OM FEST 2025

ФЕСТИВАЛЬ ЦИФРОВЫХ ТЕХНОЛОГИЙ

ПРАВИТЕЛЬСТВО ОМСКОЙ ОБЛАСТИ

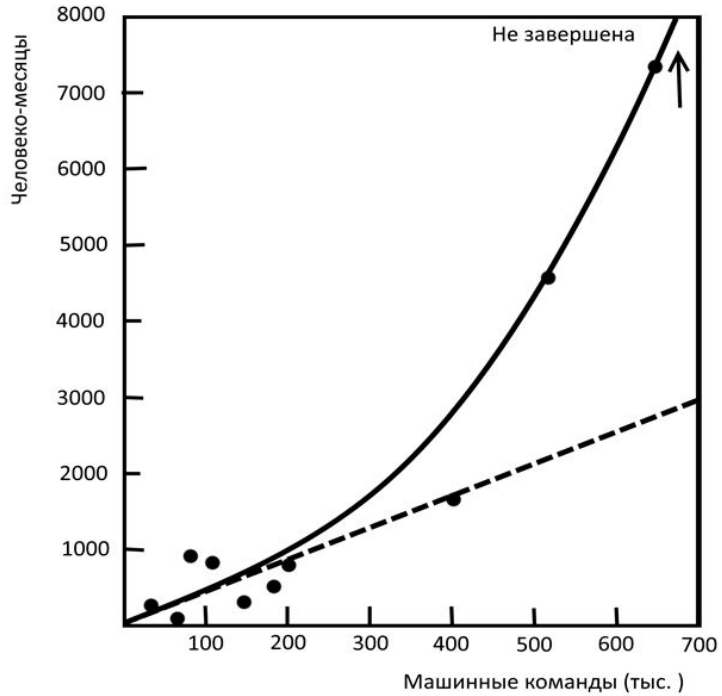
Спецификация против стохастики: баланс гибкости моделей и строгих гарантий

Иван Шарун
ОмГТУ, ФИТиКС, ПМиФИ,
аспирант, ст. преп.

План

1. Абстрагирование
2. Откуда корни проблемы?
3. Где гарантии?
4. Заключение
5. Послесловие

Скорость разработки падает с ростом проекта



Nanus B., Farr L.

Some cost contributors to large-scale programs //

AFIPS Proc. SJCC. Spring 1964. Vol. 25. P. 239-248.

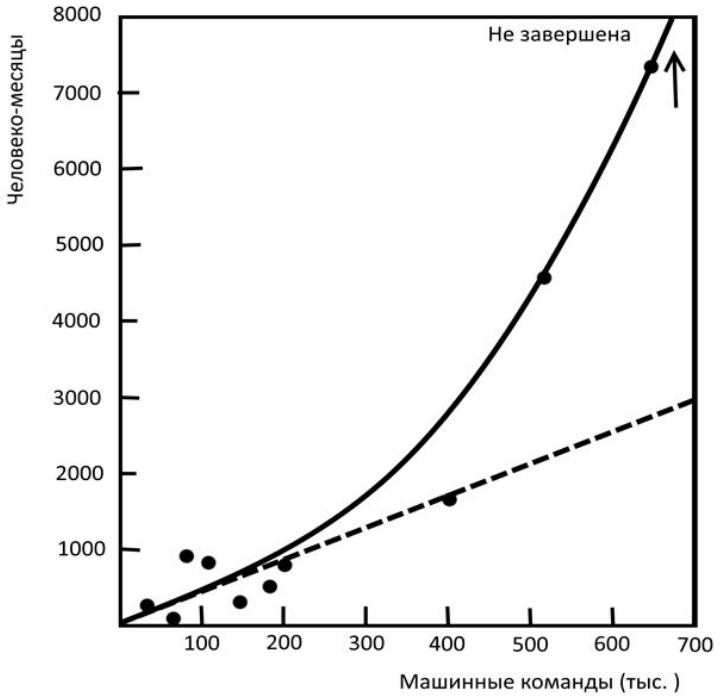
Степенная функция с показателем 1.5

OCP — достаточное условие постоянной скорости разработки

Тюменцев Е.А. О формализации процесса разработки программного кода

в части “закрыты от модификации”

справедливы для любого языка программирования

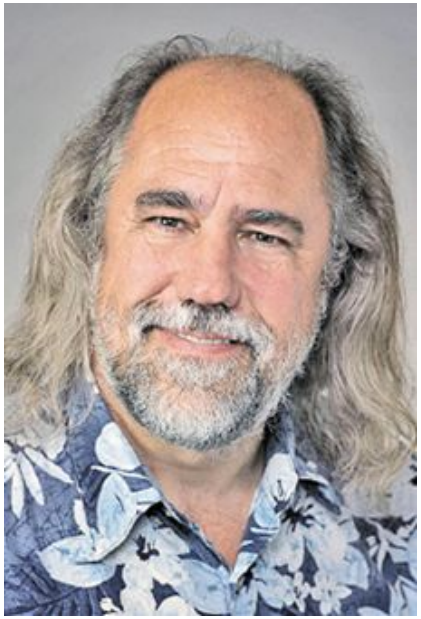


https://cyberleninka.ru/article/n/o-formalizatsii-protsessa-razrabotki-programmnogo-obespecheniya/viewer

Абстрагирование

Определение абстракции

«Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом четко определяет его концептуальные границы с точки зрения наблюдателя»



Гради Буч

Алфавит, слово

{0 1 2 3 4 5 6 7 8 9}

{а б в г д … э ю я}

{H He Li … Ts Og}

Алфавит, слово

{0 1 2 3 4 5 6 7 8 9}

{а б в г д … э ю я}

{H He Li … Ts Og}

{int double class { } () , ; public private }

Алфавит, слово

{0 1 2 3 4 5 6 7 8 9}	23 123 56325
{а б в г д … э ю я}	мама мыла раму
{H He Li … Ts Og}	NaCL
{int double class { } () , ; public private }	class A{};

Формальный язык

Σ - алфавит

Σ^* - множество всех слов в алфавите Σ

Тогда $L \subset \Sigma^*$ - формальный язык над алфавитом Σ.

Исчисление

Пусть L - формальный язык над алфавитом Σ.

Ax - множество слов языка L.

R - конечное множество, не менее, чем двухместных отношений на L.

Тогда на L задано исчисление или дедуктивная система

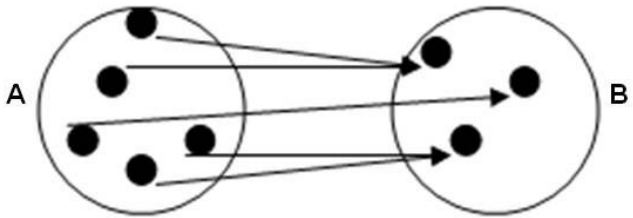
Изоморфизм Карри — Ховарда

OM FEST 2025

- формула — тип
- высказывательная переменная — типовая переменная
- логическая связка — конструктор типа;
- импликация — пространство функций;
- конъюнкция — произведение;
- дизъюнкция — копроизведение;
- дно — ненаселённый (пустой) тип;
- доказательство — терм (программа);
- введение связки — конструктор значения;
- удаление связки — деструктор значения;
- доказуемость — населённость;
- теорема — населённый (замкнутый) тип

Сюръективное отображение

На множество - «сюръекция»



Соответствие. при котором каждому элементу множества *A* указан *единственный* элемент множества *B*, а каждому элементу множества *B* можно указать *хотя бы* один элемент множества *A*, называется отображением множества *A* **на** множество *B*

Определение абстракции

Пусть Σ – произвольный алфавит, $Abs \subseteq \Sigma^*$ – некоторое подмножество Σ^*, X – произвольное множество.

Тогда сюръективное отображение $F:X \rightarrow Abs$ называется **абстрагированием** множества X над алфавитом Σ.

Элементы множества X – **сущности**.

Элементы множества Abs – **абстракции**.

Тюменцев Е.А. "О формальном определении абстракции"// Математические структуры и моделирование 2018. No 1(45). С. 131–143.

Обобщение

Несколько сущностей -> одна абстракция

Определение обобщения

Пусть Σ – некоторый алфавит, тогда будем называть абстрагирование $F: X \to A, A \subseteq \Sigma^*$ **абстрагированием с обобщением**, если и только если $\exists x, y \in X: F(x) = F(y)$.

Свойства абстракций

1) Любое абстрагирование должно быть обобщением

Любое абстрагирование должно быть обобщением

Пусть Σ – произвольный алфавит символов, w – некоторое слово в этом алфавите, $F : X \to A$, где $A \subset \Sigma^*$, – некоторое абстрагирование множества X над алфавитом Σ.

Предположим, что слово w содержит все абстракции множества A в качестве подслов.

Тогда:

либо F является абстрагированием с обобщением,

либо X – конечное множество,

либо F определено частично на множестве X.

Свойства абстракций

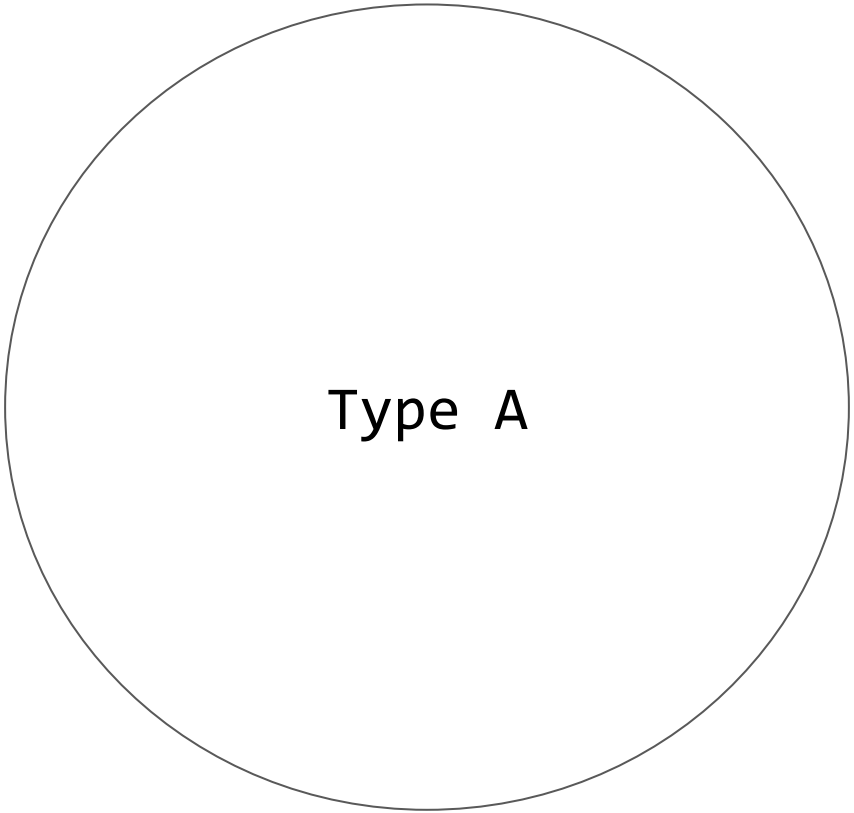
1) Любое абстрагирование должно быть обобщением
2) Множество абстракций должно состоять из одного элемента

Множество абстракций должно состоять из одного элемента

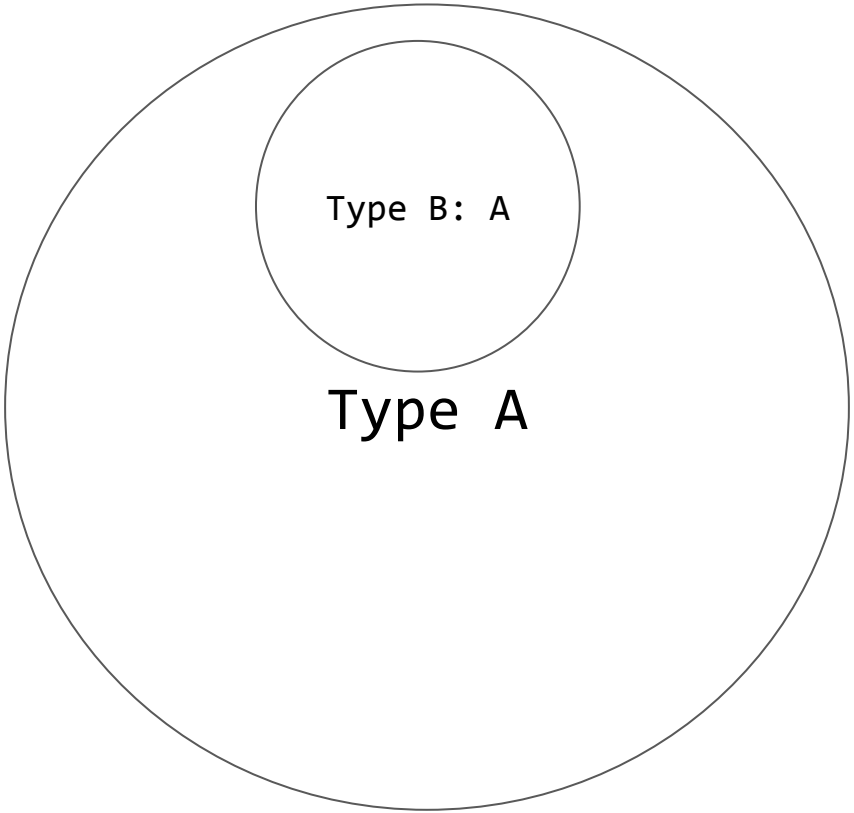
Пусть Σ – произвольный алфавит символов, $F : X \to A$, где $A \subset \Sigma^*$, – некоторое абстрагирование с обобщением множества X над алфавитом Σ, причём $|A| > 1$. Предположим, что Y – множество такое, что $|Y| > 1$, $id : Y \to AY$ – взаимно-однозначное абстрагирование множества Y над алфавитом Σ. Тогда существует такое отображение $G : X \to Y$, для которого не существует $g : A \to AY$, чтобы следующая диаграмма была коммутативной:

$$\begin{array}{ccc} X & \overset{F}{\to} & A \\ G\downarrow & & \downarrow g \\ Y & \overset{id}{\to} & AY \end{array}$$

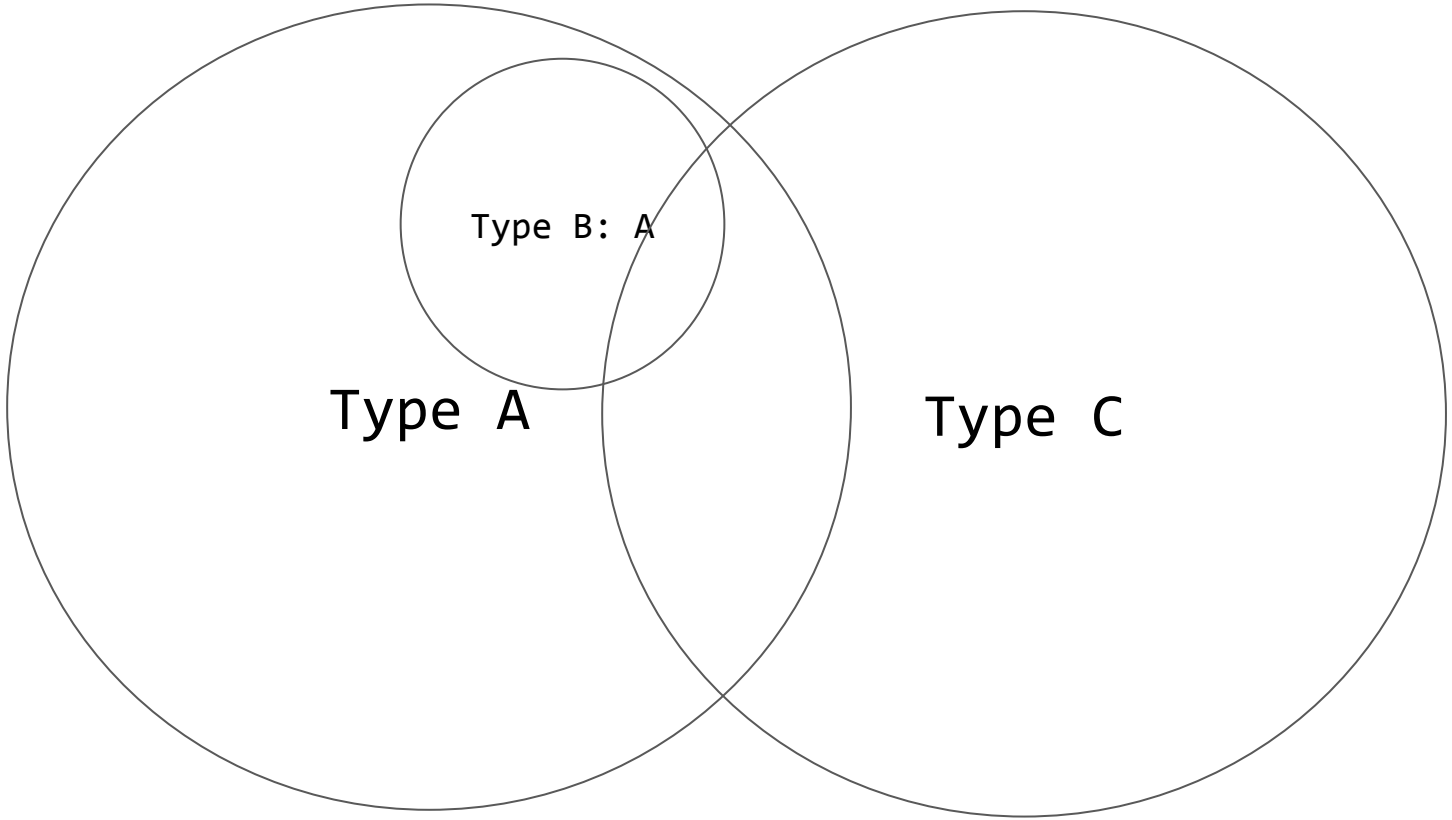
Каждая новая строка кода — ограничение



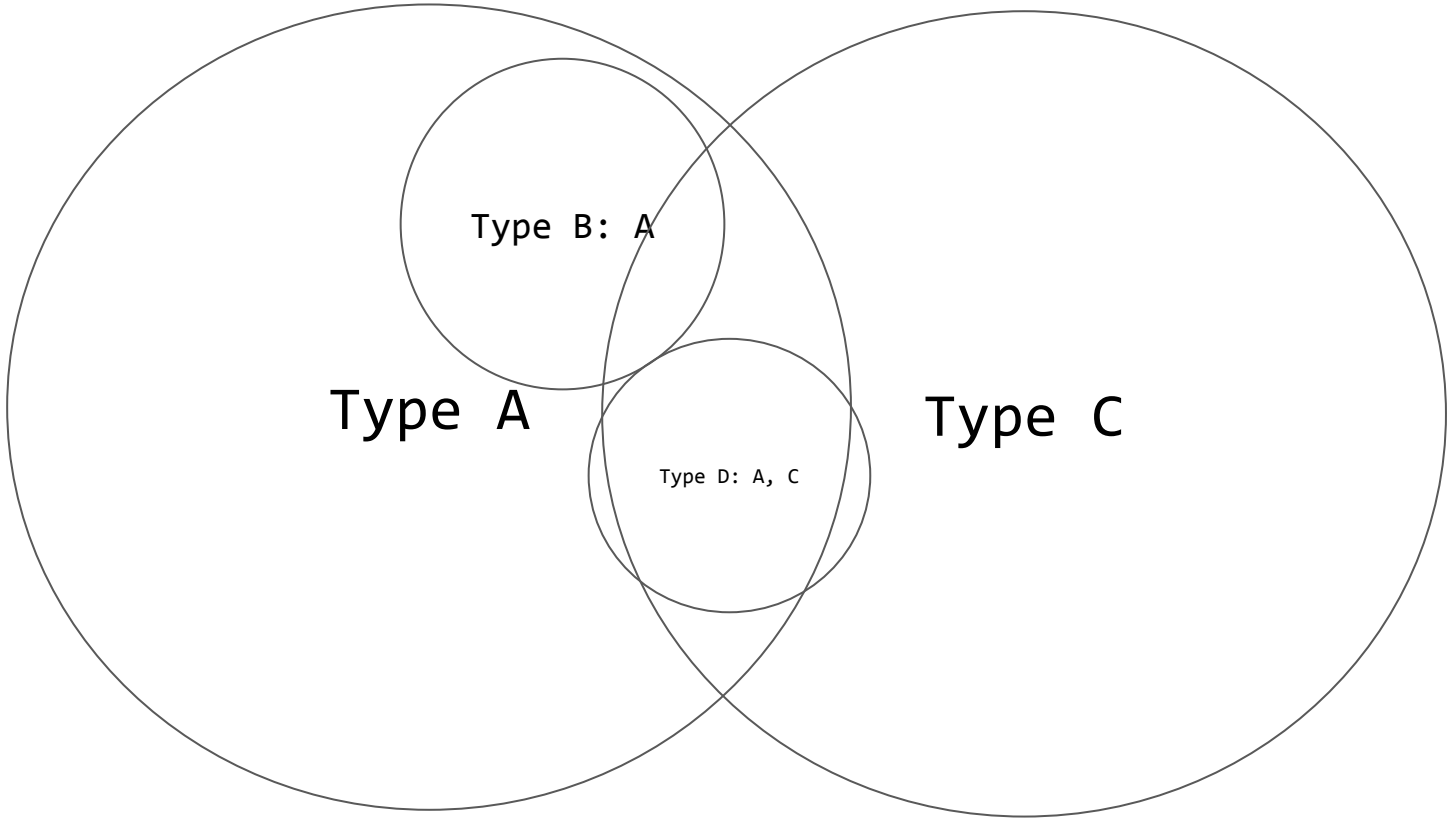
Каждая новая строка кода — ограничение



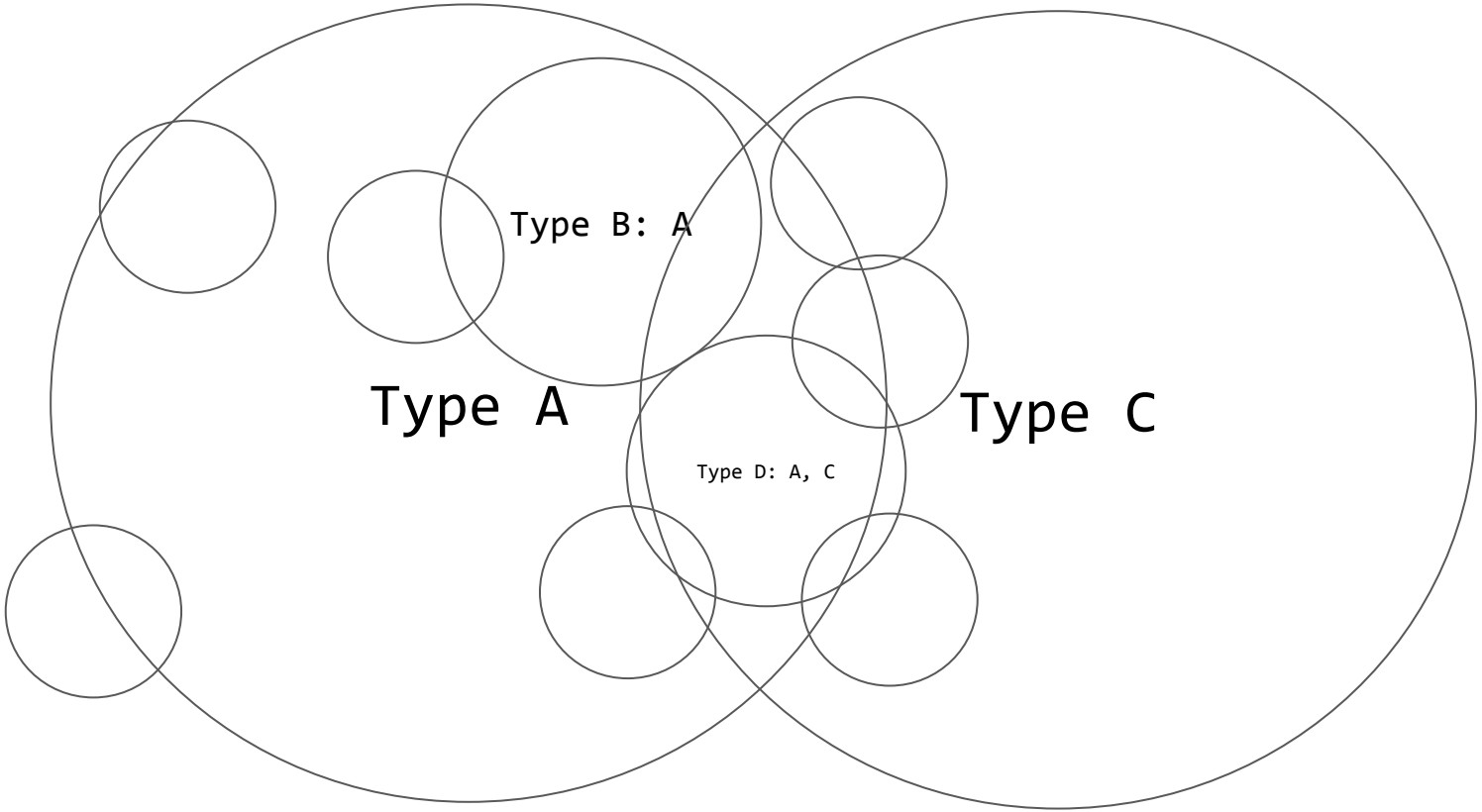
Каждая новая строка кода — ограничение



Каждая новая строка кода — ограничение



Каждая новая строка кода — ограничение



Откуда корни проблемы?

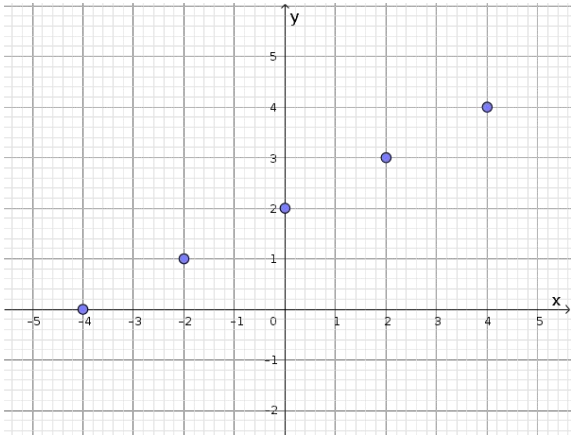
Задача аппроксимации

$$f: X \to Y$$

Аналитический

$$f(x) = kx+b$$

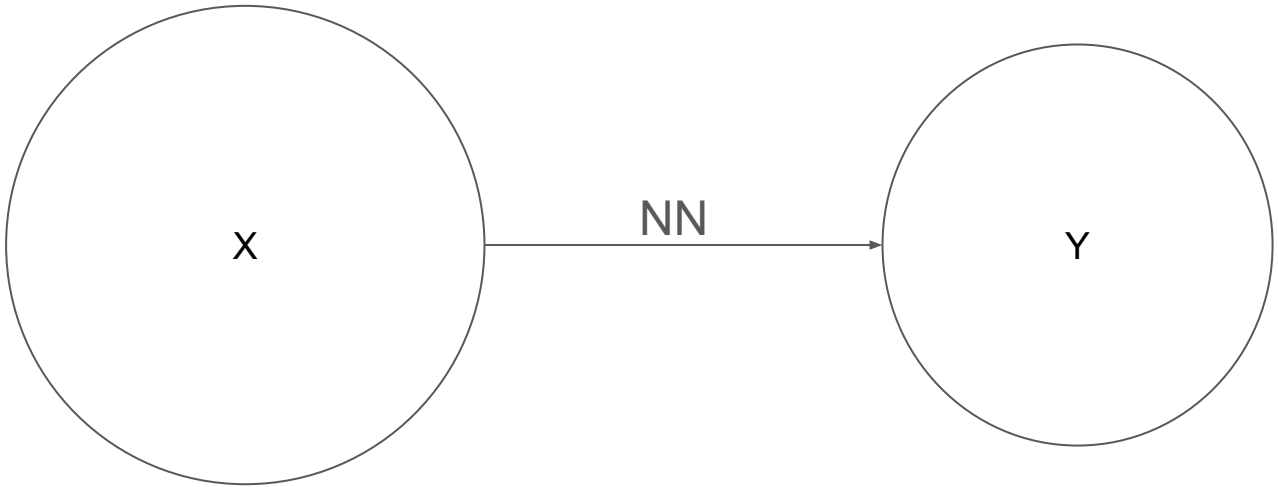
Графический



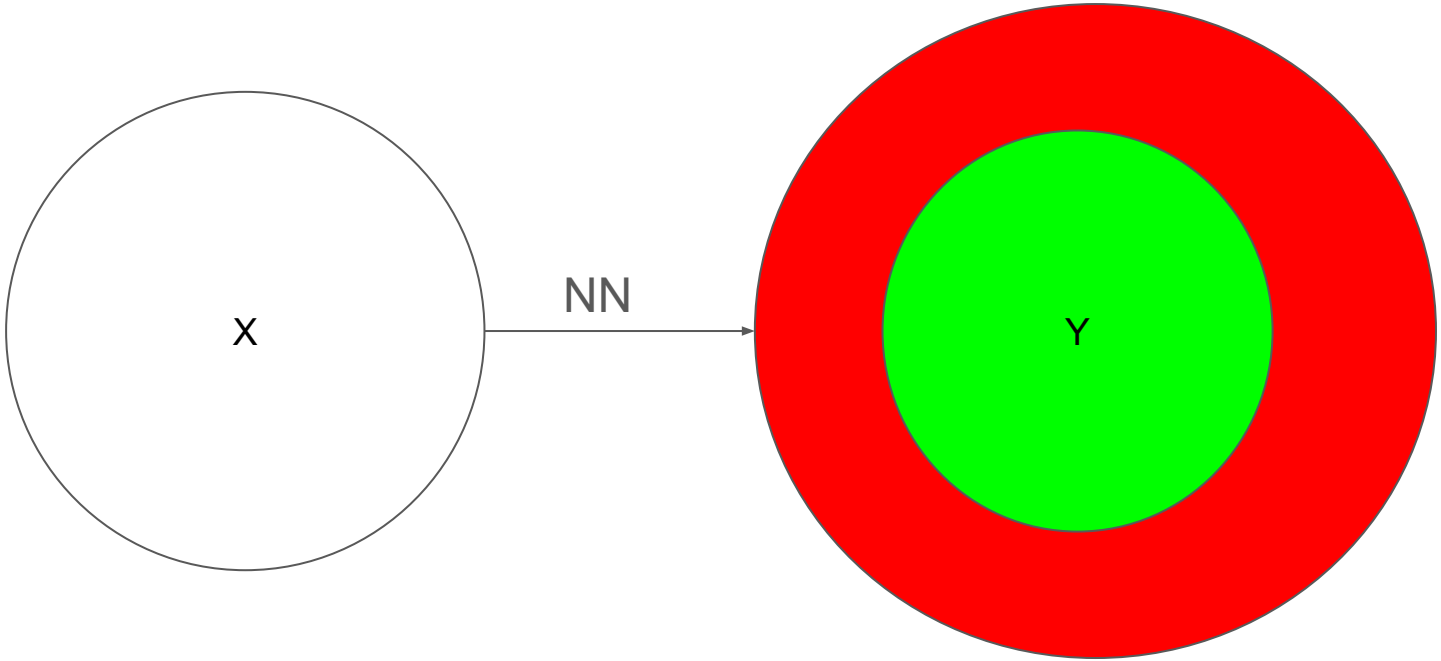
Табличный

аллюр	СУЩЕСТВИТЕЛЬНОЕ
квелая	ПРИЛАГАТЕЛЬНОЕ
Мягковыми	СУЩЕСТВИТЕЛЬНОЕ
прокутившая	ПРИЛАГАТЕЛЬНОЕ
проповедываем	ГЛАГОЛ
эхоэнцефалограмм	СУЩЕСТВИТЕЛЬНОЕ
батызе	СУЩЕСТВИТЕЛЬНОЕ
отапливающей	ПРИЛАГАТЕЛЬНОЕ
поддернувшаяся	ПРИЛАГАТЕЛЬНОЕ
переподписывало	ГЛАГОЛ
расплавливании	СУЩЕСТВИТЕЛЬНОЕ
стегануло	ГЛАГОЛ
Хен	СУЩЕСТВИТЕЛЬНОЕ
ельцинцев	СУЩЕСТВИТЕЛЬНОЕ
лесоперерабатывающей	ПРИЛАГАТЕЛЬНОЕ

Задача аппроксимации



Задача аппроксимации



Где гарантии?

Программист делает в среднем *M* ошибок на *N* строчек.
M != 0

TLA+	Логика действий во времени	Верификация распределённых алгоритмов
π-исчисление	Модель мобильных процессов	Анализ взаимодействия в динамических системах
Акторная модель	Обмен сообщениями между автономными агентами	Параллельное программирование (Сверхтьюринговые вычисления)
Модель-чекеры (TLC, SPIN)	Перебор состояний системы	Поиск гонок, тупиков, ошибок логики
Интерактивные пруверы (Coq, Lean)	Доказательство теорем о программе	Высокоудостоверенные системы

Спецификация (coq)

```
Fixpoint fact_std (n : nat) : nat :=
  match n with
  | 0 => 1
  | S n' => (S n') * fact_std n'
  end.
```

Спецификация (coq)

```
Fixpoint fact_acc (n acc : nat) : nat :=
  match n with
  | 0 => acc
  | S n' => fact_acc n' (acc * S n')
  end.

Definition fact (n : nat) : nat := fact_acc n 1.
```

Спецификация (coq)

```
Lemma fact_acc_correct' : forall n acc : nat,
  fact_acc n acc = acc * fact_std n.
Proof.
  induction n as [| n' IH]; intros acc.
  - simpl. reflexivity.
  - simpl. apply IH.
Qed.

Corollary fact_correct : forall n, fact n = fact_std n.
Proof.
  unfold fact. intros n.
  rewrite (fact_acc_correct' n 1).
  simpl. rewrite Nat.mul_1_r. reflexivity.
Qed.
```

Результат (python)

```python
from typing import Tuple

def fact_acc(n: int, acc: int) -> int:
    if n <= 0:
        return acc
    return fact_acc(n - 1, acc * n)

def fact(n: int) -> int:
    return fact_acc(n, 1)
```

Как это применить сейчас?

Генерировать не код, а формальные спецификации

Программирование — искусство?

OM FEST 2025

ФЕСТИВАЛЬ ЦИФРОВЫХ ТЕХНОЛОГИЙ

ПРАВИТЕЛЬСТВО ОМСКОЙ ОБЛАСТИ

Иван Шарун
ОмГТУ, ФИТиКС, ПМиФИ,
аспирант, ст. преп.

@zaryanezrya
ivan@sha.run