# Семинар «Глобус»

## Computer Science: попытка апологии

LIRMM CNRS, Poncelet lab, ИППИ РАН
Supported in part by NAFIT ANR project

# Общие замечания

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)
- Наш предмет: computer science как часть математики (изучаются абстрактные объекты)

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)
- Наш предмет: computer science как часть математики (изучаются абстрактные объекты)
- В чём интерес:

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)
- Наш предмет: computer science как часть математики (изучаются абстрактные объекты)
- В чём интерес:
  - философский

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)
- Наш предмет: computer science как часть математики (изучаются абстрактные объекты)
- В чём интерес:
  - философский
  - математический

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)
- Наш предмет: computer science как часть математики (изучаются абстрактные объекты)
- В чём интерес:
  - философский
  - математический
  - практический

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)
- Наш предмет: computer science как часть математики (изучаются абстрактные объекты)
- В чём интерес:
  - философский
  - математический
  - практический
- Отступление: схема «математики доказали — инженеры применили» и реальность

# Общие замечания

- Нелепое название. От физики (транзисторы) до социологии (клиенты соцсетей)
- Наш предмет: computer science как часть математики (изучаются абстрактные объекты)
- В чём интерес:
  - философский
  - математический
  - практический
- Отступление: схема «математики доказали — инженеры применили» и реальность
- Математическая логика и теория алгоритмов $\subseteq$ CS $\subset$ математика

# Истинность и доказуемость

# Истинность и доказуемость

# Истинность и доказуемость


- несовпадение (Гёдель)

# Истинность и доказуемость


- несовпадение (Гёдель)
- неалгоритмичность и неарифметичность границы между истиной и ложью (Тарский)

# Истинность и доказуемость


- несовпадение (Гёдель)
- неалгоритмичность и неарифметичность границы между истиной и ложью (Тарский)
- неотделимость: нет алгоритмического правила, позволяющего отделить доказуемые от опровержимых, даже есть не требовать совпадения с истинностью

# Вычислимость, разрешимость, перечислимость

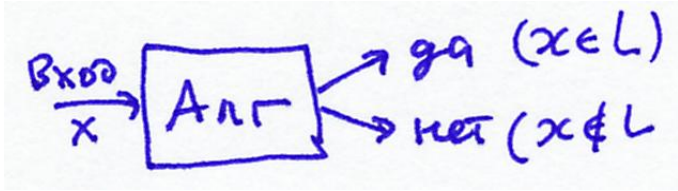
# Вычислимость, разрешимость, перечислимость

- 

вычислимость

# Вычислимость, разрешимость, перечислимость

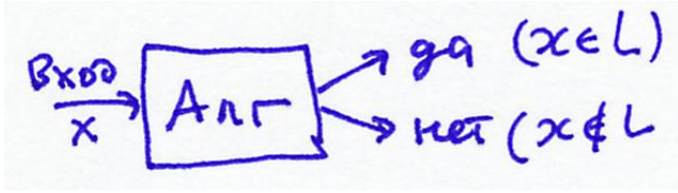
- 

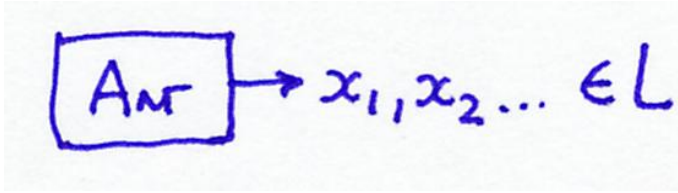
вычислимость
- 

разрешимость

# Вычислимость, разрешимость, перечислимость

- 

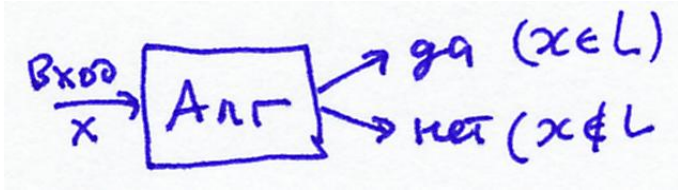
вычислимость
- 

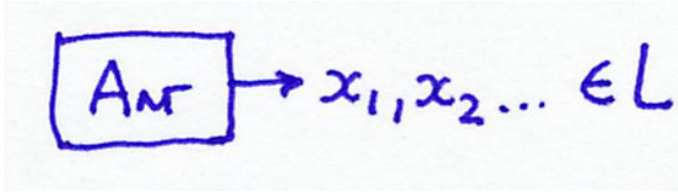
разрешимость
- 

перечислимость

# Вычислимость, разрешимость, перечислимость

- 

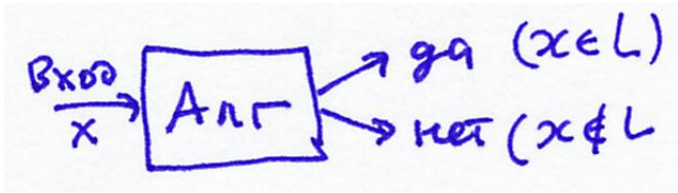
вычислимость
- 

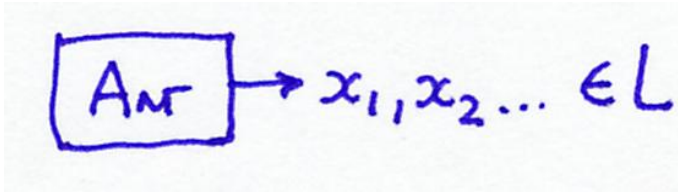
разрешимость
- 

перечислимость
- Примеры:

# Вычислимость, разрешимость, перечислимость

- 

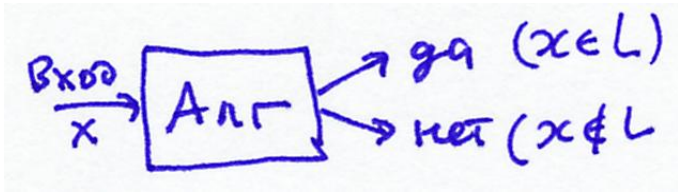
вычислимость
- 

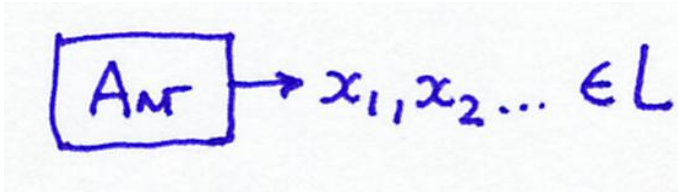
разрешимость
- 

перечислимость
- Примеры:
  - разрешимое множество перечислимо

# Вычислимость, разрешимость, перечислимость

- 

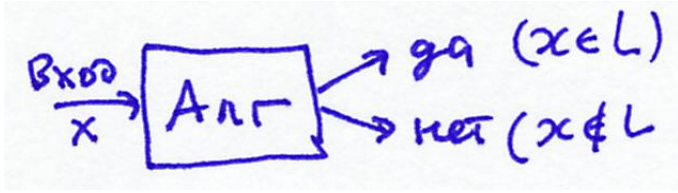
вычислимость
- 

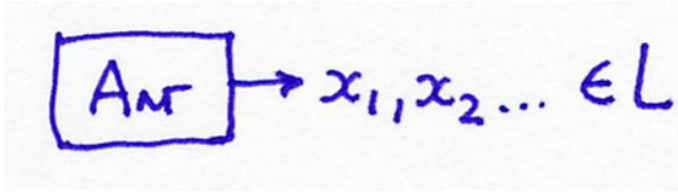
разрешимость
- 

перечислимость
- Примеры:
  - разрешимое множество перечислимо
  - если множество и дополнение перечислимы, то оно разрешимо

# Вычислимость, разрешимость, перечислимость

- 

вычислимость
- 

разрешимость
- 

перечислимость
- Примеры:
  - разрешимое множество перечислимо
  - если множество и дополнение перечислимы, то оно разрешимо
  - область определения вычислимой функции перечислима

# Структура на счётных множествах

# Структура на счётных множествах

- $X$ — счётное множество (натуральные числа $\mathbb{N}$, двоичные слова $\mathbb{B}^*$, графы, разбиения, рациональные числа,...)

# Структура на счётных множествах

- $X$ — счётное множество (натуральные числа $\mathbb{N}$, двоичные слова $\mathbb{B}^*$, графы, разбиения, рациональные числа,...)
- $A \subset X$: конечно, бесконечно и имеет бесконечное дополнение, имеет конечное дополнение

# Структура на счётных множествах

- $X$ — счётное множество (натуральные числа $\mathbb{N}$, двоичные слова $\mathbb{B}^*$, графы, разбиения, рациональные числа,...)
- $A \subset X$: конечно, бесконечно и имеет бесконечное дополнение, имеет конечное дополнение
- другие свойства, устойчивые относительно «естественных» биекций между этими множествами?

# Структура на счётных множествах

- $X$ — счётное множество (натуральные числа $\mathbb{N}$, двоичные слова $\mathbb{B}^*$, графы, разбиения, рациональные числа,...)
- $A \subset X$: конечно, бесконечно и имеет бесконечное дополнение, имеет конечное дополнение
- другие свойства, устойчивые относительно «естественных» биекций между этими множествами?
- разрешимость, перечислимость,...

# Структура на счётных множествах

- $X$ — счётное множество (натуральные числа $\mathbb{N}$, двоичные слова $\mathbb{B}^*$, графы, разбиения, рациональные числа,...)
- $A \subset X$: конечно, бесконечно и имеет бесконечное дополнение, имеет конечное дополнение
- другие свойства, устойчивые относительно «естественных» биекций между этими множествами?
- разрешимость, перечислимость,...
- вычислимое число: есть алгоритм, по заданной точности указывающий приближение

# Структура на счётных множествах

- $X$ — счётное множество (натуральные числа $\mathbb{N}$, двоичные слова $\mathbb{B}^*$, графы, разбиения, рациональные числа,...)
- $A \subset X$: конечно, бесконечно и имеет бесконечное дополнение, имеет конечное дополнение
- другие свойства, устойчивые относительно «естественных» биекций между этими множествами?
- разрешимость, перечислимость,...
- вычислимое число: есть алгоритм, по заданной точности указывающий приближение
- алгебраически замкнутое счётное поле, содержащее $e$, $\pi$ и почти все «мыслимые» числа

# Перечислимость и «доказательства»

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»
- система корректна для множества слов $L$:

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»
- система корректна для множества слов $L$:
  - $s \in L$: существует $p$, которое является доказательством для $s$;

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»
- система корректна для множества слов $L$:
  - $s \in L$: существует $p$, которое является доказательством для $s$;
  - $s \notin L$: никакое $p$ не является доказательством для $s$

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»
- система корректна для множества слов $L$:
  - $s \in L$: существует $p$, которое является доказательством для $s$;
  - $s \notin L$: никакое $p$ не является доказательством для $s$
- $L$ есть проекция множества $\{(p, s) \mid A(p, s)\}$

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»
- система корректна для множества слов $L$:
  - $s \in L$: существует $p$, которое является доказательством для $s$;
  - $s \notin L$: никакое $p$ не является доказательством для $s$
- $L$ есть проекция множества $\{(p, s) \mid A(p, s)\}$
- Всеведущий Мерлин хочет убедить недоверчивого Артура, что $s \in L$; система проверки представляемых Мерлином доказательств — алгоритм $A$

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»
- система корректна для множества слов $L$:
  - $s \in L$: существует $p$, которое является доказательством для $s$;
  - $s \notin L$: никакое $p$ не является доказательством для $s$
- $L$ есть проекция множества $\{(p, s) \mid A(p, s)\}$
- Всеведущий Мерлин хочет убедить недоверчивого Артура, что $s \in L$; система проверки представляемых Мерлином доказательств — алгоритм $A$
- такое бывает для перечислимых множеств

# Перечислимость и «доказательства»

- система проверки доказательств: алгоритм $A$, по словам $p$ и $s$ говорящий «да» или «нет» (всегда); «$p$ есть доказательство для $s$»
- система корректна для множества слов $L$:
  - $s \in L$: существует $p$, которое является доказательством для $s$;
  - $s \notin L$: никакое $p$ не является доказательством для $s$
- $L$ есть проекция множества $\{(p, s) \mid A(p, s)\}$
- Всеведущий Мерлин хочет убедить недоверчивого Артура, что $s \in L$; система проверки представляемых Мерлином доказательств — алгоритм $A$
- такое бывает для перечислимых множеств
- логические теории — частные случаи

# Два наблюдения

# Два наблюдения

- «машинно-независимость»

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же
  - разные процессоры могут выполнять одни и те же программы

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же
  - разные процессоры могут выполнять одни и те же программы
- существует универсальная вычислимая функция $U(x, y)$

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же
  - разные процессоры могут выполнять одни и те же программы
- существует универсальная вычислимая функция $U(x, y)$
  - $U_x \colon y \mapsto U(x, y)$

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же
  - разные процессоры могут выполнять одни и те же программы
- существует универсальная вычислимая функция $U(x, y)$
  - $U_x\colon\ y \mapsto U(x, y)$
  - [все $U_x$ вычислимы]

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же
  - разные процессоры могут выполнять одни и те же программы
- существует универсальная вычислимая функция $U(x, y)$
  - $U_x\colon\ y \mapsto U(x, y)$
  - [все $U_x$ вычислимы]
  - $U$ универсальна, если среди $U_x$ есть все вычислимые функции

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же
  - разные процессоры могут выполнять одни и те же программы
- существует универсальная вычислимая функция $U(x, y)$
  - $U_x\colon\ y \mapsto U(x, y)$
  - [все $U_x$ вычислимы]
  - $U$ универсальна, если среди $U_x$ есть все вычислимые функции
  - $U$ — интерпретатор универсального языка программирования

# Два наблюдения

- «машинно-независимость»
  - разные вычислительные модели приводят к одному и тому же классу вычислимых функций
  - разные языки программирования позволяют запрограммировать одно и то же
  - разные процессоры могут выполнять одни и те же программы
- существует универсальная вычислимая функция $U(x, y)$
  - $U_x\colon\ y \mapsto U(x, y)$
  - [все $U_x$ вычислимы]
  - $U$ универсальна, если среди $U_x$ есть все вычислимые функции
  - $U$ — интерпретатор универсального языка программирования
  - хранимая программа: один и тот же процессор для всего

# Неразрешимые алгоритмические проблемы

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены
- **if** $U(x, x)$ определено **then** $U(x, x) + 1$ **else** $0$ **fi** ?

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены
- **if** $U(x, x)$ определено **then** $U(x, x) + 1$ **else** $0$ **fi** ?
- отличается от всех вычислимых: неразрешимость проблемы остановки

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены
- **if** $U(x, x)$ определено **then** $U(x, x) + 1$ **else** $0$ **fi** ?
- отличается от всех вычислимых: неразрешимость проблемы остановки
- конкретная неразрешимая проблема (Конвей, FRACTRAN):

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены
- **if** $U(x, x)$ определено **then** $U(x, x) + 1$ **else** $0$ **fi** ?
- отличается от всех вычислимых: неразрешимость проблемы остановки
- конкретная неразрешимая проблема (Конвей, FRACTRAN):
  - есть набор рациональных положительных $r_1, \ldots, r_k$ и целое положительное $a$

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены
- **if** $U(x, x)$ определено **then** $U(x, x) + 1$ **else** $0$ **fi** ?
- отличается от всех вычислимых: неразрешимость проблемы остановки
- конкретная неразрешимая проблема (Конвей, FRACTRAN):
  - есть набор рациональных положительных $r_1, \ldots, r_k$ и целое положительное $a$
  - на каждом шаге, если $ar_i$ целое при некотором $i$, берём минимальное $i$ и заменяем $a$ на $ar_i$

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены
- **if** $U(x, x)$ определено **then** $U(x, x) + 1$ **else** $0$ **fi** ?
- отличается от всех вычислимых: неразрешимость проблемы остановки
- конкретная неразрешимая проблема (Конвей, FRACTRAN):
  - есть набор рациональных положительных $r_1, \ldots, r_k$ и целое положительное $a$
  - на каждом шаге, если $ar_i$ целое при некотором $i$, берём минимальное $i$ и заменяем $a$ на $ar_i$
  - останавливаемся, если все $ar_i$ нецелые

# Неразрешимые алгоритмические проблемы

- парадокс: $x \mapsto U(x, x) + 1$ вычислимо, но отличается от всех вычислимых функций?
- нет: $U$ не всюду определена, и $U(x, x) + 1$ и $U_x(x)$ могут быть не определены
- **if** $U(x, x)$ определено **then** $U(x, x) + 1$ **else** $0$ **fi** ?
- отличается от всех вычислимых: неразрешимость проблемы остановки
- конкретная неразрешимая проблема (Конвей, FRACTRAN):
  - есть набор рациональных положительных $r_1, \ldots, r_k$ и целое положительное $a$
  - на каждом шаге, если $ar_i$ целое при некотором $i$, берём минимальное $i$ и заменяем $a$ на $ar_i$
  - останавливаемся, если все $ar_i$ нецелые
  - задача: по данным $r_i$ и $a$ определить, случится ли остановка

# Классификация подмножеств $\mathbb{N}$

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок
- и не все подмножества $\mathbb{N}$ «одинаковы»

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок
- и не все подмножества $\mathbb{N}$ «одинаковы»
- классификация по числу кванторов

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок
- и не все подмножества $\mathbb{N}$ «одинаковы»
- классификация по числу кванторов
  - останавливающие программы без входа: $\exists$ «существует $t$, для которого программа останавливается за $t$ шагов»

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок
- и не все подмножества $\mathbb{N}$ «одинаковы»
- классификация по числу кванторов
  - останавливающие программы без входа: $\exists$ «существует $t$, для которого программа останавливается за $t$ шагов»
  - останавливающиеся на любом входе программы: $\forall\exists$: «для любого входа есть число шагов, за которое программа на нём останавливается»

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок
- и не все подмножества $\mathbb{N}$ «одинаковы»
- классификация по числу кванторов
  - останавливающие программы без входа: $\exists$ «существует $t$, для которого программа останавливается за $t$ шагов»
  - останавливающиеся на любом входе программы: $\forall\exists$: «для любого входа есть число шагов, за которое программа на нём останавливается»
  - и так далее

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок
- и не все подмножества $\mathbb{N}$ «одинаковы»
- классификация по числу кванторов
  - останавливающие программы без входа: $\exists$ «существует $t$, для которого программа останавливается за $t$ шагов»
  - останавливающиеся на любом входе программы: $\forall\exists$: «для любого входа есть число шагов, за которое программа на нём останавливается»
  - и так далее
  - все классы $\exists$, $\forall$, $\exists\forall$, $\forall\exists$, $\exists\forall\exists$,... разные

# Классификация подмножеств $\mathbb{N}$

- В $\mathbb{N}$ есть структура, инвариантная относительно всех естественных перестановок
- и не все подмножества $\mathbb{N}$ «одинаковы»
- классификация по числу кванторов
  - останавливающие программы без входа: $\exists$ «существует $t$, для которого программа останавливается за $t$ шагов»
  - останавливающиеся на любом входе программы: $\forall\exists$: «для любого входа есть число шагов, за которое программа на нём останавливается»
  - и так далее
  - все классы $\exists$, $\forall$, $\exists\forall$, $\forall\exists$, $\exists\forall\exists$,... разные
- классификация по сложности: $A(x)$ проще $B(x)$, если есть алгоритм, вычисляющий $A(x)$ и вызывающий $B(x)$ как внешнюю процедуру

# Структура на конечных множествах

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности
- сложность $B(x_1, \ldots, x_n)$: минимальное количество бинарных операций, нужное для вычисления функции $B$

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности
- сложность $B(x_1, \ldots, x_n)$: минимальное количество бинарных операций, нужное для вычисления функции $B$
- модель: программа из присваиваний; булева схема

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности
- сложность $B(x_1, \ldots, x_n)$: минимальное количество бинарных операций, нужное для вычисления функции $B$
- модель: программа из присваиваний; булева схема
- техническое: не то же самое, что размер формулы (повторное использование результатов)

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности
- сложность $B(x_1, \ldots, x_n)$: минимальное количество бинарных операций, нужное для вычисления функции $B$
- модель: программа из присваиваний; булева схема
- техническое: не то же самое, что размер формулы (повторное использование результатов)
- «13-ая проблема Гильберта» в конечном варианте

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности
- сложность $B(x_1, \ldots, x_n)$: минимальное количество бинарных операций, нужное для вычисления функции $B$
- модель: программа из присваиваний; булева схема
- техническое: не то же самое, что размер формулы (повторное использование результатов)
- «13-ая проблема Гильберта» в конечном варианте
- большинство функций имеют сложность порядка $2^n$

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности
- сложность $B(x_1, \ldots, x_n)$: минимальное количество бинарных операций, нужное для вычисления функции $B$
- модель: программа из присваиваний; булева схема
- техническое: не то же самое, что размер формулы (повторное использование результатов)
- «13-ая проблема Гильберта» в конечном варианте
- большинство функций имеют сложность порядка $2^n$
- но доказанные нижние оценки для конкретных функций только $5n$ (!)

# Структура на конечных множествах

- функции $A\colon \mathbb{B}^n \to \mathbb{B}$ = подмножества $\mathbb{B}^n$
- можно ли их как-то классифицировать по сложности
- сложность $B(x_1, \ldots, x_n)$: минимальное количество бинарных операций, нужное для вычисления функции $B$
- модель: программа из присваиваний; булева схема
- техническое: не то же самое, что размер формулы (повторное использование результатов)
- «13-ая проблема Гильберта» в конечном варианте
- большинство функций имеют сложность порядка $2^n$
- но доказанные нижние оценки для конкретных функций только $5n$ (!)
- а для большего множества вместо $\mathbb{B}$ вообще константа

# класс P

# класс P

- алгоритмы, работающие полиномиальное время от длины входа

# класс P

- алгоритмы, работающие полиномиальное время от длины входа
- предикат $L \subset \mathbb{B}^*$ = функция $\mathbb{B}^* \to \mathbb{B}$ («язык», «задача»)

# класс P

- алгоритмы, работающие полиномиальное время от длины входа
- предикат $L \subset \mathbb{B}^*$ = функция $\mathbb{B}^* \to \mathbb{B}$ («язык», «задача»)
- аккуратный выбор модели: «небольшое число битов обрабатывается за один шаг»

# класс P

- алгоритмы, работающие полиномиальное время от длины входа
- предикат $L \subset \mathbb{B}^*$ = функция $\mathbb{B}^* \to \mathbb{B}$ («язык», «задача»)
- аккуратный выбор модели: «небольшое число битов обрабатывается за один шаг»
- (скажем, сложение и умножение целых чисел произвольного размера не годится)

# класс P

- алгоритмы, работающие полиномиальное время от длины входа
- предикат $L \subset \mathbb{B}^*$ = функция $\mathbb{B}^* \to \mathbb{B}$ («язык», «задача»)
- аккуратный выбор модели: «небольшое число битов обрабатывается за один шаг»
- (скажем, сложение и умножение целых чисел произвольного размера не годится)
- первое приближение: задачи с полиномиальными алгоритмами = практически разрешимые

# класс P

- алгоритмы, работающие полиномиальное время от длины входа
- предикат $L \subset \mathbb{B}^*$ = функция $\mathbb{B}^* \to \mathbb{B}$ («язык», «задача»)
- аккуратный выбор модели: «небольшое число битов обрабатывается за один шаг»
- (скажем, сложение и умножение целых чисел произвольного размера не годится)
- первое приближение: задачи с полиномиальными алгоритмами = практически разрешимые
- линейное программирование (проверка совместности неравенств с целыми коэффициентами), паросочетание данного размера, эйлеров цикл

# класс P

- алгоритмы, работающие полиномиальное время от длины входа
- предикат $L \subset \mathbb{B}^*$ = функция $\mathbb{B}^* \to \mathbb{B}$ («язык», «задача»)
- аккуратный выбор модели: «небольшое число битов обрабатывается за один шаг»
- (скажем, сложение и умножение целых чисел произвольного размера не годится)
- первое приближение: задачи с полиномиальными алгоритмами = практически разрешимые
- линейное программирование (проверка совместности неравенств с целыми коэффициентами), паросочетание данного размера, эйлеров цикл
- предикат $L$ = семейство булевых функций $L_n \colon \mathbb{B}^n \to \mathbb{B}$

# класс P

- алгоритмы, работающие полиномиальное время от длины входа
- предикат $L \subset \mathbb{B}^*$ = функция $\mathbb{B}^* \to \mathbb{B}$ («язык», «задача»)
- аккуратный выбор модели: «небольшое число битов обрабатывается за один шаг»
- (скажем, сложение и умножение целых чисел произвольного размера не годится)
- первое приближение: задачи с полиномиальными алгоритмами = практически разрешимые
- линейное программирование (проверка совместности неравенств с целыми коэффициентами), паросочетание данного размера, эйлеров цикл
- предикат $L$ = семейство булевых функций $L_n \colon \mathbb{B}^n \to \mathbb{B}$
- $L \in P \Leftrightarrow$ есть последовательность схем для $L_n$ полиномиального размера, вычислимая за полиномиальное время

# класс NP

# класс NP

- вопросы о существовании объекта с каким-то свойством

# класс NP

- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$

# класс NP

- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$
  $p$ — многочлен, $R$ — свойство пары, проверяемое за полиномиальное время от $|x| + |y|)$

# класс NP

- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$
  $p$ — многочлен, $R$ — свойство пары, проверяемое за полиномиальное время от $|x| + |y|)$
- Мерлин по-прежнему всемогущий, Артур — полиномиально ограниченный и недоверчивый

# класс NP

- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$
  $p$ — многочлен, $R$ — свойство пары, проверяемое за полиномиальное время от $|x| + |y|)$
- Мерлин по-прежнему всемогущий, Артур — полиномиально ограниченный и недоверчивый
- $P \subset NP$

# класс NP

- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$
  $p$ — многочлен, $R$ — свойство пары, проверяемое за полиномиальное время от $|x| + |y|)$
- Мерлин по-прежнему всемогущий, Артур — полиномиально ограниченный и недоверчивый
- $P \subset NP$
- про некоторые задачи из NP удаётся доказать, что в P

# класс NP

- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$
  $p$ — многочлен, $R$ — свойство пары, проверяемое за полиномиальное время от $|x| + |y|)$
- Мерлин по-прежнему всемогущий, Артур — полиномиально ограниченный и недоверчивый
- $P \subset NP$
- про некоторые задачи из NP удаётся доказать, что в P
- а про некоторые нет: есть ли 0-1-решения у системы линейных неравенств с целыми коэффициентами; клика данного размера; гамильтонов цикл

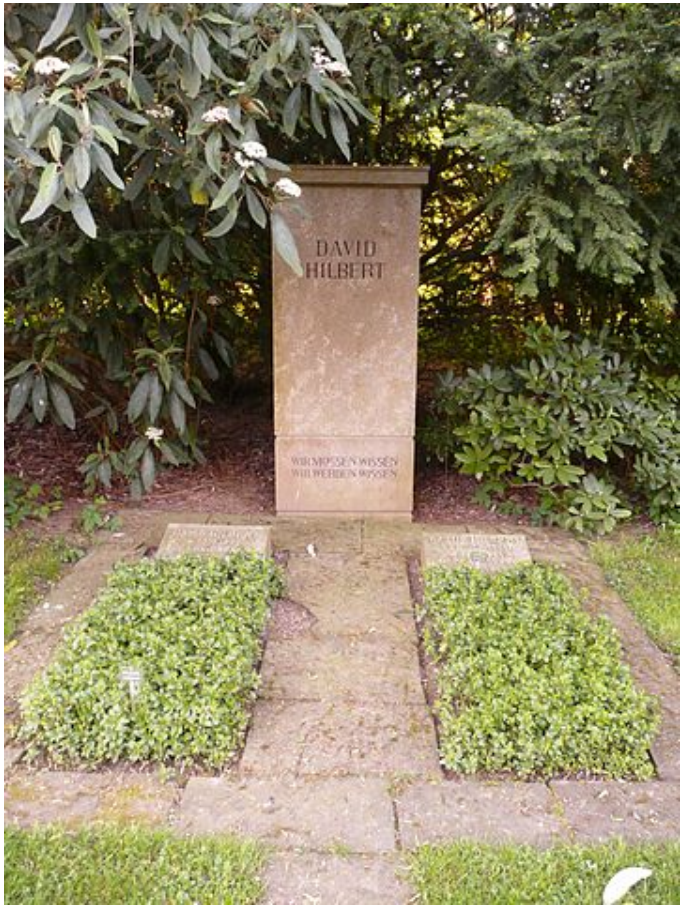
# класс NP

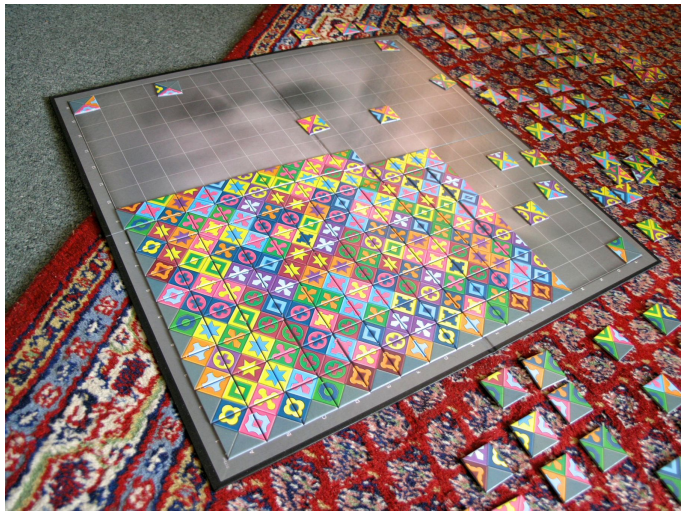
- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$
  $p$ — многочлен, $R$ — свойство пары, проверяемое за полиномиальное время от $|x| + |y|)$
- Мерлин по-прежнему всемогущий, Артур — полиномиально ограниченный и недоверчивый
- $P \subset NP$
- про некоторые задачи из NP удаётся доказать, что в P
- а про некоторые нет: есть ли 0-1-решения у системы линейных неравенств с целыми коэффициентами; клика данного размера; гамильтонов цикл
- плохо: никаких идей, как доказать, что не в P

# класс NP

- вопросы о существовании объекта с каким-то свойством
- $A(x) \Leftrightarrow \exists y[|y| \leq p(x) \wedge R(x, y)]$
  $p$ — многочлен, $R$ — свойство пары, проверяемое за полиномиальное время от $|x| + |y|)$
- Мерлин по-прежнему всемогущий, Артур — полиномиально ограниченный и недоверчивый
- $P \subset NP$
- про некоторые задачи из NP удаётся доказать, что в P
- а про некоторые нет: есть ли 0-1-решения у системы линейных неравенств с целыми коэффициентами; клика данного размера; гамильтонов цикл
- плохо: никаких идей, как доказать, что не в P
- хорошо: про многие доказано, что если в P, то $P = NP$.

# отступление о познаваемости мира



“Wir müssen wissen, wir werden wissen” – Really?

# Анализ сложности игр

# Анализ сложности игр

- игра полиномиального размера:

# Анализ сложности игр

- игра полиномиального размера:
  - входное слово $x$ известно обоим

# Анализ сложности игр

- игра полиномиального размера:
  - входное слово $x$ известно обоим
  - игроки ходят по очереди, записывая свои ходы словами полиномиальной от $|x|$ длины

# Анализ сложности игр

- игра полиномиального размера:
  - входное слово $x$ известно обоим
  - игроки ходят по очереди, записывая свои ходы словами полиномиальной от $|x|$ длины
  - игра с полной информацией

# Анализ сложности игр

- игра полиномиального размера:
  - входное слово $x$ известно обоим
  - игроки ходят по очереди, записывая свои ходы словами полиномиальной от $|x|$ длины
  - игра с полной информацией
  - длина игры ограничена полиномом от $|x|$

# Анализ сложности игр

- игра полиномиального размера:
  - входное слово $x$ известно обоим
  - игроки ходят по очереди, записывая свои ходы словами полиномиальной от $|x|$ длины
  - игра с полной информацией
  - длина игры ограничена полиномом от $|x|$
  - в конце приходит полиномиально ограниченный судья и определяет победителя

# Анализ сложности игр

- игра полиномиального размера:
  - входное слово $x$ известно обоим
  - игроки ходят по очереди, записывая свои ходы словами полиномиальной от $|x|$ длины
  - игра с полной информацией
  - длина игры ограничена полиномом от $|x|$
  - в конце приходит полиномиально ограниченный судья и определяет победителя
- теорема: можно определить победителя, имея полиномиальное от $|x|$ количество памяти, и так получаются все множества, разрешимые с полиномиальной памятью

# Анализ сложности игр

- игра полиномиального размера:
  - входное слово $x$ известно обоим
  - игроки ходят по очереди, записывая свои ходы словами полиномиальной от $|x|$ длины
  - игра с полной информацией
  - длина игры ограничена полиномом от $|x|$
  - в конце приходит полиномиально ограниченный судья и определяет победителя
- теорема: можно определить победителя, имея полиномиальное от $|x|$ количество памяти, и так получаются все множества, разрешимые с полиномиальной памятью
- ещё: игры на полиномиальной доске (без ограничения длины) — вычисления за экспоненциальное (от полинома) время

# Вероятностные алгоритмы

# Вероятностные алгоритмы

- Машина с датчиком случайных битов

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):
  - хотим проверить некоторое $n$

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):
  - хотим проверить некоторое $n$
  - берём случайно $a$ по модулю $n$

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):
  - хотим проверить некоторое $n$
  - берём случайно $a$ по модулю $n$
  - проверяем $a^{n-1} = 1 \pmod{n}$

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):
  - хотим проверить некоторое $n$
  - берём случайно $a$ по модулю $n$
  - проверяем $a^{n-1} = 1 \pmod{n}$
  - почти достаточно, если бы не числа Кармайкла

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):
  - хотим проверить некоторое $n$
  - берём случайно $a$ по модулю $n$
  - проверяем $a^{n-1} = 1 \pmod{n}$
  - почти достаточно, если бы не числа Кармайкла
  - $n - 1 = 2^s m$, где $m$ нечётно

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):
  - хотим проверить некоторое $n$
  - берём случайно $a$ по модулю $n$
  - проверяем $a^{n-1} = 1 \pmod{n}$
  - почти достаточно, если бы не числа Кармайкла
  - $n - 1 = 2^s m$, где $m$ нечётно
  - сначала $a^m$, а потом возводим в квадрат

# Вероятностные алгоритмы

- Машина с датчиком случайных битов
- корректность на большинстве входов или на любом входе с большой вероятностью?
- проверка полиномиальных тождеств над конечным полем
- проверка простоты числа (за полиномиальное от его битовой длины время):
  - хотим проверить некоторое $n$
  - берём случайно $a$ по модулю $n$
  - проверяем $a^{n-1} = 1 \pmod{n}$
  - почти достаточно, если бы не числа Кармайкла
  - $n - 1 = 2^s m$, где $m$ нечётно
  - сначала $a^m$, а потом возводим в квадрат
  - признак непростоты: $x^2 = 1$ при $\neq \pm 1$

# Интерактивные доказательства

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- + интерактивность («не книга, а лекция»)

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- + интерактивность («не книга, а лекция»)
- Всеведущий Мерлин убеждает полиномиального недоверчивого вероятностного Артура, отвечая на вопросы

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- + интерактивность («не книга, а лекция»)
- Всеведущий Мерлин убеждает полиномиального недоверчивого вероятностного Артура, отвечая на вопросы
- У Артура есть вероятностный алгоритм $V$, какие вопросы задавать и как реагировать на ответы (и когда считать доказанным)

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- + интерактивность («не книга, а лекция»)
- Всеведущий Мерлин убеждает полиномиального недоверчивого вероятностного Артура, отвечая на вопросы
- У Артура есть вероятностный алгоритм $V$, какие вопросы задавать и как реагировать на ответы (и когда считать доказанным)
- полнота и корректность:

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- \+ интерактивность («не книга, а лекция»)
- Всеведущий Мерлин убеждает полиномиального недоверчивого вероятностного Артура, отвечая на вопросы
- У Артура есть вероятностный алгоритм $V$, какие вопросы задавать и как реагировать на ответы (и когда считать доказанным)
- полнота и корректность:
  - если $x \in L$, то Мерлин может убедить Артура (с большой вероятностью)

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- + интерактивность («не книга, а лекция»)
- Всеведущий Мерлин убеждает полиномиального недоверчивого вероятностного Артура, отвечая на вопросы
- У Артура есть вероятностный алгоритм $V$, какие вопросы задавать и как реагировать на ответы (и когда считать доказанным)
- полнота и корректность:
  - если $x \in L$, то Мерлин может убедить Артура (с большой вероятностью)
  - если $x \notin L$, то при любой стратегии Мерлина вероятность убедить Артура мала

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- + интерактивность («не книга, а лекция»)
- Всеведущий Мерлин убеждает полиномиального недоверчивого вероятностного Артура, отвечая на вопросы
- У Артура есть вероятностный алгоритм $V$, какие вопросы задавать и как реагировать на ответы (и когда считать доказанным)
- полнота и корректность:
  - если $x \in L$, то Мерлин может убедить Артура (с большой вероятностью)
  - если $x \notin L$, то при любой стратегии Мерлина вероятность убедить Артура мала
- протокол для неизоморфизма графов: узнать, какой взят граф, если вершины перенумерованы случайно

# Интерактивные доказательства

- случайность в алгоритмах – а в доказательствах?
- + интерактивность («не книга, а лекция»)
- Всеведущий Мерлин убеждает полиномиального недоверчивого вероятностного Артура, отвечая на вопросы
- У Артура есть вероятностный алгоритм $V$, какие вопросы задавать и как реагировать на ответы (и когда считать доказанным)
- полнота и корректность:
  - если $x \in L$, то Мерлин может убедить Артура (с большой вероятностью)
  - если $x \notin L$, то при любой стратегии Мерлина вероятность убедить Артура мала
- протокол для неизоморфизма графов: узнать, какой взят граф, если вершины перенумерованы случайно
- теорема: такой алгоритм проверки существует в точности для языков, распознаваемых за полиномиальное время

# Локально проверяемые доказательства

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)
- Артур недоверчивый, полиномиально ограниченный, вероятностный

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)
- Артур недоверчивый, полиномиально ограниченный, вероятностный
- и может прочитать лишь $O(1)$ битов доказательства

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)
- Артур недоверчивый, полиномиально ограниченный, вероятностный
- и может прочитать лишь $O(1)$ битов доказательства
- полнота и корректность:

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)
- Артур недоверчивый, полиномиально ограниченный, вероятностный
- и может прочитать лишь $O(1)$ битов доказательства
- полнота и корректность:
  - если $x \in L$, то есть гарантированно убеждающее доказательство

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)
- Артур недоверчивый, полиномиально ограниченный, вероятностный
- и может прочитать лишь $O(1)$ битов доказательства
- полнота и корректность:
  - если $x \in L$, то есть гарантированно убеждающее доказательство
  - если $x \notin L$, то любое доказательство срабатывает с малой вероятностью

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)
- Артур недоверчивый, полиномиально ограниченный, вероятностный
- и может прочитать лишь $O(1)$ битов доказательства
- полнота и корректность:
  - если $x \in L$, то есть гарантированно убеждающее доказательство
  - если $x \notin L$, то любое доказательство срабатывает с малой вероятностью
- такое возможно для всех задач из класса NP

# Локально проверяемые доказательства

- возвращаемся к привычной схеме: доказательство есть текст (двоичное слово)
- Артур недоверчивый, полиномиально ограниченный, вероятностный
- и может прочитать лишь $O(1)$ битов доказательства
- полнота и корректность:
  - если $x \in L$, то есть гарантированно убеждающее доказательство
  - если $x \notin L$, то любое доказательство срабатывает с малой вероятностью
- такое возможно для всех задач из класса NP
- связано с NP-трудностью приближённых задач: трудно отличить набор условий, которые можно выполнить одновременно, от набора условий, где можно выполнить не более $(1 - \varepsilon)$-доли

# Природа случайности

# Природа случайности

- свойство объекта или процесса? пример: тасовальная машина, отбраковывающая «плохо перетасованные колоды»

# Природа случайности

- свойство объекта или процесса? пример: тасовальная машина, отбраковывающая «плохо перетасованные колоды»
- случайный объект: несжимаемость (нельзя описать короче, чем перечислением всех битов)

# Природа случайности

- свойство объекта или процесса? пример: тасовальная машина, отбраковывающая «плохо перетасованные колоды»
- случайный объект: несжимаемость (нельзя описать короче, чем перечислением всех битов)
- псевдослучайный объект: описание короткое хотя и есть, но очень долго восстанавливается

# Природа случайности

- свойство объекта или процесса? пример: тасовальная машина, отбраковывающая «плохо перетасованные колоды»
- случайный объект: несжимаемость (нельзя описать короче, чем перечислением всех битов)
- псевдослучайный объект: описание короткое хотя и есть, но очень долго восстанавливается
- псевдослучайный генератор: случайность как незнание

# Природа случайности

- свойство объекта или процесса? пример: тасовальная машина, отбраковывающая «плохо перетасованные колоды»
- случайный объект: несжимаемость (нельзя описать короче, чем перечислением всех битов)
- псевдослучайный объект: описание короткое хотя и есть, но очень долго восстанавливается
- псевдослучайный генератор: случайность как незнание
  - отображение $G\colon \mathbb{B}^n \to \mathbb{B}^N$, где $N \gg n$

# Природа случайности

- свойство объекта или процесса? пример: тасовальная машина, отбраковывающая «плохо перетасованные колоды»
- случайный объект: несжимаемость (нельзя описать короче, чем перечислением всех битов)
- псевдослучайный объект: описание короткое хотя и есть, но очень долго восстанавливается
- псевдослучайный генератор: случайность как незнание
  - отображение $G\colon \mathbb{B}^n \to \mathbb{B}^N$, где $N \gg n$
  - для теста $T \subset \mathbb{B}^N$ считаем вероятности $x \in T$ (по $x \in \mathbb{B}^N$) и $G(s) \in T$ (по $s \in \mathbb{B}^n$).

# Природа случайности

- свойство объекта или процесса? пример: тасовальная машина, отбраковывающая «плохо перетасованные колоды»
- случайный объект: несжимаемость (нельзя описать короче, чем перечислением всех битов)
- псевдослучайный объект: описание короткое хотя и есть, но очень долго восстанавливается
- псевдослучайный генератор: случайность как незнание
  - отображение $G\colon \mathbb{B}^n \to \mathbb{B}^N$, где $N \gg n$
  - для теста $T \subset \mathbb{B}^N$ считаем вероятности $x \in T$ (по $x \in \mathbb{B}^N$) и $G(s) \in T$ (по $s \in \mathbb{B}^n$).
  - они должны быть близки для простых $T$ (но не для всех $T$, это невозможно)

# Разделение секрета

# Разделение секрета

- код от сейфа $= a + b \bmod N$, где $a$ выбирается случайно

# Разделение секрета

- код от сейфа $= a + b \bmod N$, где $a$ выбирается случайно
- $a$ и $b$ знают А и Б соответственно

# Разделение секрета

- код от сейфа $= a + b \bmod N$, где $a$ выбирается случайно
- $a$ и $b$ знают А и Б соответственно
- цифровой вариант сейфа с двумя ключами

# Разделение секрета

- код от сейфа $= a + b \bmod N$, где $a$ выбирается случайно
- $a$ и $b$ знают А и Б соответственно
- цифровой вариант сейфа с двумя ключами
- $5$ участников, надо $3$ для восстановления ключа:

# Разделение секрета

- код от сейфа $= a + b \bmod N$, где $a$ выбирается случайно
- $a$ и $b$ знают А и Б соответственно
- цифровой вариант сейфа с двумя ключами
- $5$ участников, надо $3$ для восстановления ключа:
- ключ — свободный член квадратного трёхчлена, остальные коэффициенты случайны

# Разделение секрета

- код от сейфа $= a + b \bmod N$, где $a$ выбирается случайно
- $a$ и $b$ знают А и Б соответственно
- цифровой вариант сейфа с двумя ключами
- $5$ участников, надо $3$ для восстановления ключа:
- ключ — свободный член квадратного трёхчлена, остальные коэффициенты случайны
- по трём точкам можно восстановить параболу, а по двум нет (все свободные члены равновозможны)

# Коммуникационная сложность

# Коммуникационная сложность

- выделение одного аспекта: передачи данных

# Коммуникационная сложность

- выделение одного аспекта: передачи данных
- задача: вычислить $P(a, b)$, если $a$ и $b$ у А и Б, и передача данных от А к Б и обратно стоит дорого

# Коммуникационная сложность

- выделение одного аспекта: передачи данных
- задача: вычислить $P(a, b)$, если $a$ и $b$ у А и Б, и передача данных от А к Б и обратно стоит дорого
- коммуникационная сложность алгоритма: число передаваемых битов в худшем случае

# Коммуникационная сложность

- выделение одного аспекта: передачи данных
- задача: вычислить $P(a, b)$, если $a$ и $b$ у А и Б, и передача данных от А к Б и обратно стоит дорого
- коммуникационная сложность алгоритма: число передаваемых битов в худшем случае
- проверка равенства $n$-битовых слов $a = b$ требует $n$ битов

# Коммуникационная сложность

- выделение одного аспекта: передачи данных
- задача: вычислить $P(a, b)$, если $a$ и $b$ у А и Б, и передача данных от А к Б и обратно стоит дорого
- коммуникационная сложность алгоритма: число передаваемых битов в худшем случае
- проверка равенства $n$-битовых слов $a = b$ требует $n$ битов
- доказательство: протоколы успешных сравнений должны быть различны

# Коммуникационная сложность

- выделение одного аспекта: передачи данных
- задача: вычислить $P(a, b)$, если $a$ и $b$ у А и Б, и передача данных от А к Б и обратно стоит дорого
- коммуникационная сложность алгоритма: число передаваемых битов в худшем случае
- проверка равенства $n$-битовых слов $a = b$ требует $n$ битов
- доказательство: протоколы успешных сравнений должны быть различны
- вероятностный алгоритм сложности $O(\log n)$

# Польза от отсутствия алгоритма

# Польза от отсутствия алгоритма

- Односторонние функции и пароли

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)
  - храним не пароли, а значения односторонней функции на них

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)
  - храним не пароли, а значения односторонней функции на них
  - тогда проверять пароль легко

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)
  - храним не пароли, а значения односторонней функции на них
  - тогда проверять пароль легко
  - но даже если враг получит хранимую таблицу, то не сможет зайти

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)
  - храним не пароли, а значения односторонней функции на них
  - тогда проверять пароль легко
  - но даже если враг получит хранимую таблицу, то не сможет зайти
- Бросание монеты по телефону

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)
  - храним не пароли, а значения односторонней функции на них
  - тогда проверять пароль легко
  - но даже если враг получит хранимую таблицу, то не сможет зайти
- Бросание монеты по телефону
  - в конечной игре с полной информацией у одного из игроков есть выигрышная стратегия

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)
  - храним не пароли, а значения односторонней функции на них
  - тогда проверять пароль легко
  - но даже если враг получит хранимую таблицу, то не сможет зайти
- Бросание монеты по телефону
  - в конечной игре с полной информацией у одного из игроков есть выигрышная стратегия
  - но может быть так, что у каждого есть вероятностная стратегия, которая гарантирует вероятность выигрыша около $1/2$ против любой простой стратегии противника (возможно, вероятностной)

# Польза от отсутствия алгоритма

- Односторонние функции и пароли
  - односторонняя функция: $x \to f(x)$ легко, обратно сложно (найти хотя бы один прообраз)
  - храним не пароли, а значения односторонней функции на них
  - тогда проверять пароль легко
  - но даже если враг получит хранимую таблицу, то не сможет зайти
- Бросание монеты по телефону
  - в конечной игре с полной информацией у одного из игроков есть выигрышная стратегия
  - но может быть так, что у каждого есть вероятностная стратегия, которая гарантирует вероятность выигрыша около $1/2$ против любой простой стратегии противника (возможно, вероятностной)
- Сложностная криптография: разделение ключа на открытую и закрытую части