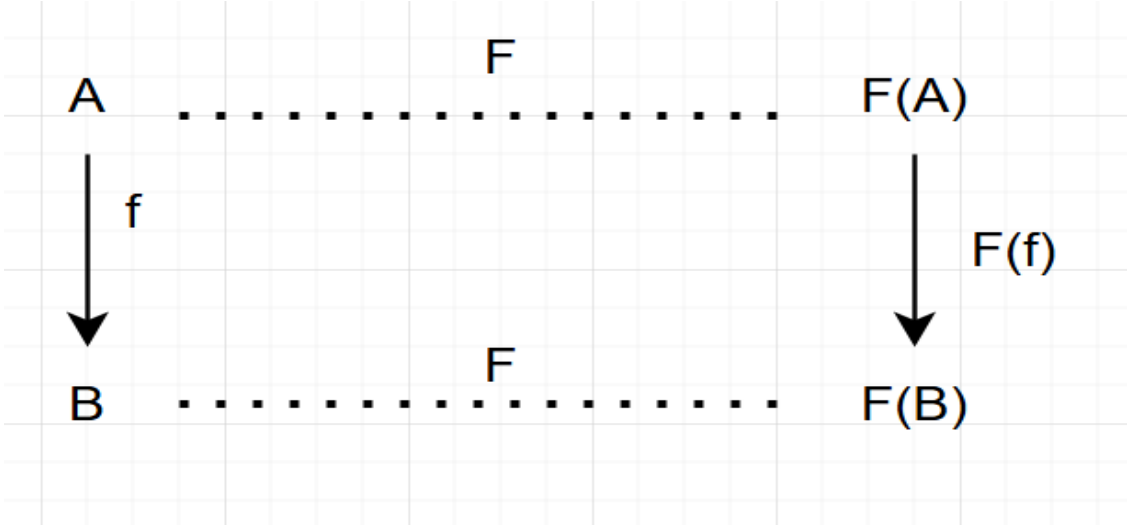
μ: M*M→M (умножение)
η: Unit→M (морфизм единичного элемента)
1: M→M (тождественный морфизм)
λ=μ(η*1)= Unit*M→M (левый унитор)
ρ=μ(1*η) = M*Unit→M (правый унитор)

В программировании Unit и тождественный морфизм будем опускать без потери строгости.



Функтор

Функтор - отображение между категориями, которое сохраняет структуру категории (объекты,морфизмы, композицию и тождественные морфизмы)



List<T> как функтор

List: Type → Type **(эндофунктор)**

int → List<int>

f: int →string => F: List<int> → List<string>

F(f) : List<A> → List<B>

List<T> как эндофунктор (проверка аксиом)

```
public static class Test
{
    public static List<B> F<A, B>(this List<A> a, Func<A, B> f)
    {
        var list_b = new List<B>();
        foreach (var item in a)
        {
            list_b.Add(f(item));
        }
        return list_b;
    }

    public static void Foo()
    {
        var list = new List<int>{1, 2, 3};
        var left = list.F(x => x);          //F(id_a)
        var right = list;                   //id(F(A))
        Console.WriteLine(left.SequenceEqual(right));
    }
}
```

List<T> как эндофунктор (композиция)

```
public static class Test
{
    // fmap — наш функтор для List<T>
    public static List<R> F<T, R>(this List<T> source, Func<T, R> f)
        => source.Select(f).ToList();

    public static void TestCompositionPreservation()
    {
        var list = new List<int> { 1, 2, 3 };

        Func<int, int>    f = x => x + 1;        // +1
        Func<int, string> g = x => x.ToString(); // в строку

        // Закон функтора: fmap(g ∘ f)  ≡  fmap(g) ∘ fmap(f)

        var left  = list.F(f.Then(g));           // fmap(g ∘ f)
        var right = list.F(f).F(g);              // fmap(g) ∘ fmap(f)

        Console.WriteLine("Сохранение композиции:");
        Console.WriteLine($"левая:  [{string.Join(", ", left)}]");   // ["2", "3", "4"]
        Console.WriteLine($"правая: [{string.Join(", ", right)}]");
        Console.WriteLine($"равны: {left.SequenceEqual(right)}");     // True
    }
}
```

List<T> как эндофунктор

List<T> с операцией F является эндофунктором, потому что:

Отображает объекты: List: Type → Type

Отображает морфизмы: f: A → B — F: List<A> → List<B>

Сохраняет тождества: List(id_A) = id_List<A>

Сохраняет композицию: List(g ∘ f) = List(g) ∘ List(f)

F - это именно та операция, которая делает List<T> функтором, отображающим функции между типами в функции между списками типов.

Категория эндофункторов

Объекты - это эндофункторы List<T>, Task<T>, IEnumerable<T>, Maybe<T> и т.д.

Морфизмы - **естественные преобразования** между эндофункторами

Композиция - композиция **естественных преобразований**

Тождества - тождественные преобразования

Пример (класс Maybe)

```
using System;

public readonly struct Maybe<T>
{
    public T Value { get; }
    public bool HasValue { get; }

    private Maybe(T value, bool hasValue)
    {
        Value = value;
        HasValue = hasValue;
    }

    public static Maybe<T> Some(T value)
    {
        if (value is null)
            throw new ArgumentNullException(nameof(value));
        return new Maybe<T>(value, true);
    }

    public static Maybe<T> None => new Maybe<T>(default!, false);

    public Maybe<U> Map<U>(Func<T, U> f)
    {
        if (!HasValue || f is null)
            return Maybe<U>.None;
        return Maybe<U>.Some(f(Value));
    }
}
```

Естественное преобразование

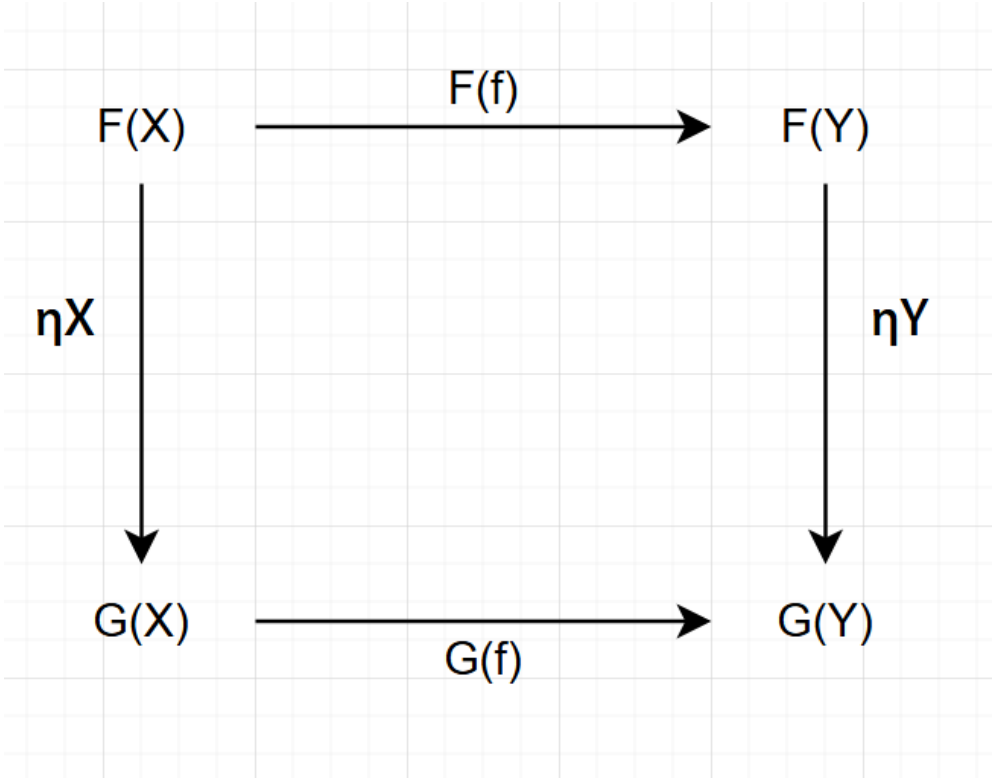
$\eta: F \to G$ - это семейство морфизмов $F(A) \to G(A)$ для любого объекта A, которое согласовано с функторами.

Естественное преобразование

f: X→Y (морфизм)
F,G - функторы (List<T>, Maybe<T> ...),
η: F→G (естественное преобразование)
List<T> -> Maybe<T>

Естественное преобразование не вмешивается в работу функторов, оно лишь переводит результат из одного функтора в другой.

Аксиома
G(f)∘ηX = ηY∘F(f)



Пример и проверка аксиомы

```
public void Test()
{
    //Путь 1
    //F(x)
    var maybeInt = Maybe<int>.Some(42);
    //ηx
    var listInt = MaybeToList(maybeInt);                //[42]
    var listString = listInt.Select(x => x.ToString());// ["42"]

    //Путь 2
    //F(x)
    var _maybeInt = Maybe<int>.Some(42);
    //F(f)
    var maybeString = _maybeInt.Map(x => x.ToString());// Some("42")
    //ηy
    var _listString = MaybeToList(maybeString);

    Console.WriteLine(listString.SequenceEqual(_listString));
}
```

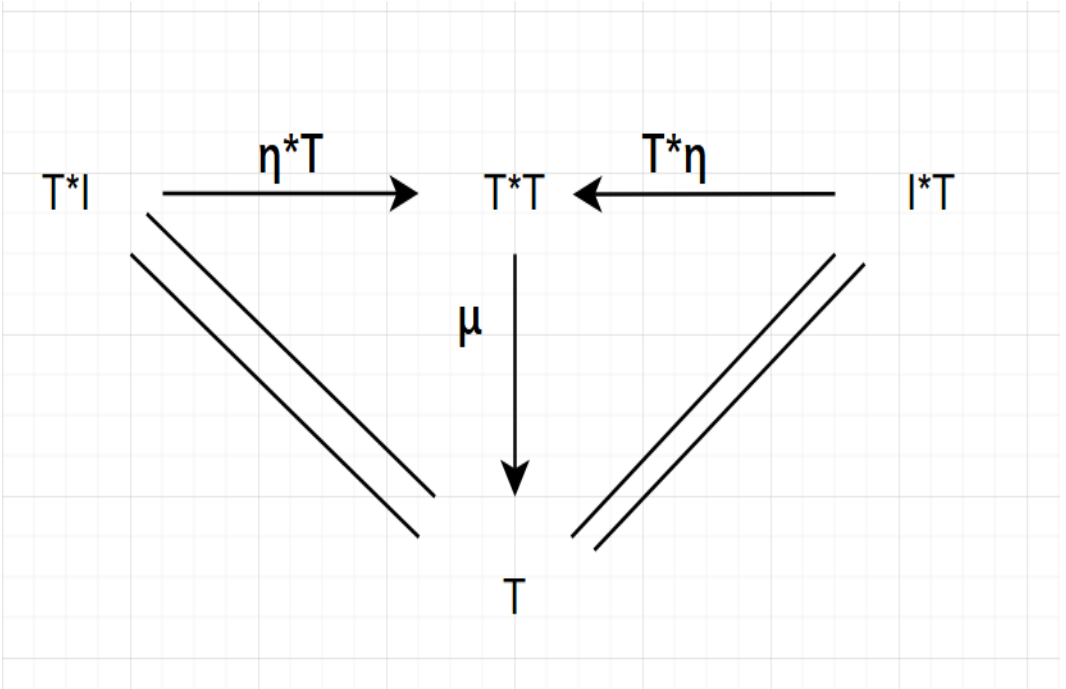
Вывод из аксиомы

Порядок применения не важен. Сначала сменить функтор, а потом применить морфизм f или сначала применить морфизм f, а затем сменить функтор.

Моноид в категории эндофункторов

η= A→Maybe(A) (упаковка)
μ: TxT →T (сплющивание или умножение монады)

Maybe<Maybe<...>> → Maybe



Пример

```
public static Maybe<T> Multiply<T>(Maybe<Maybe<T>> nested)
{
    return nested.HasValue ? nested.Value : Maybe<T>.None;
}
```

Неудобно...

Операция Bind

Введем новую операцию:

Дано:
ma: M(A)

f : A→M(B) - морфизм f

Применяем функтор к морфизму f

M(f) : M(A) → M(M(B))

Применяем M(f) к ma = M(M(B))

Умножаем μ(M(M(B)) = M(B)

$$M(A) \rightarrow [\ A \rightarrow M(B)\ ] \rightarrow M(B)$$

Операция Bind (Аксиомы)

bind(η(a), f) = f(a) - левая единица

bind(ma, η) = ma - правая единица

bind(bind(ma, f), g) = bind(ma, bind(f(x), g)) - ассоциативность

bind(ma, f) = μ ∘ M(f)(ma) - композиция

Пример

```
public static Maybe<U> Bind<T, U>(Maybe<T> maybe, Func<T, Maybe<U>> f)
{
    return maybe.HasValue ? f(maybe.Value) : Maybe<U>.None;
}
```

Что даёт?

Композиция

Ассоциативность

Согласованность

Переиспользование

Предсказуемость

Пример

```
public static string GetUserEmail(int userId)
{
    var user = GetUser(userId);
    if (user == null) return null;

    var profile = GetProfile(user.Id);
    if (profile == null) return null;

    return profile.Email;
}
```

Пример

```
static Maybe<string> GetUserEmail(int userId)
{
    return Maybe<int>.Some(userId)           // Maybe<int>
        .Bind(id => GetUser(id))             // Maybe<User>
        .Bind(user => GetProfile(user.Id))   // Maybe<Profile>
        .Map(profile => profile.Email);      // Maybe<string>
}
```

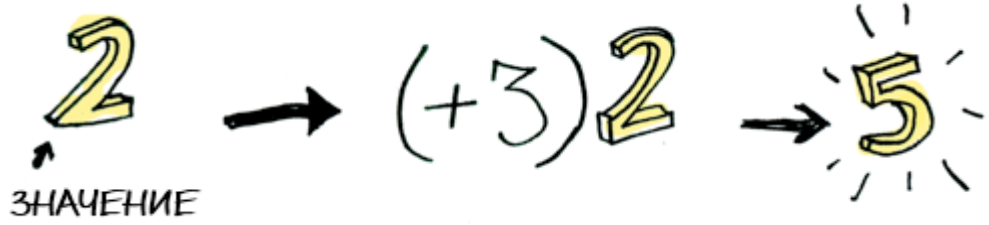
Пример

```
public Maybe<U> Select<U>(Func<T, U> selector)
{
    return Map(selector);
}

public Maybe<U> SelectMany<U>(Func<T, Maybe<U>> selector)
{
    return Bind(selector);
}

public Maybe<V> SelectMany<U, V>(Func<T, Maybe<U>> selector, Func<T, U, V> resultSelector)
{
    if (!HasValue) return Maybe<V>.None;

    var selected = selector(Value);
    if (!selected.HasValue) return Maybe<V>.None;

    return Maybe<V>.Some(resultSelector(Value, selected.Value));
}
```

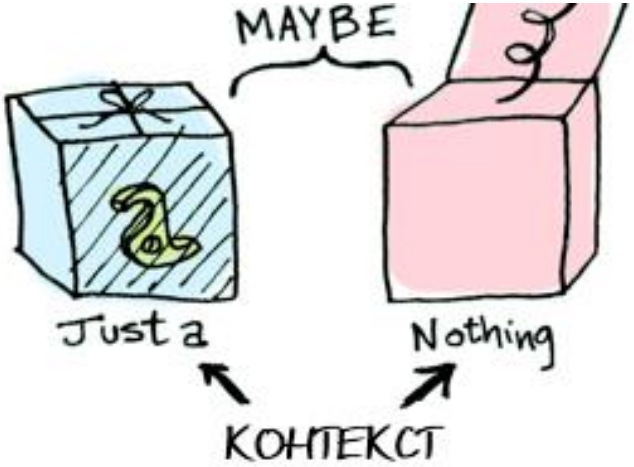
Пример

```
static Maybe<string> GetUserEmail(int userId)
{
    return from user in GetUser(userId)
           from profile in GetProfile(user.Id)
           select profile.Email;
}
```

Аналогия



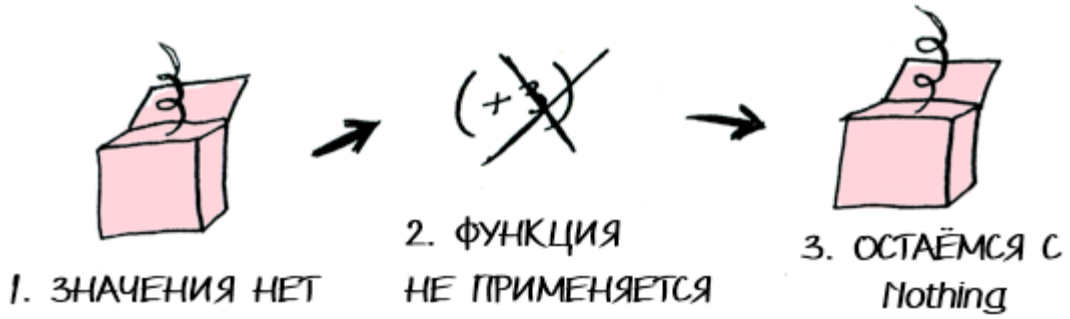
Аналогия



Аналогия


(+3)
2
5
2. ПРИМЕНЯЕТ
К НЕМУ ФУНКЦИЮ
1. РАЗВОРАЧИВАЕТ ЗНАЧЕНИЕ
ИЗ КОНТЕКСТА
3. ВНОВЬ ОБОРАЧИВАЕТ
ЗНАЧЕНИЕ В КОНТЕКСТ

Аналогия


1. ЗНАЧЕНИЯ НЕТ
2. ФУНКЦИЯ НЕ ПРИМЕНЯЕТСЯ
3. ОСТАЁМСЯ С Nothing

Какие есть библиотеки?

- **LanguageExt** — самая полная: Option, Either, Try, Validation, Reader/State/Writer, LINQ-выражения, коллекции. Отлична для демонстраций и продакшена. https://github.com/louthy/language-ext
- **CSharpFunctionalExtensions** — практичный набор: Result, Maybe, Either (в новых версиях), много удобных экстеншенов и Railway Oriented. Отлично заходит в бизнес-код. https://github.com/vkhorikov/CSharpFunctionalExtensions
- **Optional** — минималистичный Option<T>/Maybe<T> с LINQ. Хорош для простых кейсов/учебных примеров. https://github.com/nlkl/Optional

Что если надо ловить ошибки? (Монада Either)

```
private static Either<string, int> ParseQty(string qtyText)
    => int.TryParse(qtyText, out var q) && q > 0
        ? Either<string, int>.Right(q)
        : Either<string, int>.Left($"Количество некорректно: '{qtyText}'");

private static Either<string, decimal> FindPrice(string sku)
    => Prices.TryGetValue(sku, out var p)
        ? Either<string, decimal>.Right(p)
        : Either<string, decimal>.Left($"Неизвестный товар: '{sku}'");

private static Either<string, decimal> CalcTotal(OrderRequest req)
    => from qty in ParseQty(req.QtyText)
       from price in FindPrice(req.Sku)
       select qty * price;

private static void DemoGood(OrderRequest req)
{
    var result = CalcTotal(req);

    result.Match(
        Right: total => Console.WriteLine($"Заказ: {req.Sku} x{req.QtyText} = {total:C}"),
        Left:  err   => Console.WriteLine($"Ошибка: {err}")
    );
}
```

Какие монады уже есть в C#

Встроенные:

- IEnumerable<T>: последовательности (LINQ).
- Task<T>/ValueTask<T>: композиция асинхронных вычислений без if/try. (async await как сахар)
- IAsyncEnumerable<T>: асинхронные последовательности (await foreach + LINQ).
- Nullable<T> (T?): иногда рассматривают как Maybe

Еще монады

```
public readonly struct Log<T>
{
    public T Value { get; }
    public IReadOnlyList<string> Messages { get; }

    public Log(T value, IReadOnlyList<string> messages)
    {
        Value = value;
        Messages = messages;
    }
}
```

Еще монады

```
public Log<R> Map<R>(Func<T, R> f)
    => new Log<R>(f(Value), Messages);

public Log<R> Bind<R>(Func<T, Log<R>> f)
{
    var next = f(Value);
    var combined = new List<string>(capacity: Messages.Count + next.Messages.Count);
    combined.AddRange(Messages);
    combined.AddRange(next.Messages);
    return new Log<R>(next.Value, combined);
}

public Log<R> Select<R>(Func<T, R> f) => Map(f);

public Log<R> SelectMany<U, R>(Func<T, Log<U>> bind, Func<T, U, R> project)
{
    return Bind(t =>
    {
        var u = bind(t);
        return new Log<R>(project(t, u.Value), u.Messages);
    });
}
```

Изобретаем свою монаду

```
private static Log<decimal> GetBasePrice(string sku)
{
    var price = sku == "SKU1" ? 100m : 50m;
    return new Log<decimal>(price,
        messages: new[] { $"Базовая цена для {sku}: {price:C}" });
}

private static Log<decimal> ApplyDiscount(decimal price, decimal rate)
{
    var p = Math.Round(price * (1 - rate), 2);
    return new Log<decimal>(p,
        messages: new[] { $"Скидка {rate:P0}: {price:C} → {p:C}" });
}

private static Log<decimal> ApplyTax(decimal price, decimal rate)
{
    var p = Math.Round(price * (1 + rate), 2);
    return new Log<decimal>(p,
        messages: new[] { $"НДС {rate:P0}: {price:C} → {p:C}" });
}
```

Изобретаем свою монаду

```
public static void Foo4()
{
    var result =
        from basePrice in GetBasePrice("SKU1")
        from discounted in ApplyDiscount(basePrice, rate: 0.1m)
        from taxed in ApplyTax(discounted, rate: 0.2m)
        select taxed;

    Console.WriteLine($"[Writer] Итог: {result.Value:C}");

    foreach (var m in result.Messages)
        Console.WriteLine($" → {m}");
}
```

Изобретаем свою монаду

```
[Writer] Итог: 108,00 ₽
 → Базовая цена для SKU1: 100,00 ₽
 → Скидка 10%: 100,00 ₽ → 90,00 ₽
 → НДС 20%: 90,00 ₽ → 108,00 ₽
```

Выводы

Монады - мощный инструмент из теории категорий, позволяющий создавать гибкий, переиспользуемый, предсказуемый и безопасный код