Тинькофф
Университет

Теория доказательств и верификация программ

Введение

Евгений Ивашкевич

Цена ошибок программирования

- Сбои в работе медицинского оборудования «Therac-25»
- Сбой в работе телефонной сети «AT&T»
- Ошибка деления в процессоре «Intel Pentium»
- Взрыв ракеты «Ariane 5»
- Использование разных версий ПО при разработке «Airbus A380»
- Кража «Эфира»

Запрограммированное убийство

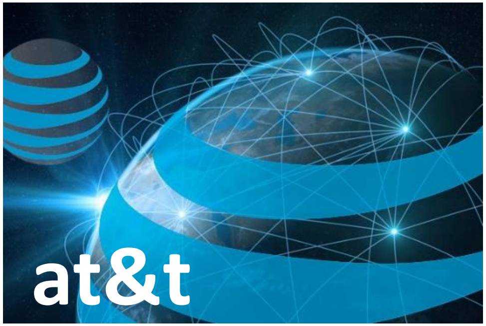
Во время сеансов радиационной терапии на установке «Therac-25», принадлежащей Канадскому Агентству Атомной Энергии, смертельную дозу облучения получили 6 человек. Количество пациентов, подвергшихся переоблучению, но выживших, точно не известно.

В результате расследования выяснилось, что все программы управления установкой были написаны на ассемблере, были плохо документированы и содержали множество ошибок.

DESIINE SPERARE QUI HIS INTRAS

Сбой в работе телефонной сети «AT&T»

15 января 1990 г. Около 75 миллионов абонентов телекоммуникационной компании «AT&T» остались без связи. Согласно стандартной логике работы междугороднего коммутатора, перезагружаются, если получает соответствующий сигнал от соседнего коммутатора.



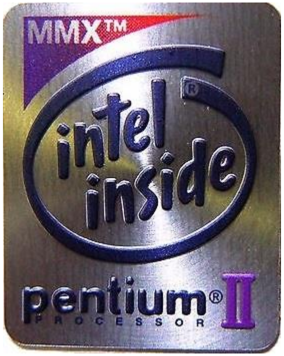
Ошибка в новой версии прошивки коммутаторов привела к тому, что при восстановлении после сбоя каждый коммутатор повторно отправлял сигнал на перезагрузку соседних коммутаторов.

Все началось с падения и перезагрузки коммутатора в Нью-Йорке, который вызвал масштабную цепную реакцию, в результате которой все 114 коммутаторов сети перезагружались непрерывно каждые 6 секунд в течение 9 часов.

VANITAS VANITATUM AT OMNIA VANITAS

Ошибка деления в процессоре «Pentium»

Ошибка в модуле операций с плавающей запятой была впервые обнаружена и опубликована профессором Линчбургского колледжа Томасом Найсли в октябре 1994 года. При делении чисел с плавающей запятой при помощи математического сопроцессора в некоторых случаях результат получался некорректным.



И `4195835.-3145727.*(4195835./3145727.) = 0`

Л `4195835.-3145727.*(4195835./3145727.) = 256`

Согласно заявлению Intel причиной проблемы послужили неточности в таблице поиска, используемой при проведении операции деления

EADEM OBERRARE CHORDA

Взрыв ракеты «Ariane 5»

4 июня 1996 г. ракета «Ariane 5», на борту которой были 4 научных спутника, успешно стартовала с космодрома Куру во Французской Гвиане. Полет продолжался нормально первые 36 секунд полета. Далее ракета внезапно изменила направление полета и взорвалась.

В программе системы управления было предусмотрено преобразование 64-битового числа с плавающей точкой в 16-битовое целое число. Для больших чисел это преобразование выполнялось некорректно.

SIC ITUR AD ASTRA

Устаревшее ПО в «Airbus A380»

Немецкое и французское подразделение авиастроительной корпорации Airbus занимались разработкой двух разных частей нового борта A380. При проектировании коммуникационных сетей они использовали одну и ту же программу CATIA, но разных версий.

При совмещении двух частей проекта инженеры не смогли соединить друг с другом некоторые кабели. Учитывая сложность проекта, разрешение этой проблемы отсрочило официальный запуск нового самолета на целый год.

BARBARUS HIC EGO SUM, QUIA NON INTELLIGOR ULLI

Кража «Эфира»

Ethereum — проект создания различных платформ на базе распределенного реестра (блокчейна). 17июня 2016 года неизвестный хакер стал массово выводить единицы Эфира, на базе которых работает фонд “The DAO”. Это привело к краже $53 миллионов, а также к обвалу курса единицы Эфира и DAO на криптовалютных биржах.

Мошенник воспользовался стандартной пользовательской функцией, которая вознаграждает пользователей бонусными единицами Эфира в процессе разделения фонда. Оказалось, что эту функцию можно было вызывать рекурсивно.

Стандартный цикл разработки программного обеспечения

Бизнес-требования

Проектирование

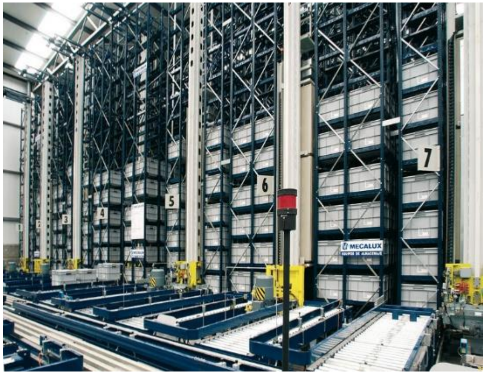
Кодирование

Тестирование

Внедрение

Бизнес-требования

Недавно интернет-гигант «Amazon» представил свой новый склад восьмого поколения, в котором производится полностью автоматизированная доставка товара со складов к сортировщикам.



Несколько сотен миниатюрных роботов передвигаются по складам и перемещают на себе небольшие стеллажи, на которых находятся необходимые товары. Сами сотрудники склада при этом никуда не ходят: они лишь стоят и ждут, когда робот привезет им необходимый стеллаж.

Разумеется, для такого склада требуется высоко-интеллектуальное программный комплекс. Основной его частью является **программа сортировки грузов**.

Проектирование

Для того чтобы информацию о товаре мог читать не только человек, но и компьютер, используются штрих-коды. При этом каждая единица хранения представляется числом, а программа сортировки грузов, таким образом, сводится к программе **сортировки списка натуральных чисел**.

В результате технолог пишет кодировщику следующее задание:

Техническое задание

Написать программу, которая принимает в качестве аргумента произвольный список натуральных чисел и возвращает другой, упорядоченный, список, который состоит из тех же элементов, что и исходный.

Проектирование

После некоторого размышления кодировщик приходит с вопросами:

1) Должен ли возвращаемый список быть упорядочен по возрастанию или по убыванию?
2) Должен ли возвращаемый список иметь ту же длину, что и исходный?

Технолог вносит пояснения в техническое задание и одновременно формулирует задание для тестировщиков:

Задание тестировщику

1) **Sort** [10; 9; 8; 7; 6; 5; 4; 3; 2; 1; 0]
2) **Sort** [1; 0; 6; 4; 9; 2; 10; 7; 5; 3; 8]
3) **Sort** [4; 7; 3; 9; 8; 5; 0; 10; 1; 6; 2]

$\Rightarrow$ [0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10]

</> Кодирование

Программист пишет простой и элегантный код программы сортировки:

```
Fixpoint Insert (n : nat) (x : list nat) : list nat :=
 match x with
  | [] ⇒ [n]
  | h :: t ⇒ if (n ≤ h)
               then (h :: n :: t)
               else (h :: Insert n t)
 end.

Fixpoint Sort (x : list nat) : list nat :=
 match x with
  | [] ⇒ []
  | h :: t ⇒ Insert h (Sort t)
 end.
```

Тестирование

Compute (**Sort** [10; 9; 8; 7; 6; 5; 4; 3; 2; 1; 0]).

⇒ [0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10]

Тестирование

Compute (**Sort** [10; 9; 8; 7; 6; 5; 4; 3; 2; 1; 0]).

⇒ [0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10]

Compute (**Sort** [1; 0; 6; 4; 9; 2; 10; 7; 5; 3; 8]).

⇒ [8; 1; 0; 6; 4; 2; 7; 5; 3; 10; 9]

Compute (**Sort** [4; 7; 3; 9; 8; 5; 0; 10; 1; 6; 2]).

⇒ [2; 0; 1; 6; 4; 3; 5; 10; 7; 9; 8]

Что-то пошло не так...

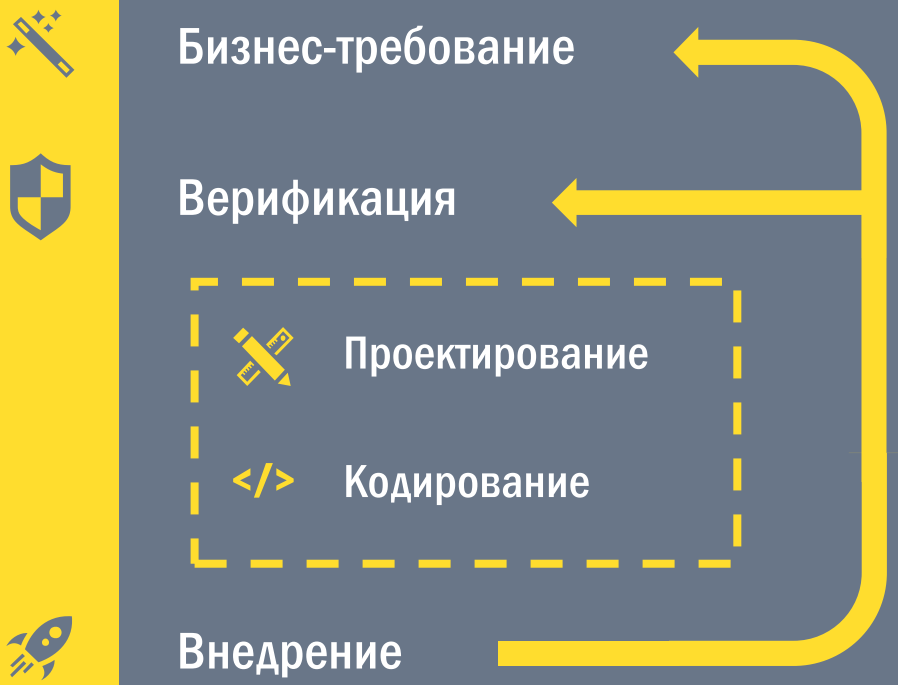
</> Кодирование

```
Fixpoint Insert (n : nat) (x : list nat) : list nat :=
 match x with
  | [] ⇒ [n]
  | h :: t ⇒ if (n ≤ h)
               then (h :: n :: t)
               else (h :: Insert n t)
 end.
```

ошибка!

```
Fixpoint Sort (x : list nat) : list nat :=
 match x with
  | [] ⇒ []
  | h :: t ⇒ Insert h (Sort t)
 end.
```

Цикл разработки защищённого программного обеспечения в Coq



Проектирование

При разработке защищенного ПО основная задача проектировщика заключается не столько в том, чтобы дать словесное описание задачи и требуемого результата, сколько в том, чтобы **создать макро-язык для полного формального описания задачи**.

Теперь техническое задание для программы сортировки списка натуральных чисел будет выглядеть так:

ТЗ : спецификация функции сортировки

Program Sort (x : list nat) :
{ y : list nat | **Ordered** y ∧ **Permutation** x y }

Это формализованное ТЗ показывает, что для полного описания задачи нам необходимо определить только два предиката:

1) свойство упорядоченности списка
2) отношение перестановочной эквивалентности двух списков.

Проектирование

Мы говорим, что список упорядочен, если он:

1) пуст
2) состоит из одного единственного элемента
3) состоит из двух элементов и его первый элемент не больше второго
4) состоит из трех элементов, его первый элемент не больше второго, и при этом «хвост» из последних двух элементов упорядочен

…

ТЗ : спецификация свойства упорядоченности

```
Inductive Ordered : list nat → Prop :=
  | Ord0 : Ordered []
  | Ord1 (n : nat) : Ordered [n]
  | Ord2 (n m : nat) (x : list nat) :
        n ≤ m → Ordered (m :: x) → Ordered (n :: m :: x).
```

Проектирование

Из комбинаторики известно, что отношение перестановочной эквивалентности полностью определяется следующими свойствами:

ТЗ : спецификация отношения перестановки

Lemma PermEquiv :
$\forall$ (x y z : list nat),
1) $x \sim x$
2) $(x \sim y) \to (y \sim x)$
3) $(x \sim y) \to (y \sim z) \to (x \sim z)$.

Notation " x ~ y "
:= **Permutation** x y.

Lemma PermSkip :
$\forall$ (n : nat) (x y : list nat), $(x \sim y) \Leftrightarrow (n :: x) \sim (n :: y)$.

Lemma PermSwap :
$\forall$ (n m : nat) (x : list nat), $(n :: m :: x) \sim (m :: n :: x)$.

</> Кодирование

```
Fixpoint Insert (n : nat) (x : list nat) :
 { y : list nat | y ~ (n :: x) /\ (Ordered x → Ordered y)} :=
 match x with
  | [] ⇒ [n]
  | h :: t ⇒ if (n ≤ h)
               then (n :: h :: t)
               else (h :: Insert n t)
 end.

Fixpoint Sort (x : list nat) :
 { y : list nat | Ordered y ∧ Permutation x y } :=
 match x with
  | [] ⇒ []
  | h :: t ⇒ Insert h (Sort t)
 end.
```

спецификация!

Верификация

Obligation 1 of **Insert** :

$$\forall\ (n : nat)\ (x : list\ nat),$$
$$[] = x \rightarrow [n] \sim [n] \wedge (\mathbf{Ordered}\ [] \rightarrow \mathbf{Ordered}\ [n]).$$

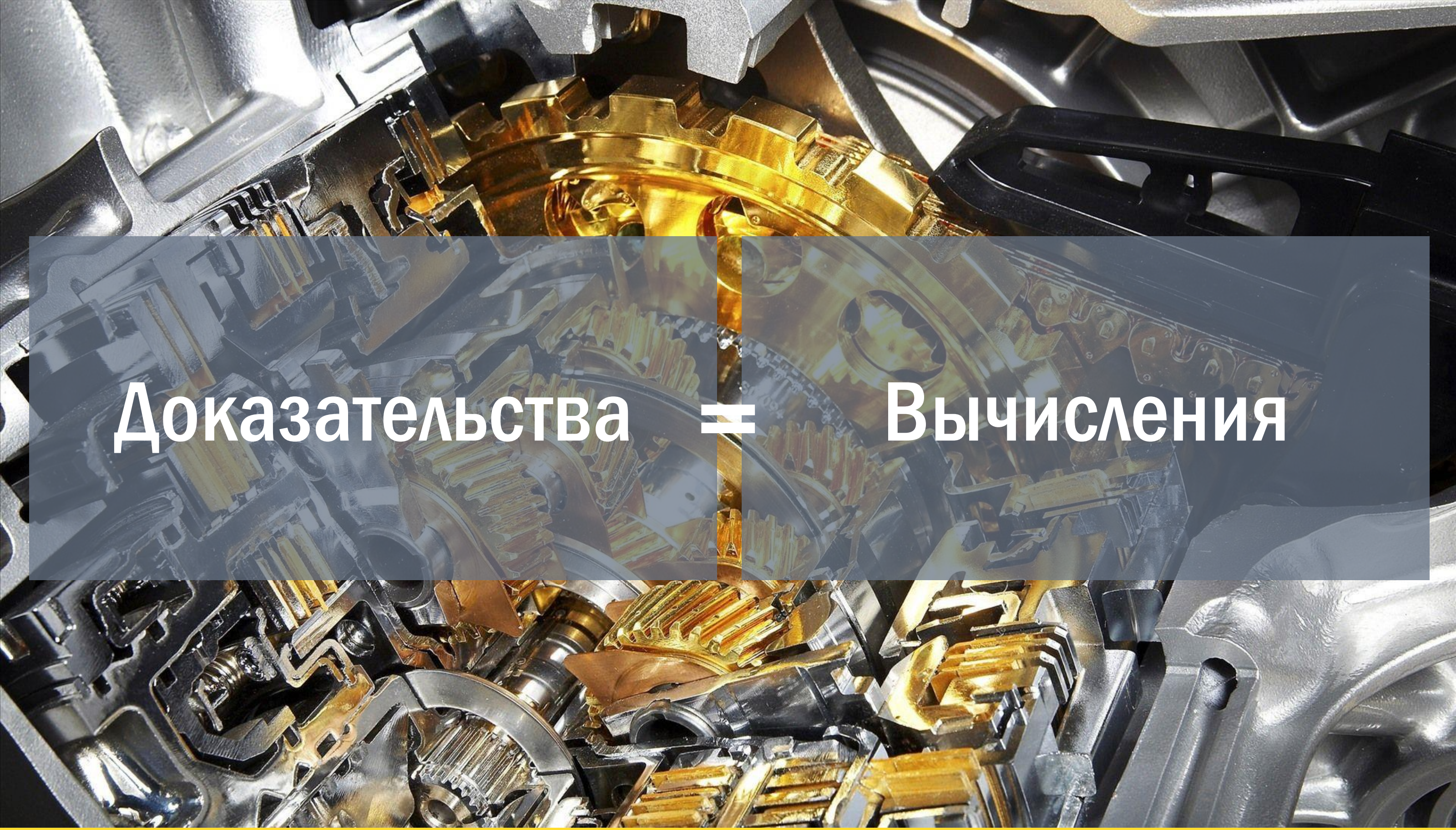
Obligation 2 of **Insert** :

$$\forall\ (n\ h : nat)\ (x\ t : list\ nat),$$
$$n \leq h \rightarrow x = (h :: t)$$
$$\rightarrow (n :: x) \sim (n :: x) \wedge (\mathbf{Ordered}\ x \rightarrow \mathbf{Ordered}\ (n :: x)).$$

Obligation 3 of **Insert** :

$$\forall\ (n\ h : nat)\ (x\ t : list\ nat),$$
$$h < n \rightarrow x \sim (h :: t) \rightarrow (\mathbf{Ordered}\ t \rightarrow \mathbf{Ordered}\ x)$$
$$\rightarrow (h :: x) \sim (n :: h :: t) \wedge (\mathbf{Ordered}\ (h :: t) \rightarrow \mathbf{Ordered}\ (h :: x)).$$

Как это работает?



Как это работает?

Доказательства

Логические высказывания

Логические связки

Логические эквивалентности

Тавтологии

Правила вывода

Закон исключённого третьего

Логические высказывания

Мы говорим, что высказывание является логическим, если оно:

1. написано в принятом алфавите
2. соответствует правилам грамматики и морфологии языка
3. имеет единственный смысл
4. является истинным или ложным **в данном контексте.**

Примеры логических высказываний	
«Москва – столица России»	Истинно в 2016 году. Ложно в 1812 году.
«Сумма углов в треугольнике равна 180°»	Истинно в геометрии Евклида. Ложно в геометрии Лобачевского.
«Всякое четное число большее 2 можно представить как сумму двух простых чисел»	Неизвестно!

$\wedge \urcorner \rightarrow$ Логические связки

Таблицы истинности

x	y	$x \wedge y$	$x \vee y$	$x \to y$	$\neg x$	$\neg y$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$

Формулы

Из элементарных высказываний и логических связок можно собирать более сложные высказывания – формулы. Например:

$$\mathbb{A} = (x \vee z) \wedge (y \vee z), \qquad \mathbb{B} = (x \to z) \to (y \to z)$$

≋ Логические эквивалентности

Изучая таблицы истинности можно установить следующие эквивалентности между логическими формулами:

x	$x \to \mathbb{F}$	$\neg x$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$

1) $\neg x \approx x \to \mathbb{F}$
2) $\mathbb{T} \approx \neg \mathbb{F}$
3) $x \land y \approx \neg(x \to \neg y)$
4) $x \lor y \approx (\neg x) \to y$

x	y	$\neg y$	$x \to \neg y$	$\neg(x \to \neg y)$	$x \land y$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$

$\mathbb{T}$ Тавтологии

В математике особый класс высказываний составляют так называемые тавтологии или тождественно истинные высказывания. Все теоремы математики являются тавтологиями.

Легко проверить, что каждой эквивалентности вида: $\mathbb{A} \approx \mathbb{B}$ соответствует тавтология вида:

$$(\mathbb{A} \rightarrow \mathbb{B}) \wedge (\mathbb{B} \rightarrow \mathbb{A}) \approx \mathbb{T}$$

Стоящую с левой стороны формулу для краткости часто обозначают специальной логической связкой $(\mathbb{A} \Leftrightarrow \mathbb{B})$ (читается «если и только если»).

При помощи таблиц истинности можно также убедиться в справедливости следующих логических законов:

$\mathbb{A} \vee (\neg \mathbb{A}) \approx \mathbb{T}$ (Закон исключенного третьего)

$\mathbb{A} \rightarrow (\mathbb{B} \rightarrow \mathbb{A}) \approx \mathbb{T}$ (Истина следует из любого утверждения)

Правила вывода

Контекст ($\mathbb{\Gamma}$) – список гипотез.

Секвенция ($\mathbb{\Gamma} \vdash \mathbb{A}$) – утверждение о выводимости формулы $\mathbb{A}$ из гипотез $\mathbb{\Gamma}$.

Правило вывода: $\left(\dfrac{\Sigma_1, \ldots, \Sigma_n}{\Sigma}\right)$ – утверждение о выводимости секвенции Σ из выводимости секвенций $\Sigma_1, \ldots, \Sigma_n$.

Рассмотрим тавтологию:

$$\mathbb{A} \to (\mathbb{B} \to \mathbb{A})$$

Её формальный вывод:

$$\dfrac{\dfrac{\dfrac{\mathbb{A} \in \mathbb{A} :: \mathbb{B}}{\mathbb{A} :: \mathbb{B} \vdash \mathbb{A}}\ \text{Вывод из гипотез}}{\mathbb{A} \vdash \mathbb{B} \to \mathbb{A}}\ \text{Правило дедукции}}{\vdash \mathbb{A} \to (\mathbb{B} \to \mathbb{A})}\ \text{Правило дедукции}$$

Правила вывода

Вывод из гипотез:

$$\dfrac{\mathbb{A} \in \mathbb{\Gamma}}{\mathbb{\Gamma} \vdash \mathbb{A}}$$

Modus Ponens:

$$\dfrac{\mathbb{\Gamma} \vdash \mathbb{A}, \quad \mathbb{\Gamma} \vdash \mathbb{A} \to \mathbb{B}}{\mathbb{\Gamma} \vdash \mathbb{B}}$$

Правило дедукции:

$$\dfrac{\mathbb{\Gamma} :: \mathbb{A} \vdash \mathbb{B}}{\mathbb{\Gamma} \vdash \mathbb{A} \to \mathbb{B}}$$

Снятие двойного отрицания:

$$\dfrac{\mathbb{\Gamma} \vdash \neg\neg \mathbb{A}}{\mathbb{\Gamma} \vdash \mathbb{A}}$$

Закон исключённого третьего

Правило снятия двойного отрицания эквивалентно закону исключённого третьего, который широко используется в математике для проведения «доказательства от противного».

Многие математики отмечали, что для бесконечных множеств справедливость закона исключённого третьего не очевидна.

Направление в математике, которое стремится получить все результаты без использования закона исключенного третьего, называется интуиционизмом.

Правила вывода

Вывод из гипотез:

$$\frac{\mathbb{A} \in \Gamma}{\Gamma \vdash \mathbb{A}}$$

Modus Ponens:

$$\frac{\Gamma \vdash \mathbb{A}, \quad \Gamma \vdash \mathbb{A} \to \mathbb{B}}{\Gamma \vdash \mathbb{B}}$$

Правило дедукции:

$$\frac{\Gamma :: \mathbb{A} \vdash \mathbb{B}}{\Gamma \vdash \mathbb{A} \to \mathbb{B}}$$

Снятие двойного отрицания:

$$\frac{\Gamma \vdash \neg\neg \mathbb{A}}{\Gamma \vdash \mathbb{A}}$$

Вычисления

- Инструкция
- Программа
- Функция
- Лямбда-исчисление
- Типизация
- Типизированное лямбда-исчисление

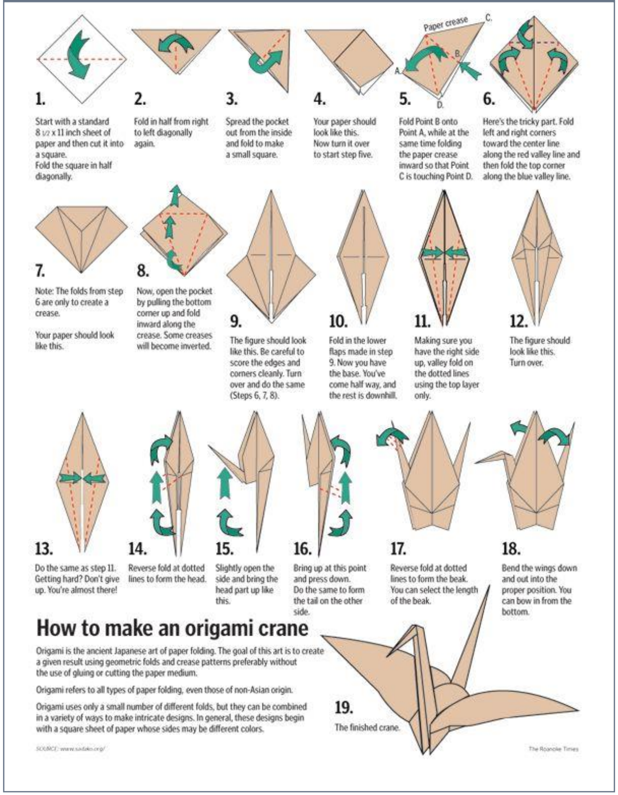
Инструкция

Инструкция – описание последовательного порядка действий исполнителя для достижения некоторого результата на основе заданных начальных данных.

Свойства инструкции

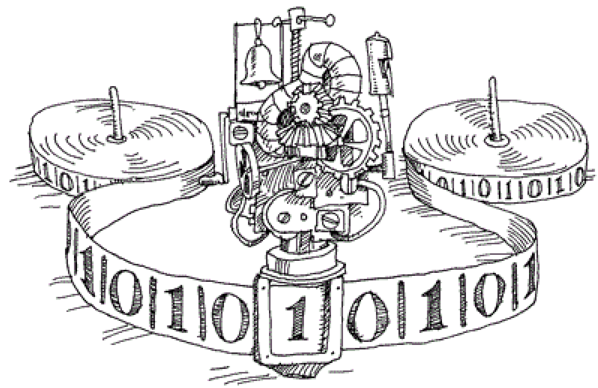
1. Вход (диапазон начальных данных)
2. Выход (ожидаемый результат)
3. Дискретность (элементарные шаги)
4. Линейный порядок (четкая последовательность выполнения шагов)
5. Конечность (окончание процедуры для всех допустимых начальных данных)

Как сложить журавлика?



Программа

Программа – набор инструкций, описывающих порядок действий исполнителя с возможностью проверки условий и передачи управления от одной инструкции к другой.



Свойства программы

1. **Вход** (диапазон начальных данных)
2. **Выход** (ожидаемый тип результата)
3. **Дискретность** (элементарные шаги вычислений)
4. **Детерминированность** (четкая определенность выполнения шагов)

Команды машины Поста

1. **Сдвинуть вправо:** $[\Rightarrow \quad k]$
2. **Сдвинуть влево:** $[\Leftarrow \quad k]$
3. **Напечатать метку:** [■ k]
4. **Стереть метку:** [⬚ k]
5. **Если-то-иначе:** $[? \quad n \quad m]$
6. **Остановка:** ⌘

Функция

Функция – это программа, которая завершает свои вычисления и выдает ожидаемый результат при любом значении начальных данных из разрешенного диапазона.

Свойства функции

1. **Вход** (диапазон начальных данных)
2. **Выход** (ожидаемый результат)
3. **Дискретность** (элементарные шаги)
4. **Детерминированность** (четкая определенность выполнения шагов)
5. **Конечность** (окончание процедуры для всех допустимых начальных данных)

Константы:

$0, 1, 2, 3, \ldots$

Базовые функции:

plus (_ , _)

mult (_ , _)

Переменные:

$x, y, z, \ldots$

Выражения:

$u(x, y) \equiv 3 * (x + y)^2 + x$

Лямбда-функции:

$\lambda x \cdot u[x, y] \equiv_\alpha \lambda z \cdot u[z, y]$

Применение функции:

$(\lambda x \cdot u[x, y])\ z * y$

$= 3 * (z * y + y)^2 + z * y$

λ Лямбда-исчисление

Формальное лямбда-исчисление оказалось чрезвычайно мощным инструментом программирования. Как доказал Алан Тьюринг, любая (не обязательно конечная!) программа на машине Тьюринга может быть реализована в терминах лямбда-исчисления и наоборот.

Правила построения выражений

$u \equiv$		выражение
$\mid$	x	переменные
$\mid$	$\lambda x \cdot u$	лямбда-функции
$\mid$	$u\, u$	применение

Правила вычислений

альфа-конверсия:

$$\lambda x \cdot u[x, y] \equiv_\alpha \lambda z \cdot u[z, y]$$

бета-редукция:

$$(\lambda x \cdot u)\, v \Rightarrow_\beta [v \leftarrow x]\, u$$

λ Лямбда-исчисление

Формальное лямбда-исчисление оказалось чрезвычайно мощным инструментом программирования. Как доказал Алан Тьюринг, любая (не обязательно конечная!) программа на машине Тьюринга может быть реализована в терминах лямбда-исчисления и наоборот.

Почему?

Причина в расширенном толковании операции применения одного выражения к другому!

Построение выражений

u	$\equiv$	выражение
	$\mid\ x$	переменные
	$\mid\ \lambda\, x \cdot u$	лямбда-функции
	$\mid\ u\ u$	применение

Правила вычислений

альфа-конверсия:

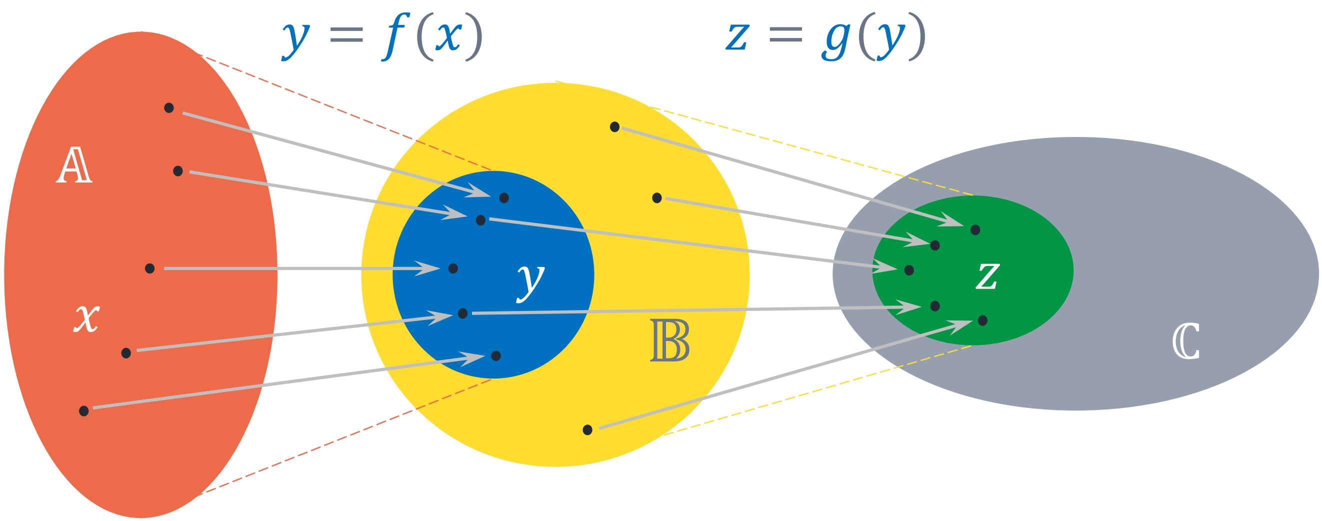
$$\lambda\, x \cdot u[x, y] \equiv_{\alpha} \lambda\, z \cdot u[z, y]$$

бета-редукция:

$$(\lambda\, x \cdot u)\ v \Rightarrow_{\beta} [v \leftarrow x]\ u$$

Типизация

Каждая функция имеет область определения и область значений. Расставляя соответствующие синтаксические метки (типы) мы можем определить к каким выражениям можно применять ту или иную функцию, а к каким – нет.



$x\colon \mathbb{A}$ $\quad f\colon \mathbb{A} \to \mathbb{B}$ $\quad y\colon \mathbb{B}$ $\quad g\colon \mathbb{B} \to \mathbb{C}$ $\quad z\colon \mathbb{C}$

Типизированное лямбда-исчисление

Для того, чтобы получить реалистичную теорию лямбда-функций, введем следующие правила типизации:

Типы

$\mathbb{T} \equiv$		тип
	$\mid \mathbb{A}$	основной
	$\mid \mathbb{T} \to \mathbb{T}$	функциональный

Построение выражений

$u \equiv$		выражение
	$\mid x : \mathbb{T}$	переменные
	$\mid \lambda\, x : \mathbb{T} \cdot u$	лямбда-функции
	$\mid u\ u$	применение

Правила типизации

Ссылка:

$$\frac{x : \mathbb{A} \in \Gamma}{\Gamma \vdash x : \mathbb{A}}$$

Абстракция:

$$\frac{\Gamma :: (x : \mathbb{A}) \vdash u : \mathbb{B}}{\Gamma \vdash (\lambda\, x : \mathbb{A} \cdot u) : \mathbb{A} \to \mathbb{B}}$$

Применение:

$$\frac{\Gamma \vdash x : \mathbb{A}, \quad \Gamma \vdash f : \mathbb{A} \to \mathbb{B}}{\Gamma \vdash f\ x : \mathbb{B}}$$

Изоморфизм Карри-Ховарда

Вычисления

1 Типы

Базовые: $A, B, C, \ldots$

Функциональные: $A \to B, \ldots$

2 Программы

Переменные: $x,\ y,\ z, \ldots$

Применение: $f\,x, \ldots$

Абстракция: $\lambda\, x \bullet y, \ldots$

3 Правила типизации

Применение:

$$\frac{\Gamma \vdash f : A \to B, \quad \Gamma \vdash x : A}{\Gamma \vdash f\,x : B}$$

Абстракция:

$$\frac{(x : A) :: \Gamma \vdash y : B}{\Gamma \vdash \lambda\, x \bullet y : A \to B}$$

Доказательства

1 Утверждения

Элементарные: $P, Q, R, \ldots$

Условные: $P \to Q, \ldots$

2 Доказательства

Прямые свидетельства: $u,\ v,\ w, \ldots$

Modus Ponens: $u\,v, \ldots$

Дедукция: $\lambda\, u \bullet v, \ldots$

3 Правила вывода

Modus Ponens:

$$\frac{\Gamma \vdash t : P \to Q, \quad \Gamma \vdash u : P}{\Gamma \vdash t\,u : Q}$$

Теорема дедукции:

$$\frac{(u : P) :: \Gamma \vdash v : Q}{\Gamma \vdash \lambda\, u \bullet v : P \Rightarrow Q}$$

Coq

Proof Assistant

Спасибо!

Загрузить Coq можно по ссылке:

https://coq.inria.fr/download

Вопросы можно направлять по адресу:

Evgeny.Ivashkevich@gmail.com