Уводна лекция

Трифон Трифонов

λ-смятане и теория на доказателствата, 2018/19 г.

19–26 февруари 2019 г.

За курса

- теоретичен курс с практически елементи

За курса

- теоретичен курс с практически елементи
- основи на функционалното програмиране

За курса

- теоретичен курс с практически елементи
- основи на функционалното програмиране
- връзка между теоретичната информатика и математическата логика

За курса

- теоретичен курс с практически елементи
- основи на функционалното програмиране
- връзка между теоретичната информатика и математическата логика
- **изисквания**: базови познания по логика, алгебра и дискретна математика

За курса

- теоретичен курс с практически елементи
- основи на функционалното програмиране
- връзка между теоретичната информатика и математическата логика
- **изисквания**: базови познания по логика, алгебра и дискретна математика
- **оценка**: задачи за домашна работа

За курса

- теоретичен курс с практически елементи
- основи на функционалното програмиране
- връзка между теоретичната информатика и математическата логика
- **изисквания**: базови познания по логика, алгебра и дискретна математика
- **оценка**: задачи за домашна работа
- материали в `https://learn.fmi.uni-sofia.bg`

За курса

- теоретичен курс с практически елементи
- основи на функционалното програмиране
- връзка между теоретичната информатика и математическата логика
- **изисквания**: базови познания по логика, алгебра и дискретна математика
- **оценка**: задачи за домашна работа
- материали в `https://learn.fmi.uni-sofia.bg`

Да се запознаем!

Знаете ли какво е...

- ...функция и релация

Знаете ли какво е...

- ...функция и релация
- ...терм и формула

Знаете ли какво е...

- ...функция и релация
- ...терм и формула
- ...предикатно смятане от първи ред

Знаете ли какво е...

- ...функция и релация
- ...терм и формула
- ...предикатно смятане от първи ред
- ...функция от по-висок ред

Знаете ли какво е...

- ...функция и релация
- ...терм и формула
- ...предикатно смятане от първи ред
- ...функция от по-висок ред
- ...математическа индукция: обикновена, пълна и структурна

Знаете ли какво е...

- ...функция и релация
- ...терм и формула
- ...предикатно смятане от първи ред
- ...функция от по-висок ред
- ...математическа индукция: обикновена, пълна и структурна
- ...тип

Знаете ли какво е...

- ...функция и релация
- ...терм и формула
- ...предикатно смятане от първи ред
- ...функция от по-висок ред
- ...математическа индукция: обикновена, пълна и структурна
- ...тип
- ...динамично и статично типизиране

Знаете ли какво е...

- ...функция и релация
- ...терм и формула
- ...предикатно смятане от първи ред
- ...функция от по-висок ред
- ...математическа индукция: обикновена, пълна и структурна
- ...тип
- ...динамично и статично типизиране
- ...изчислима функция и разрешимо множество

Четирите стълба на математическата логика

- **Теория на изчислимостта**

Четирите стълба на математическата логика

- **Теория на изчислимостта**
- **Теория на доказателствата**

Четирите стълба на математическата логика

- **Теория на изчислимостта**
- **Теория на доказателствата**
- **Теория на множествата**

Четирите стълба на математическата логика

- **Теория на изчислимостта**
- **Теория на доказателствата**
- **Теория на множествата**
- **Теория на моделите**

Четирите стълба на математическата логика

- **Теория на изчислимостта** — ще засегнем
- **Теория на доказателствата**
- **Теория на множествата**
- **Теория на моделите**

Четирите стълба на математическата логика

- **Теория на изчислимостта** — ще засегнем
- **Теория на доказателствата** — ще въведем
- **Теория на множествата**
- **Теория на моделите**

Четирите стълба на математическата логика

- **Теория на изчислимостта** — ще засегнем
- **Теория на доказателствата** — ще въведем
- **Теория на множествата** — ще използваме
- **Теория на моделите**

Четирите стълба на математическата логика

- **Теория на изчислимостта** — ще засегнем
- **Теория на доказателствата** — ще въведем
- **Теория на множествата** — ще използваме
- **Теория на моделите** — ще споменем съвсем бегло

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)
  - Безкрайно малки величини, математически анализ (Newton, Leibniz)

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)
  - Безкрайно малки величини, математически анализ (Newton, Leibniz)
- XIX век — абстракция и формализация

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)
  - Безкрайно малки величини, математически анализ (Newton, Leibniz)
- XIX век — абстракция и формализация
  - аксиоматичен подход към анализа (Cauchy, Bolzano, Weierstrass, Dedekind)

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)
  - Безкрайно малки величини, математически анализ (Newton, Leibniz)
- XIX век — абстракция и формализация
  - аксиоматичен подход към анализа (Cauchy, Bolzano, Weierstrass, Dedekind)
  - теория на групите (Abel, Galois)

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)
  - Безкрайно малки величини, математически анализ (Newton, Leibniz)
- XIX век — абстракция и формализация
  - аксиоматичен подход към анализа (Cauchy, Bolzano, Weierstrass, Dedekind)
  - теория на групите (Abel, Galois)
  - неевклидови геометрии (Lambert, Лобачевский, Riemann)

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)
  - Безкрайно малки величини, математически анализ (Newton, Leibniz)
- XIX век — абстракция и формализация
  - аксиоматичен подход към анализа (Cauchy, Bolzano, Weierstrass, Dedekind)
  - теория на групите (Abel, Galois)
  - неевклидови геометрии (Lambert, Лобачевский, Riemann)
  - булева алгебра и логика (Boole, De Morgan, Peirce)

Малко история

- 300 г. пр.н.е. — Евклид “Елементи”
- XVII век — математически методи
  - координатна система (Descartes)
  - Безкрайно малки величини, математически анализ (Newton, Leibniz)
- XIX век — абстракция и формализация
  - аксиоматичен подход към анализа (Cauchy, Bolzano, Weierstrass, Dedekind)
  - теория на групите (Abel, Galois)
  - неевклидови геометрии (Lambert, Лобачевский, Riemann)
  - булева алгебра и логика (Boole, De Morgan, Peirce)
  - предикатно смятане от първи ред (Frege, Russel, Peano)

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)
  - формулировка: формален език за описание на цялата математика

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)
  - формулировка: формален език за описание на цялата математика
  - пълнота: всички математически теореми могат да се докажат

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)
  - формулировка: формален език за описание на цялата математика
  - пълнота: всички математически теореми могат да се докажат
  - съвместимост: не можем да докажем противоречиви твърдения

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)
  - формулировка: формален език за описание на цялата математика
  - пълнота: всички математически теореми могат да се докажат
  - съвместимост: не можем да докажем противоречиви твърдения
  - разрешимост: алгоритъм за проверка на верност на твърдения

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)
  - формулировка: формален език за описание на цялата математика
  - ~~пълнота: всички математически теореми могат да се докажат~~
    - Kurt Gödel, 1931: първа теорема за непълнота
  - съвместимост: не можем да докажем противоречиви твърдения
  - разрешимост: алгоритъм за проверка на верност на твърдения

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)
  - формулировка: формален език за описание на цялата математика
  - ~~пълнота: всички математически теореми могат да се докажат~~
    - Kurt Gödel, 1931: първа теорема за непълнота
  - ~~съвместимост: не можем да докажем противоречиви твърдения~~
    - Kurt Gödel, 1931: втора теорема за непълнота
  - разрешимост: алгоритъм за проверка на верност на твърдения

Фундаментална криза в математиката (XX век)

- Парадокс на Ръсел
  - Нека $A := \{x \notin x\}$
  - $A \in A$ или $A \notin A$?!?
- Философски школи
  - формализъм (David Hilbert)
  - интуиционизъм (L.E.J. Brouwer)
  - логицизъм (Bertrand Russell, Alfred North Whitehead)
- Програма на Hilbert (1900–1920)
  - формулировка: формален език за описание на цялата математика
  - ~~пълнота: всички математически теореми могат да се докажат~~
    - Kurt Gödel, 1931: първа теорема за непълнота
  - ~~съвместимост: не можем да докажем противоречиви твърдения~~
    - Kurt Gödel, 1931: втора теорема за непълнота
  - ~~разрешимост: алгоритъм за проверка на верност на твърдения~~
    - Alan Turing, Alonzo Church, 1936: съществуват неразрешими твърдения

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \to \mathbb{N}$ е функция над естествени числа.

Примери: $f(x) = x^2$, $f(x) = x$-тото число на Фибоначи.

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \to \mathbb{N}$ е функция над естествени числа.

Примери: $f(x) = x^2$, $f(x) = x$-тото число на Фибоначи.

Въпрос 1: Какво означава да изчислим f с компютър?

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \to \mathbb{N}$ е функция над естествени числа.

Примери: $f(x) = x^2$, $f(x) = x$-тото число на Фибоначи.

Въпрос 1: Какво означава да изчислим f с компютър?

Въпрос 2: Какво означава “алгоритъм” или “програма”?

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \to \mathbb{N}$ е функция над естествени числа.

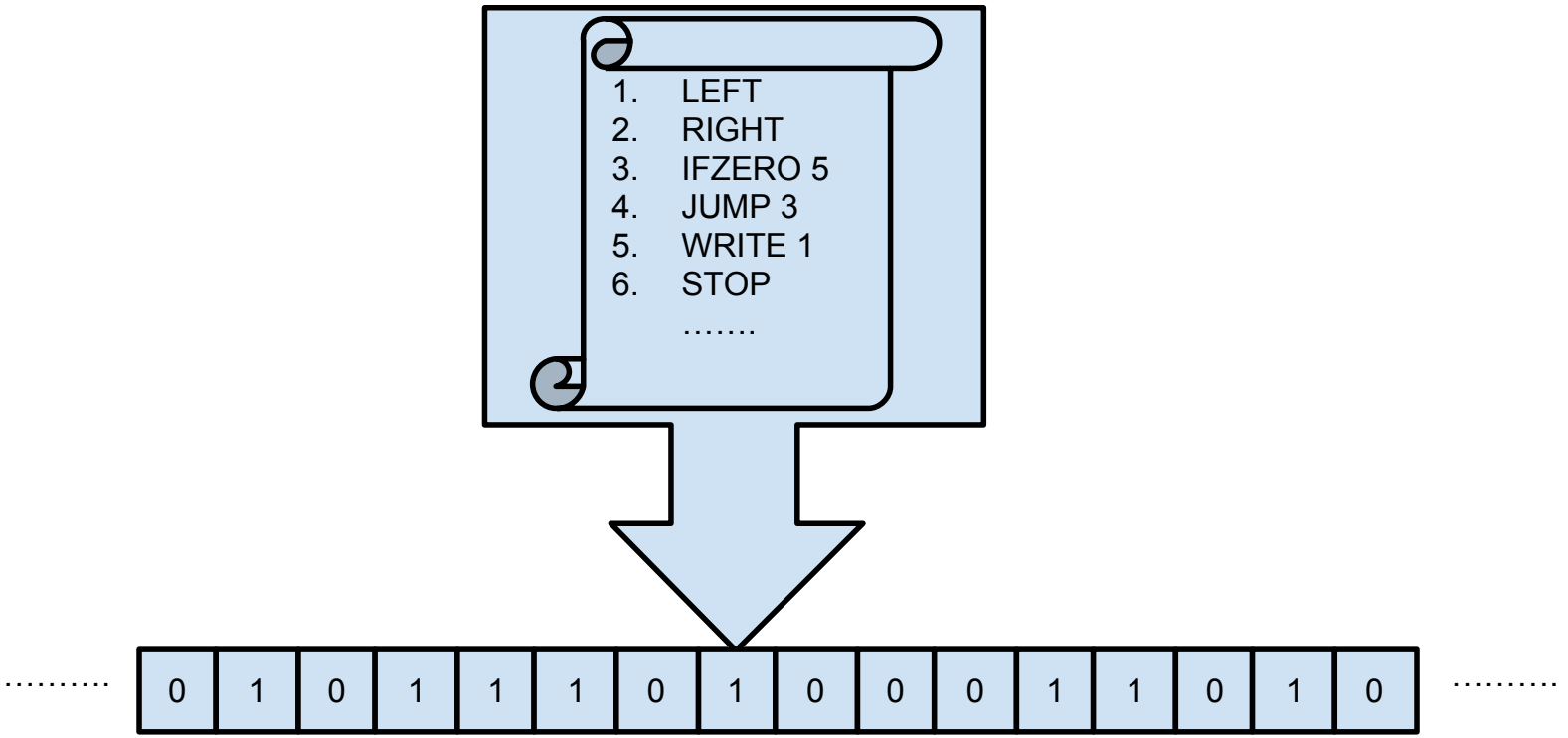
Примери: $f(x) = x^2$, $f(x) = x$-тото число на Фибоначи.

Въпрос 1: Какво означава да изчислим f с компютър?

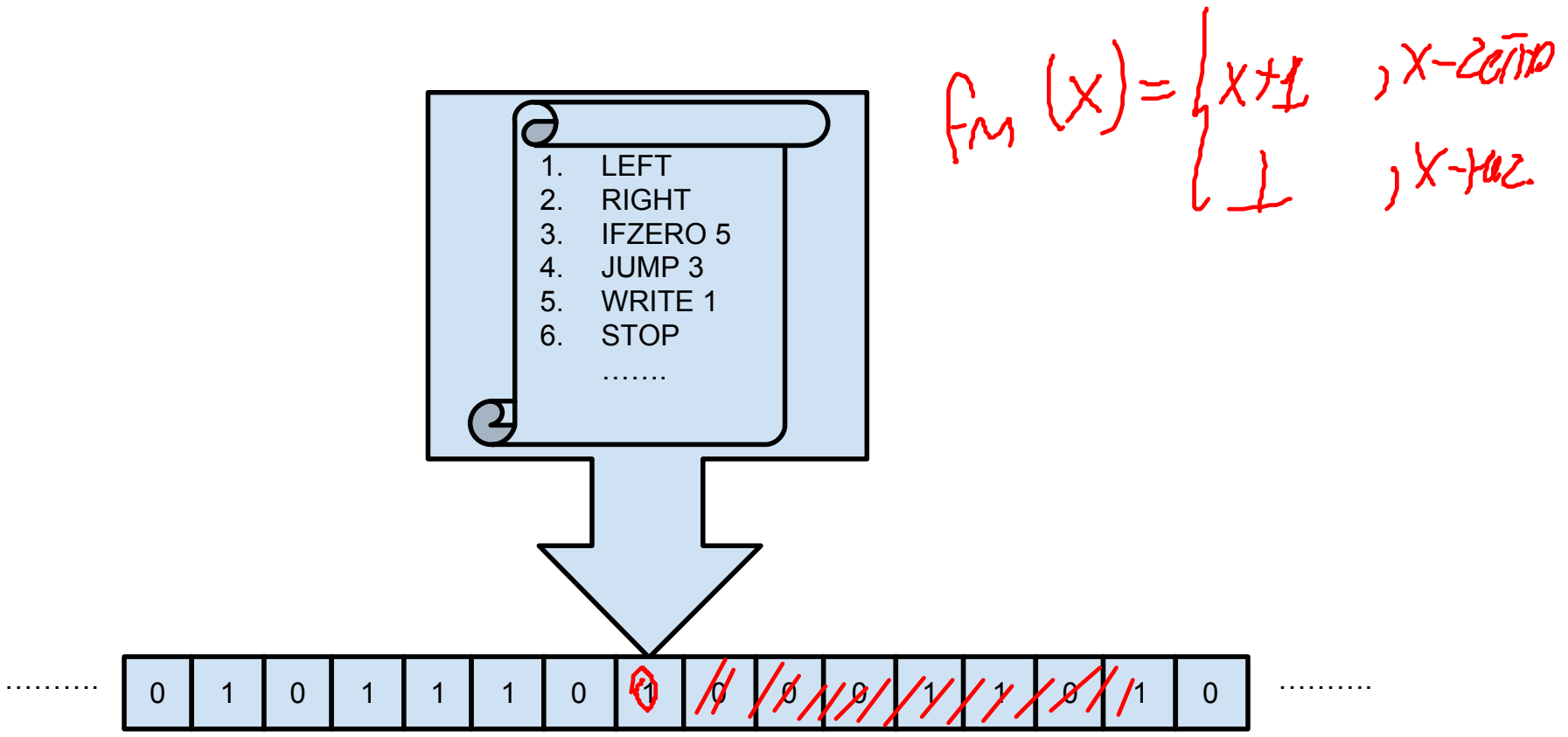
Въпрос 2: Какво означава “алгоритъм” или “програма”?

Въпрос 3: Има ли функции, които не могат да бъдат изчислени с компютър?

Машина на Turing



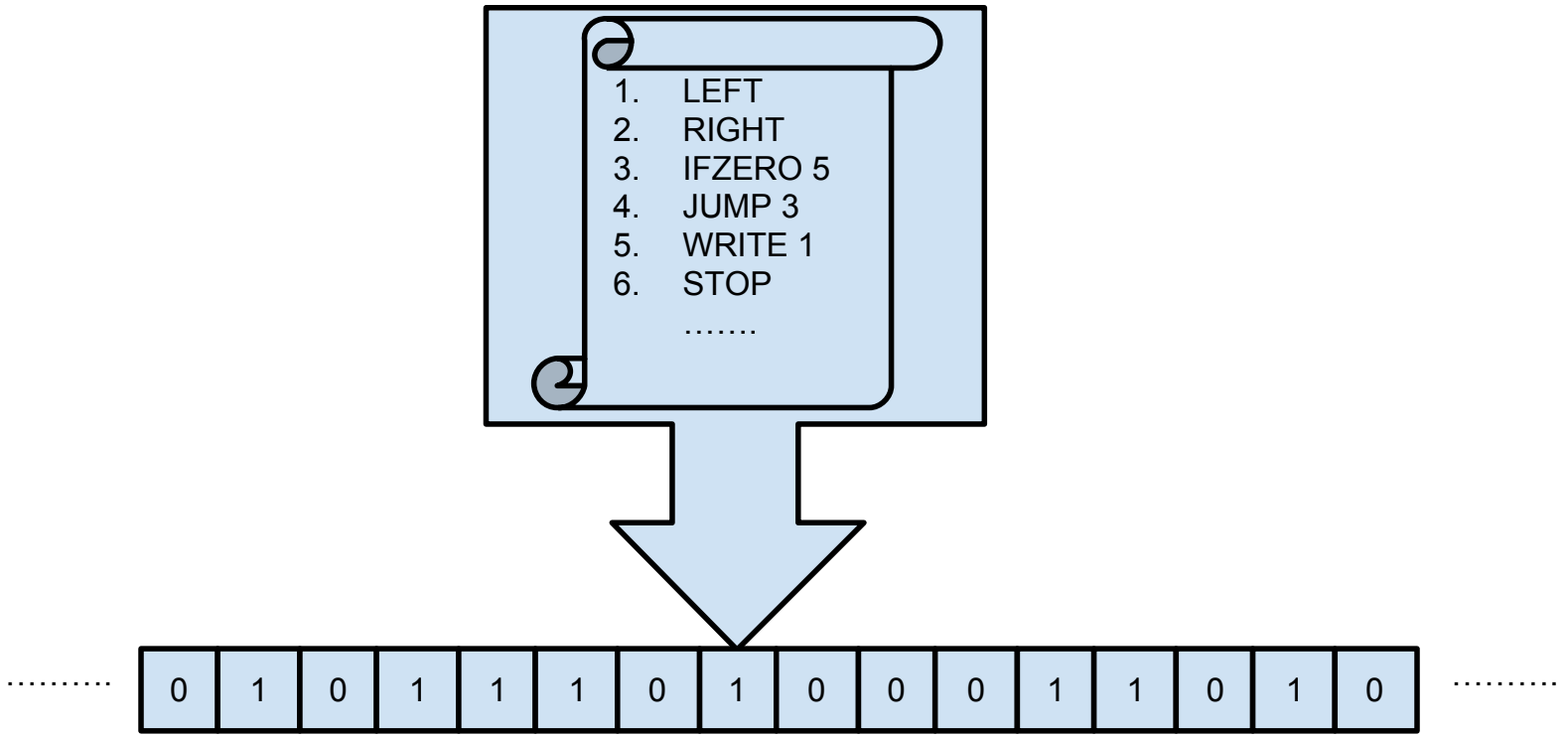
Машина на Turing



$f_M(x) = \begin{cases} x+1 & , x \text{-сетен} \\ \perp & , x \text{-нед.} \end{cases}$

M **изчислява функцията** f_M, ако при лента с числото n машината M завършва и записва върху лентата числото $f_M(n)$.

Машина на Turing



M **изчислява функцията** f_M, ако при лента с числото n машината M завършва и записва върху лентата числото $f_M(n)$.
Ако M не завърши за някое n, казваме, че $f_M(n)$ **не е дефинирана**.

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \to \mathbb{N}$ са колкото редиците от естествени числа ...

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \to \mathbb{N}$ са колкото редиците от естествени числа ...
- ... които са неизброимо много! (защо?).

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \to \mathbb{N}$ са колкото редиците от естествени числа ...
- ... които са неизброимо много! (защо?).
- Следователно, има неизброимо много неизчислими функции. □

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \to \mathbb{N}$ са колкото редиците от естествени числа ...
- ... които са неизброимо много! (защо?).
- Следователно, има неизброимо много неизчислими функции. □
- Но кои са те?

Стоп проблем

Нека с $\{n\}$ означаваме машината на Turing с код n.
Разглеждаме функцията:

$$halts(n) = \begin{cases} 1, & \text{ако } \{n\} \text{ завършва над лента с числото } n, \\ 0, & \text{иначе.} \end{cases}$$

Стоп проблем

Нека с $\{n\}$ означаваме машината на Turing с код n.
Разглеждаме функцията:

$$halts(n) = \begin{cases} 1, & \text{ако } \{n\} \text{ завършва над лента с числото } n, \\ 0, & \text{иначе.} \end{cases}$$

halts не е изчислима!

Стоп проблем

Нека с $\{n\}$ означаваме машината на Turing с код n.
Разглеждаме функцията:

$$halts(n) = \begin{cases} 1, & \text{ако } \{n\} \text{ завършва над лента с числото } n, \\ 0, & \text{иначе.} \end{cases}$$

halts не е изчислима!

Доказателство.

Да допуснем, че *halts* се изчислява от машина на Turing H.
Дефинираме нова машина на Turing D:

1. (тук слагаме всички инструкции на H)

$k+1$. `IFZERO` $k+3$

$k+2$. `JUMP` $k+1$

$k+3$. `STOP`

Нека $D = \{d\}$. Завършва ли D над d?

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число
- $f_3(n) = n$-тата цифра на числото π

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число
- $f_3(n) = n$-тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число
- $f_3(n) = n$-тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3, които могат да се умножат в някакъв ред, така че да се получи O

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число
- $f_3(n) = n$-тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3, които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число
- $f_3(n) = n$-тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3, които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула
- $f_7(n) = n$ е код на вярна предикатна формула

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число
- $f_3(n) = n$-тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3, които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула
- $f_7(n) = n$ е код на вярна предикатна формула
- $f_8(n) = m$, където $\{m\}$ пресмята f_8

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$-тото поред просто число
- $f_3(n) = n$-тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3, които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула
- $f_7(n) = n$ е код на вярна предикатна формула
- $f_8(n) = m$, където $\{m\}$ пресмята f_8
- $f_9(n) =$ машините $\{n\}$ и $\{2n\}$ изчисляват еднакви функции

Разрешими и полуразрешими множества

Нека A е множество и дефинираме функциите:

$$c_A(x) := \begin{cases} 1, & \text{ако } x \in A, \\ 0, & \text{ако } x \notin A. \end{cases}$$

$$\chi_A(x) := \begin{cases} 1, & \text{ако } x \in A, \\ \bot, & \text{ако } x \notin A. \end{cases}$$

Разрешими и полуразрешими множества

Нека A е множество и дефинираме функциите:

$$c_A(x) := \begin{cases} 1, & \text{ако } x \in A, \\ 0, & \text{ако } x \notin A. \end{cases} \qquad \chi_A(x) := \begin{cases} 1, & \text{ако } x \in A, \\ \bot, & \text{ако } x \notin A. \end{cases}$$

A е **разрешимо**, ако c_A е изчислима.

Разрешими и полуразрешими множества

Нека A е множество и дефинираме функциите:

$$c_A(x) := \begin{cases} 1, & \text{ако } x \in A, \\ 0, & \text{ако } x \notin A. \end{cases} \qquad \chi_A(x) := \begin{cases} 1, & \text{ако } x \in A, \\ \bot, & \text{ако } x \notin A. \end{cases}$$

A е **разрешимо**, ако c_A е изчислима.

A е **полуразрешимо**, ако χ_A е изчислима.

λ-смятане

Нека разполагаме с изброимо много променливи $x, y, z, \ldots$

λ-смятане

Нека разполагаме с изброимо много променливи $x, y, z, \ldots$

Три вида λ-изрази (E)

- x (променлива)
- $E_1(E_2)$ (апликация, прилагане на функция)
- $\lambda x\, E$ (абстракция, конструиране на функция)

λ-смятане

Нека разполагаме с изброимо много променливи $x, y, z, \ldots$

Три вида λ-изрази (E)

- x (променлива)
- $E_1(E_2)$ (апликация, прилагане на функция)
- $\lambda x\, E$ (абстракция, конструиране на функция)

Примери: $\lambda x\, x, \quad (\lambda x\, x)(z), \quad \lambda f \lambda x\, f(f(f(x)))$

λ-смятане

Нека разполагаме с изброимо много променливи $x, y, z, \ldots$

Три вида λ-изрази (E)

- x (променлива)
- $E_1(E_2)$ (апликация, прилагане на функция)
- $\lambda x\, E$ (абстракция, конструиране на функция)

Примери: $\lambda x\, x, \quad (\lambda x\, x)(z), \quad \lambda f \lambda x\, f(f(f(x)))$

Едно изчислително правило:

$$(\lambda x\, E_1)(E_2) \mapsto E_1[x := E_2].$$

Машини на Turing = λ-смятане

Theorem 1 (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ-израз.

Машини на Turing = λ-смятане

Theorem 1 (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ-израз.

Машини на Turing	=	императивен стил за програмиране
λ-смятане	=	функционален стил за програмиране

Машини на Turing = λ-смятане

Theorem 1 (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ-израз.

Машини на Turing	=	императивен стил за програмиране
λ-смятане	=	функционален стил за програмиране

Факт: Практически всички съвременни езици за програмиране са със същата изчислителна сила като на машините на Turing.

Машини на Turing = λ-смятане

Theorem 1 (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ-израз.

Машини на Turing	=	императивен стил за програмиране
λ-смятане	=	функционален стил за програмиране

Факт: Практически всички съвременни езици за програмиране са със същата изчислителна сила като на машините на Turing.

Тезис на Church-Turing: Всяка функция, чието изчисление може да се автоматизира, може да бъде пресметната с машина на Turing.

Какво е доказателство?

- свидетелство за верността на дадено твърдение

Какво е доказателство?

- свидетелство за верността на дадено твърдение
- подчинено на прости и общоприети **логически правила**

Какво е доказателство?

- свидетелство за верността на дадено твърдение
- подчинено на прости и общоприети **логически правила**
- изхождащо от базови и общоприети **логически аксиоми**

Какво е доказателство?

- свидетелство за верността на дадено твърдение
- подчинено на прости и общоприети **логически правила**
- изхождащо от базови и общоприети **логически аксиоми**
- твърденията и доказателствата са описани на **формален език**

Какво е доказателство?

- свидетелство за верността на дадено твърдение
- подчинено на прости и общоприети **логически правила**
- изхождащо от базови и общоприети **логически аксиоми**
- твърденията и доказателствата са описани на **формален език**
- проверка за вярност на твърдение може да не е разрешима...

Какво е доказателство?

- свидетелство за верността на дадено твърдение
- подчинено на прости и общоприети **логически правила**
- изхождащо от базови и общоприети **логически аксиоми**
- твърденията и доказателствата са описани на **формален език**
- проверка за вярност на твърдение може да не е разрешима...
- ...но проверката за коректност на доказателство трябва да е!

Какво е доказателство?

- свидетелство за верността на дадено твърдение
- подчинено на прости и общоприети **логически правила**
- изхождащо от базови и общоприети **логически аксиоми**
- твърденията и доказателствата са описани на **формален език**
- проверка за вярност на твърдение може да не е разрешима...
- ...но проверката за коректност на доказателство трябва да е!
- разглеждаме доказателствата като синтактични обекти

Формализъм

David Hilbert — основател на формализма

> *Математиката е игра на символи.*

Формализъм

David Hilbert — основател на формализма

> *Математиката е игра на символи.*

- формулите са низове от символи

Формализъм

David Hilbert — основател на формализма

> *Математиката е игра на символи.*

- формулите са низове от символи
- логическите правила са средства за манипулация на низове

Формализъм

David Hilbert — основател на формализма

> *Математиката е игра на символи.*

- формулите са низове от символи
- логическите правила са средства за манипулация на низове
- синтаксисът на формулите и доказателствата е изолиран от семантиката им

Формализъм

David Hilbert — основател на формализма

> *Математиката е игра на символи.*

- формулите са низове от символи
- логическите правила са средства за манипулация на низове
- синтаксисът на формулите и доказателствата е изолиран от семантиката им
- $A \models B$ — за всяка интерпретация, в която A е вярна, B също е вярна

Формализъм

David Hilbert — основател на формализма

Математиката е игра на символи.

- формулите са низове от символи
- логическите правила са средства за манипулация на низове
- синтаксисът на формулите и доказателствата е изолиран от семантиката им
- $A \models B$ — за всяка интерпретация, в която A е вярна, B също е вярна
  - семантично (моделно) следствие

Формализъм

David Hilbert — основател на формализма

Математиката е игра на символи.

- формулите са низове от символи
- логическите правила са средства за манипулация на низове
- синтаксисът на формулите и доказателствата е изолиран от семантиката им
- $A \models B$ — за всяка интерпретация, в която A е вярна, B също е вярна
  - семантично (моделно) следствие
- $A \vdash B$ — B може да се получи от A чрез формални трансформации

Формализъм

David Hilbert — основател на формализма

Математиката е игра на символи.

- формулите са низове от символи
- логическите правила са средства за манипулация на низове
- синтаксисът на формулите и доказателствата е изолиран от семантиката им
- $A \models B$ — за всяка интерпретация, в която A е вярна, B също е вярна
  - семантично (моделно) следствие
- $A \vdash B$ — B може да се получи от A чрез формални трансформации
  - синтактично (формално) следствие

Неконструктивни доказателства (1)

Твърдение 1

Съществуват числа $a, b \notin \mathbb{Q}$, *така че* $a^b \in \mathbb{Q}$.

Неконструктивни доказателства (1)

Твърдение 1

Съществуват числа $a, b \notin \mathbb{Q}$, *така че* $a^b \in \mathbb{Q}$.

Доказателство.

Вярно ли е, че $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$?

Неконструктивни доказателства (1)

Твърдение 1

Съществуват числа $a, b \notin \mathbb{Q}$, *така че* $a^b \in \mathbb{Q}$.

Доказателство.

Вярно ли е, че $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$?

- Да: тогава $a := b := \sqrt{2}$.

Неконструктивни доказателства (1)

Твърдение 1

Съществуват числа $a, b \notin \mathbb{Q}$, *така че* $a^b \in \mathbb{Q}$.

Доказателство.

Вярно ли е, че $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$?

- Да: тогава $a := b := \sqrt{2}$.
- Не: тогава
  $a := \sqrt{2}^{\sqrt{2}}, \quad b := \sqrt{2}, \quad a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{(\sqrt{2}\sqrt{2})} = (\sqrt{2})^2 = 2.$

Неконструктивни доказателства (1)

Твърдение 1

Съществуват числа $a, b \notin \mathbb{Q}$, *така че* $a^b \in \mathbb{Q}$.

Доказателство.

Вярно ли е, че $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$?

- Да: тогава $a := b := \sqrt{2}$.
- Не: тогава
  $a := \sqrt{2}^{\sqrt{2}}, \quad b := \sqrt{2}, \quad a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{(\sqrt{2}\sqrt{2})} = (\sqrt{2})^2 = 2.$

□

Но всъщност кои са стойностите на a и b?

Неконструктивни доказателства (2)

Твърдение 2 (Формула за пияниците)

Във всеки непразен бар има такъв човек, че ако той пие, то всички пият.

Неконструктивни доказателства (2)

Твърдение 2 (Формула за пияниците)

Във всеки непразен бар има такъв човек, че ако той пие, то всички пият.

Нека $d(x) :=$ *“*x *пие”. Тогава* $\exists_x(d(x) \to \forall_y d(y))$.

Неконструктивни доказателства (2)

Твърдение 2 (Формула за пияниците)

Във всеки непразен бар има такъв човек, че ако той пие, то всички пият.

Нека $d(x) :=$ *"*x *пие"*. *Тогава* $\exists_x(d(x) \to \forall_y d(y))$.

Доказателство.

Вярно ли е, че всички пият $(\forall_x d(x))$?

Неконструктивни доказателства (2)

Твърдение 2 (Формула за пияниците)

Във всеки непразен бар има такъв човек, че ако той пие, то всички пият.

Нека $d(x) :=$ *"*x *пие"*. *Тогава* $\exists_x(d(x) \to \forall_y d(y))$.

Доказателство.

Вярно ли е, че всички пият $(\forall_x d(x))$?

- Да: избираме произволен човек c от бара, той пие, но и всички други пият.

□

Неконструктивни доказателства (2)

Твърдение 2 (Формула за пияниците)

Във всеки непразен бар има такъв човек, че ако той пие, то всички пият.

Нека $d(x) :=$ *"*x *пие"*. *Тогава* $\exists_x(d(x) \to \forall_y d(y))$.

Доказателство.

Вярно ли е, че всички пият $(\forall_x d(x))$?

- Да: избираме произволен човек c от бара, той пие, но и всички други пият.
- Не: избираме човека x, който не пие, заради него не всички пият.

□

Неконструктивни доказателства (2)

Твърдение 2 (Формула за пияниците)

Във всеки непразен бар има такъв човек, че ако той пие, то всички пият.

Нека $d(x) :=$ *"*x *пие"*. *Тогава* $\exists_x(d(x) \to \forall_y d(y))$.

Доказателство.

Вярно ли е, че всички пият $(\forall_x d(x))$?

- Да: избираме произволен човек c от бара, той пие, но и всички други пият.
- Не: избираме човека x, който не пие, заради него не всички пият.

□

Но всъщност кой е човекът, заради когото всички пият?

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

- Допускаме само доказателства, които съдържат в себе си явна конструкция.

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

- Допускаме само доказателства, които съдържат в себе си явна конструкция.
- Да кажем, че съществува обект с дадено свойство, означава да посочим обекта.

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

- Допускаме само доказателства, които съдържат в себе си явна конструкция.
- Да кажем, че съществува обект с дадено свойство, означава да посочим обекта.
- Да кажем, че една от две възможности е в сила, означава да посочим коя от двете е в сила.

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

- Допускаме само доказателства, които съдържат в себе си явна конструкция.
- Да кажем, че съществува обект с дадено свойство, означава да посочим обекта.
- Да кажем, че една от две възможности е в сила, означава да посочим коя от двете е в сила.
- Формули, които не са интуиционистки верни:

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

- Допускаме само доказателства, които съдържат в себе си явна конструкция.
- Да кажем, че съществува обект с дадено свойство, означава да посочим обекта.
- Да кажем, че една от две възможности е в сила, означава да посочим коя от двете е в сила.
- Формули, които не са интуиционистки верни:
  - $A \vee \neg A$

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

- Допускаме само доказателства, които съдържат в себе си явна конструкция.
- Да кажем, че съществува обект с дадено свойство, означава да посочим обекта.
- Да кажем, че една от две възможности е в сила, означава да посочим коя от двете е в сила.
- Формули, които не са интуиционистки верни:
  - $A \vee \neg A$
  - $\neg\neg A \to A$

Интуиционистка (конструктивна) логика

L.E.J. Brouwer — основател на интуиционизма.

Математиката е плод на мисловна конструкция.

- Допускаме само доказателства, които съдържат в себе си явна конструкция.
- Да кажем, че съществува обект с дадено свойство, означава да посочим обекта.
- Да кажем, че една от две възможности е в сила, означава да посочим коя от двете е в сила.
- Формули, които не са интуиционистки верни:
  - $A \vee \neg A$
  - $\neg\neg A \to A$
  - $\neg\forall_x \neg A(x) \to \exists_x A(x)$

λ-смятане и теория на доказателствата

Съществува красива и естествена връзка между λ-смятане и (интуиционистката) теория на доказателствата.

λ-смятане и теория на доказателствата

Съществува красива и естествена връзка между λ-смятане и (интуиционистката) теория на доказателствата.

Доказателствата могат да се разглеждат като програми, а програмите като доказателства!

λ-смятане и теория на доказателствата

Съществува красива и естествена връзка между λ-смятане и (интуиционистката) теория на доказателствата.

Доказателствата могат да се разглеждат като програми, а програмите като доказателства!

. . . но за това ще говорим по-нататък.

За математиката в програмирането

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?

За математиката в програмирането

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?

За математиката в програмирането

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?
- Може ли тези доказателства да стават автоматично?

За математиката в програмирането

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?
- Може ли тези доказателства да стават автоматично?
- Може ли програмата да се проверява за коректност при компилация?

Формална верификация на програми

Нека с P означим програма с входни променливи $\vec{x}$ и изходни променливи $\vec{y} := P(\vec{x})$.

Нека с $A(\vec{x}, \vec{y})$ означим някаква релация между $\vec{x}$ и $\vec{y}$.

Формална верификация на програми

Нека с P означим програма с входни променливи $\vec{x}$ и изходни променливи $\vec{y} := P(\vec{x})$.

Нека с $A(\vec{x}, \vec{y})$ означим някаква релация между $\vec{x}$ и $\vec{y}$.

Верификация. Дадени са A и P. Искаме да проверим дали A е в сила за P, т.е. дали $A(\vec{x}, P(\vec{x}))$.

Формална верификация на програми

Нека с P означим програма с входни променливи $\vec{x}$ и изходни променливи $\vec{y} := P(\vec{x})$.

Нека с $A(\vec{x}, \vec{y})$ означим някаква релация между $\vec{x}$ и $\vec{y}$.

Верификация. Дадени са A и P. Искаме да проверим дали A е в сила за P, т.е. дали $A(\vec{x}, P(\vec{x}))$.

Как да направим тази проверка?

Логика на Хоар

Разглеждаме тройки от вида

{Предусловие} Програма {Следусловие}

Логика на Хоар

Разглеждаме тройки от вида

$$\{\text{Предусловие}\}\ \text{Програма}\ \{\text{Следусловие}\}$$

Строим дървета с тройки по възлите по следните правила:

$$\{P\}\ ;\ \{P\} \qquad \text{(празен оператор)}$$

Логика на Хоар

Разглеждаме тройки от вида

$$\{\text{Предусловие}\}\ \text{Програма}\ \{\text{Следусловие}\}$$

Строим дървета с тройки по възлите по следните правила:

$$\{P\}\ ;\ \{P\} \qquad \text{(празен оператор)}$$

$$\{P[x \mapsto E]\}\ x = E;\ \{P\} \qquad \text{(присвояване)}$$

Логика на Хоар (2)

$$\frac{\{P\}\ S\ \{R\} \qquad \{R\}\ T\ \{Q\}}{\{P\}\ S; T\ \{Q\}}$$

(композиция)

Логика на Хоар (2)

$$\frac{\{P\}\ S\ \{R\} \qquad \{R\}\ T\ \{Q\}}{\{P\}\ S; T\ \{Q\}}$$ (композиция)

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad \{P \wedge \neg B\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ (B)\ S\ \textbf{else}\ T\ \{Q\}}$$ (условен оператор)

Логика на Хоар (2)

$$\frac{\{P\}\ S\ \{R\} \qquad \{R\}\ T\ \{Q\}}{\{P\}\ S; T\ \{Q\}} \quad \text{(композиция)}$$

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad \{P \wedge \neg B\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ (B)\ S\ \textbf{else}\ T\ \{Q\}} \quad \text{(условен оператор)}$$

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ (B)\ S\ \{P \wedge \neg B\}} \quad \text{(цикъл)}$$

P — **инвариант** на цикъла

Логика на Хоар (3)

$$\frac{\{P\}\ S\ \{Q\} \qquad R \to P}{\{R\}\ S\ \{Q\}} \quad \text{(усилване)}$$

$$\frac{\{P\}\ S\ \{Q\} \qquad Q \to R}{\{P\}\ S\ \{R\}} \quad \text{(отслабване)}$$

Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Търсим инвариант P такъв, че:

1. $(x = 0 \wedge i = 1) \rightarrow P$

Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Търсим инвариант P такъв, че:

1. $(x = 0 \wedge i = 1) \rightarrow P$
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$

Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Търсим инвариант P такъв, че:

1. $(x = 0 \wedge i = 1) \rightarrow P$
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$

Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$$\boxed{P : x = \frac{n(i-1)}{2}}$$

Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{n(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓

Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{n(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$?

Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{n(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$?
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$$\boxed{P : x = \frac{i(i-1)}{2}}$$

Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{i(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓

Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{i(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ✓

Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{i(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ✓
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

Езици за анотиране на програми

- Java Modeling Language (JML) — Java
- ANSI/ISO C Specification Language (ACSL) — C
- Typed Assembly Language (TAL) — Assembler
- Property Specifiation Language (PSL) — Verilog (хардуер)

Пример: факториел

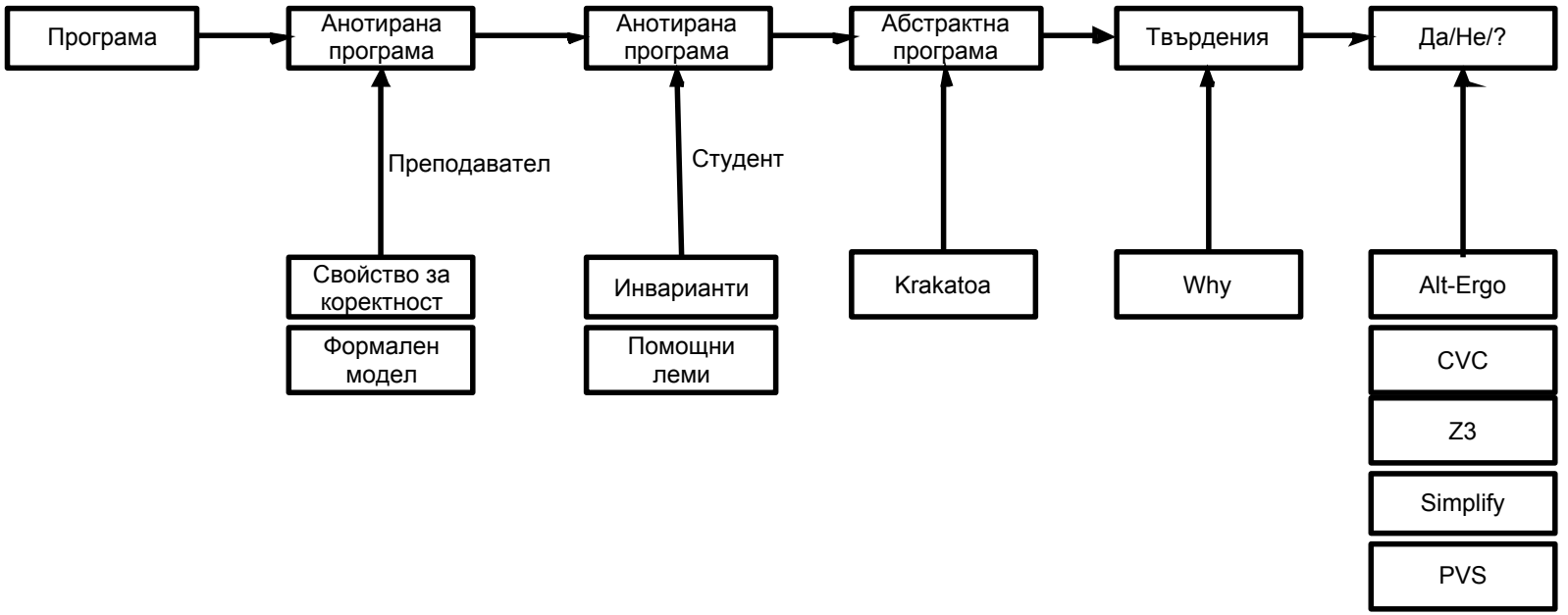
Модел:

```
/*@ inductive isfact(integer x, integer r) {
  @  case isfact0: isfact(0,1);
  @  case isfactn:
  @   \forall integer n r;
  @     n >= 1 && isfact(n-1,r) ==> isfact(n,r*n);
  @ }
  @*/
```

Пример: факториел (2)

```
/*@ requires x >= 0;
  @ ensures isfact(x, \result);
  @*/
public static int fact(int x) {
    int a = 0, y = 1;
    /*@ loop_invariant 0 <= a <= x && isfact(a,y);
      @ loop_variant x-a;
      @*/
    while (a < x) y = y * ++a;
    return y;
}
```

Обща схема


Програма
Анотирана програма
Анотирана програма
Абстрактна програма
Твърдения
Да/Не/?
Преподавател
Студент
Свойство за коректност
Формален модел
Инварианти
Помощни леми
Krakatoa
Why
Alt-Ergo
CVC
Z3
Simplify
PVS

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

Дори има **World Championship for Automated Theorem Proving**

Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони

Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)

Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)
- чрез изследване на предикатите в програмата

Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)
- чрез изследване на предикатите в програмата
- трансформации на пред- и след-условието

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

Примери: Dafny, Why

Какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист

Какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл

Какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери

Какво точно означава "да докажем"?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар

Какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист

А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл

А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек

А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг

А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

Стига глупости! Програмата се пише на програмен език с формален синтаксис, който може да се изпълни от машина!

Какво точно означава "да докажем"?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

Стига глупости! Доказателството се пише на език с формален синтаксис, така че да може да се провери от машина!

Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране

Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма

Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация

Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор

Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматично доказване на теореми $\rightsquigarrow$ автоматично генериране на програми

Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматично доказване на теореми $\rightsquigarrow$ автоматично генериране на програми
- полуавтоматично доказване на теореми $\rightsquigarrow$ писане на програми с autocomplete

Пример за формално доказателство: Евклид

Теорема

Съществуват безкрайно много прости числа.

Пример за формално доказателство: Евклид

Теорема

Съществуват безкрайно много прости числа. С други думи: за всяко естествено число n съществува просто число $p > n$.

Пример за формално доказателство: Евклид

Теорема

Съществуват безкрайно много прости числа. С други думи: за всяко естествено число n съществува просто число $p > n$.

Доказателство.

Нека $k := n! + 1$ и нека p е произволен прост делител на k.

Пример за формално доказателство: Евклид

Теорема

Съществуват безкрайно много прости числа. С други думи: за всяко естествено число n съществува просто число $p > n$.

Доказателство.

Нека $k := n! + 1$ и нека p е произволен прост делител на k. Ако допуснем, че $p \leq n$, тогава p е делител на $n!$

Пример за формално доказателство: Евклид

Теорема

Съществуват безкрайно много прости числа. С други думи: за всяко естествено число n съществува просто число $p > n$.

Доказателство.

Нека $k := n! + 1$ и нека p е произволен прост делител на k. Ако допуснем, че $p \leq n$, тогава p е делител на $n!$ Но това е невъзможно, тъй като p е делител на k.

Пример за формално доказателство: Евклид

Теорема

Съществуват безкрайно много прости числа. С други думи: за всяко естествено число n съществува просто число $p > n$.

Доказателство.

Нека $k := n! + 1$ и нека p е произволен прост делител на k. Ако допуснем, че $p \leq n$, тогава p е делител на $n!$ Но това е невъзможно, тъй като p е делител на k. Тогава $p > n$. □

Пример за формално доказателство: Lego

$\lambda n{:}|\mathbb{N}|.[\leq = ap2\_\mathbb{N}\_\mathbb{N}\_\Omega\ lessEq][<= ap2\_\mathbb{N}\_\mathbb{N}\_\Omega\ lessE][k{=}succ(fac\ n)$
$]\ has\_prime\_factor\ k(\leq\_elim\_2(fac\ n)(le\_one\_fac\ n))(\exists y{:}|\mathbb{N}|.(n{<}y\ \&\ y$
$\leq k\ \&\ prime(y)))(\lambda y{:}|\mathbb{N}|.\lambda P{:}is\_prime\_factor\ y\ k.[D{=}fst(y|k)(prime(y)$
$)P][Q{=}snd(y|k)(prime(y))P][H{=}fst(1{<}y)(\Pi u{:}|\mathbb{N}|.(1{<}u){\to}(u{<}y){\to}(u$
$\not|\ y))Q]ExIntro(|\mathbb{N}|)\ y\ (\lambda v{:}|\mathbb{N}|.(n{<}v\ \&\ v{\leq}k\ \&\ prime(v))(pair(n{<}y\ \&\ y$
$\leq\ k\ (prime(y))(pair(n{<}y)\ (y{\leq}k)(<\_intro\ y\ n\ (\lambda F{:}y{\leq}n.<\_irrefl\ 1\ ($
$extenRel\_\mathbb{N}\_\mathbb{N}\ <\ 1\ (Eq\_refl\_\mathbb{N}\ 1)y\ 1\ (divides\_lemma\_3\ y\ (fac\ n)\ ($
$fac\_divides\ y\ n(\leq\_intro\_2\ y\ (<\_succ\_intro\ 1\ y\ H))F)D)H)\bot))($
$divides\_lemma\_1\ y\ k\ D\ (y{\leq}k\ (Id\ (y{\leq}k))\ (\lambda G{:}Eq\_\mathbb{N}\ k\ 0.Succ\_not\_zero\ ($
$fac\ n)G\ (y{\leq}k)))))\ Q))(\exists y{:}|\mathbb{N}|.(n{<}y\ \&\ prime(y))\ (\lambda y{:}|\mathbb{N}|.\lambda H{:}n{<}y\ \&\ y{\leq}k$
$\&\ prime(y).ExIntro(|\mathbb{N}|)\ y(\lambda w{:}|\mathbb{N}|.(n{<}w\ \&\ prime(w))(pair(n{<}y)\ ($
$prime(y))\ (fst(n{<}y)\ (y{\leq}k)(fst(n{<}y\ \&\ y{\leq}k)(prime(y))\ H))(snd(n{<}y$
$\&\ \ y{\leq}k)\ (prime(y))H)))))))))$

Пример за формално доказателство: Mizar

```
reserve n,p for Nat;

theorem Euclid: ex p st p is prime & p > n
proof
 set k = n! + 1;
 n! > 0 by NEWTON:23;
 then n! >= 0 + 1 by NAT_1:38;
 then k >= 1 + 1 by REAL_1:55;
 then consider p such that
A1: p is prime & p divides k by INT_2:48;
A2: p <> 0 & p > 1 by A1,INT_2:def 5;
 take p;
 thus p is prime by A1;
 assume p <= n;
 then p divides n! by A2,NAT_LAT:16;
 then p divides 1 by A1,NAT_1:57;
 hence contradiction by A2,NAT_1:54;
end;

theorem {p: p is prime} is infinite
 from Unbounded(Euclid);
```

Пример за формално доказателство: Isar

```
theorem Euclid: ∃ p ∈ prime. n < p
proof −
  let ?k = n! + 1
  obtain p where prime: p ∈ prime and dvd: p dvd ?k
    using prime-factor-exists by auto
  have n < p
  proof −
    have ¬ p ≤ n
    proof
      assume p ≤ n
      with prime-g-zero have p dvd n! by (rule dvd-factorial)
      with dvd have p dvd ?k − n! by (rule dvd-diff)
      then have p dvd 1 by simp
      with prime show False using prime-nd-one by auto
    qed
    then show ?thesis by simp
  qed
  from this and prime show ?thesis ..
qed

corollary ¬ finite prime
  using Euclid by (fastsimp dest!: finite-nat-set-is-bounded simp: le-def)
```

Системи за интерактивно доказване на теореми

Системите за доказване на теореми се делят на две категории:

- Системи за автоматично доказване на теореми (automated theorem provers)
- Системи за интерактивно доказване на теореми (interactive theorem provers, proof assistants)

Системи за интерактивно доказване на теореми

Системите за доказване на теореми се делят на две категории:

- Системи за автоматично доказване на теореми (automated theorem provers)
- Системи за интерактивно доказване на теореми (interactive theorem provers, proof assistants)

Изходът на система за доказване на теореми може да бъде:

- само информация за верността “вярно”, “невярно”, “не мога да определя” или просто да забива
- формално доказателство, ако теоремата е вярна
- контрапример, ако теоремата не е вярна

Системи за интерактивно доказване на теореми

Системите за доказване на теореми се делят на две категории:

- Системи за автоматично доказване на теореми (automated theorem provers)
- Системи за интерактивно доказване на теореми (interactive theorem provers, proof assistants)

Изходът на система за доказване на теореми може да бъде:

- само информация за верността “вярно”, “невярно”, “не мога да определя” или просто да забива
- формално доказателство, ако теоремата е вярна
- контрапример, ако теоремата не е вярна

Формалното доказателство служи като сертификат за верността на дадена формула.

Примери за системи за интерактивно доказване

ACL2, Agda, Coq, HOL Light, HOL4, Isabelle, LEGO, Matita, MINLOG, Mizar, NuPRL, PhoX, PVS, TPS, Twelf

Сигурност на критични системи

- Софтуер и хардуер за космически апарати

Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер

Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки

Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки
- Мрежови устройства

Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки
- Мрежови устройства
- Протоколи за сигурна комуникация

Анализ на програми

- Автоматично генериране на unit тестове

Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери

Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване

Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация

Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits

Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

Пример: Timsort

Верифицирани компилатори

- Компилатори, които са доказано коректни.

Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.

Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.
- Приложените оптимизации не променят смисъла на програмата.

Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за C програмата, то важи и за изпълнимата програма.

Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за C програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за C

Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за C програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за C
  - CompCert, проверен чрез Coq

Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за C програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за C
  - CompCert, проверен чрез Coq
  - Verified Software Toolchain

Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.

Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks

Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks
- липса на buffer overflow

Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks
- липса на buffer overflow
- липса на нежелано изтичане на информация

Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks
- липса на buffer overflow
- липса на нежелано изтичане на информация
- коректност относно дадена спецификация

Код, носещ доказателство

- Нека имаме дадена програма

Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея

Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна

Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност

Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"

Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност

Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно

Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

Код, носещ доказателство (2)

Ако злонамерен хакер се опита да промени кода, носещ доказателство:

- доказателството ще стане невалидно, тогава то ще бъде отхвърлено; или
- доказателството ще остане валидно, тогава програмата ще продължи да удовлетворява спецификацията си

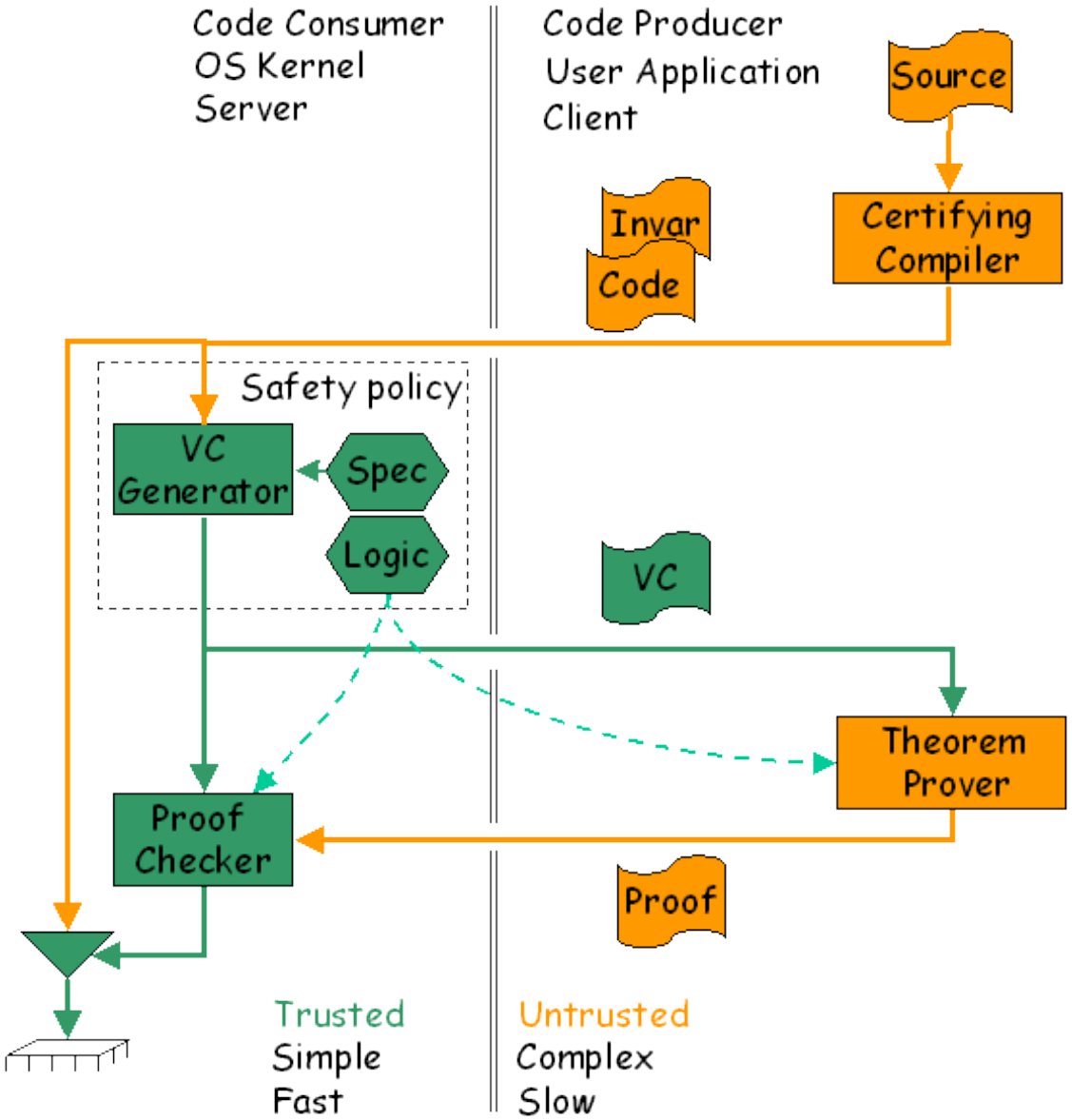
Сертифициращи компилатори

Сертифицирaщите компилатори са верифицирaщи компилатори, които генерират код, носещ доказателство.

Сертифициращи компилатори

Сертифицираща компилатори са верифициращи компилатори, които генерират код, носещ доказателство.

Вече има такъв: Touchstone, сертифициращ компилатор за Java


Code Consumer
OS Kernel
Server
Code Producer
User Application
Client
Source
Invar
Code
Certifying
Compiler
Safety policy
VC
Generator
Spec
Logic
VC
Theorem
Prover
Proof
Checker
Proof
Trusted
Simple
Fast
Untrusted
Complex
Slow