理论计算机科学导论

上海交通大学

张驰豪

2022年9月26日

可计算理论简介（cont’d）

计算问题

计算问题

计算问题：$f: \{0,1\}^* \to \{0,1\}^*$

计算问题

计算问题：$f: \{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

计算问题

计算问题：$f:\{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

$$\forall x \in \mathbb{N},\quad f((x)_2) = \begin{cases} 1, & \text{if } x \text{ is a prime;} \\ 0, & \text{otherwise .} \end{cases}$$

计算问题

计算问题：$f:\{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

$$\forall x \in \mathbb{N},\quad f((x)_2) = \begin{cases} 1, & \text{if } x \text{ is a prime;} \\ 0, & \text{otherwise .} \end{cases}$$

- 表示一个丢番图方程的解：

计算问题

计算问题：$f:\{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

$$\forall x \in \mathbb{N},\quad f((x)_2) = \begin{cases} 1, & \text{if } x \text{ is a prime;} \\ 0, & \text{otherwise .} \end{cases}$$

- 表示一个丢番图方程的解：

$$\forall \text{ 丢番图方程的编码 } x,\quad f(x) = \text{ 方程解的编码}$$

计算问题

计算问题：$f:\{0,1\}^* \to \{0,1\}^*$

可等价表示成 $f:\mathbb{N} \to \mathbb{N}$

- 表示一个数是不是素数：

$$\forall x \in \mathbb{N},\quad f((x)_2) = \begin{cases} 1, & \text{if } x \text{ is a prime;} \\ 0, & \text{otherwise .} \end{cases}$$

- 表示一个丢番图方程的解：

$$\forall \text{ 丢番图方程的编码 } x,\quad f(x) = \text{ 方程解的编码}$$

丘奇-图灵论题

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

$\Leftrightarrow$

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

$\iff$ 存在图灵机 $T: \forall x, T(x) = f(x)$

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

$\iff$ 存在图灵机 $T: \forall x, T(x) = f(x)$

$\iff$

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

$\iff$ 存在图灵机 $T: \forall x, T(x) = f(x)$

$\iff$ 存在C++程序 $P: \forall x, P(x) = f(x)$

图灵机

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的“套路”＋草稿纸

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的“套路”＋草稿纸

一个k−带图灵机是一个三元组 (Γ, Q, δ):

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的“套路”＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$

当前输入

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$$

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$

当前输入

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$$

当前算到哪了

图灵机

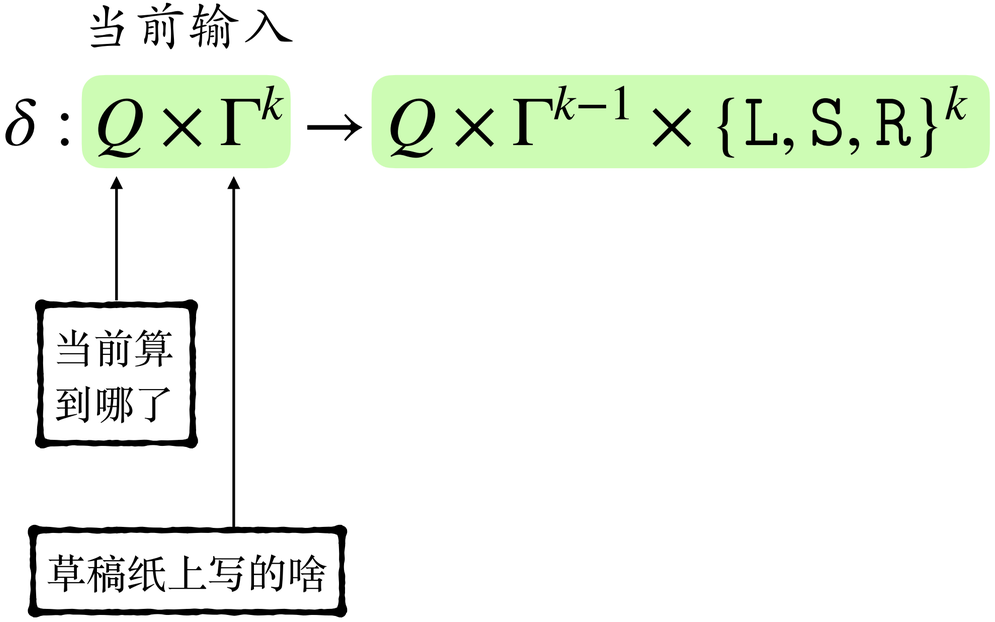
> For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

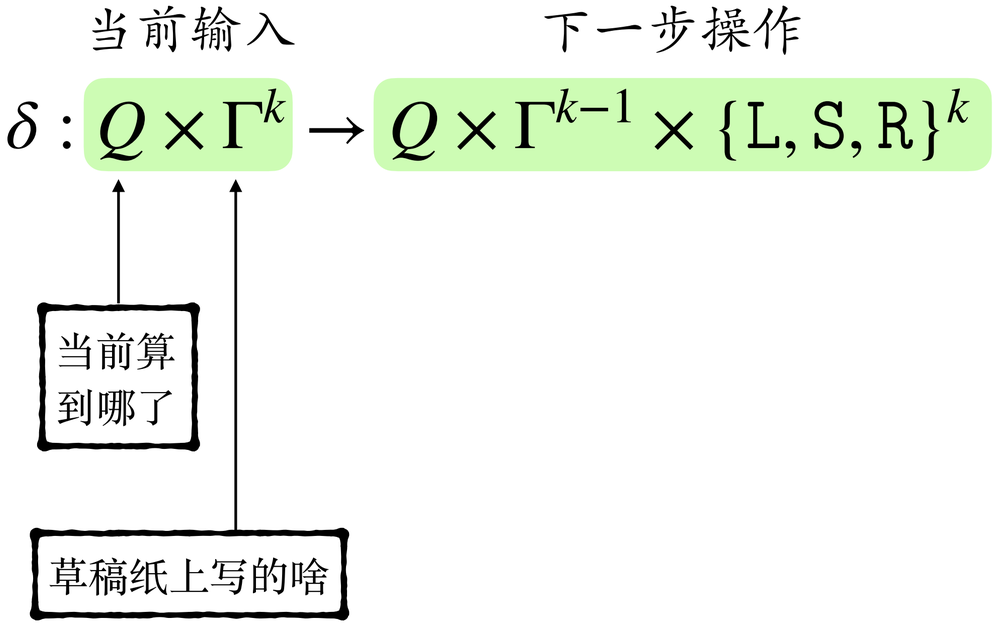
For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

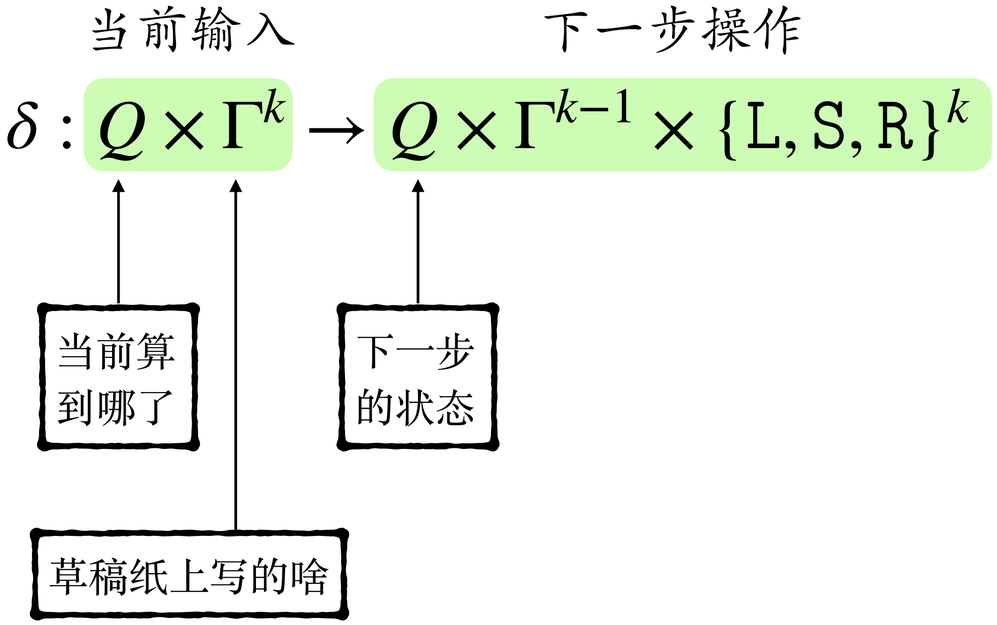
> For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

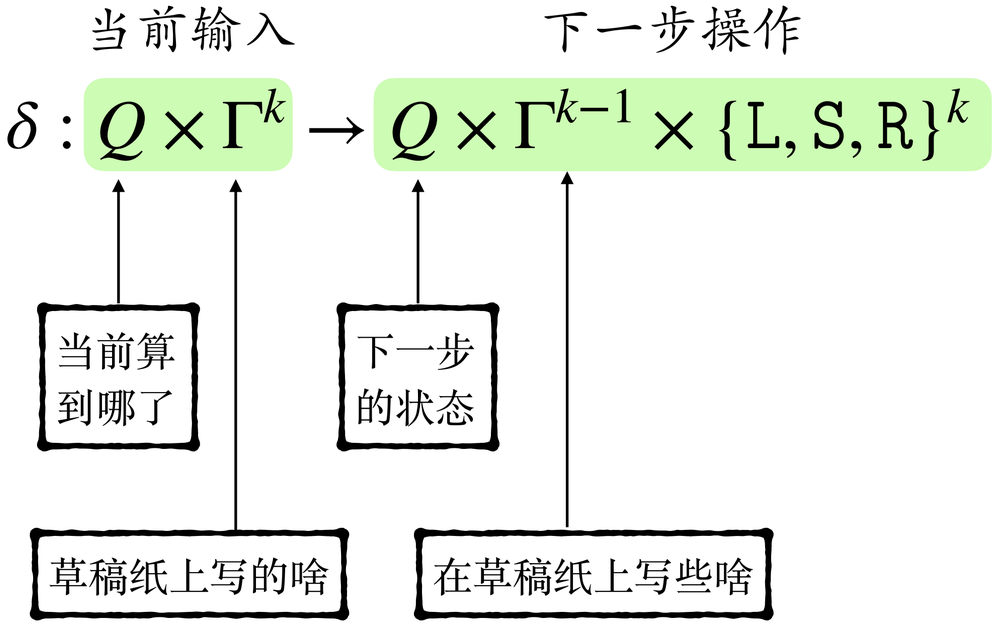
> For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

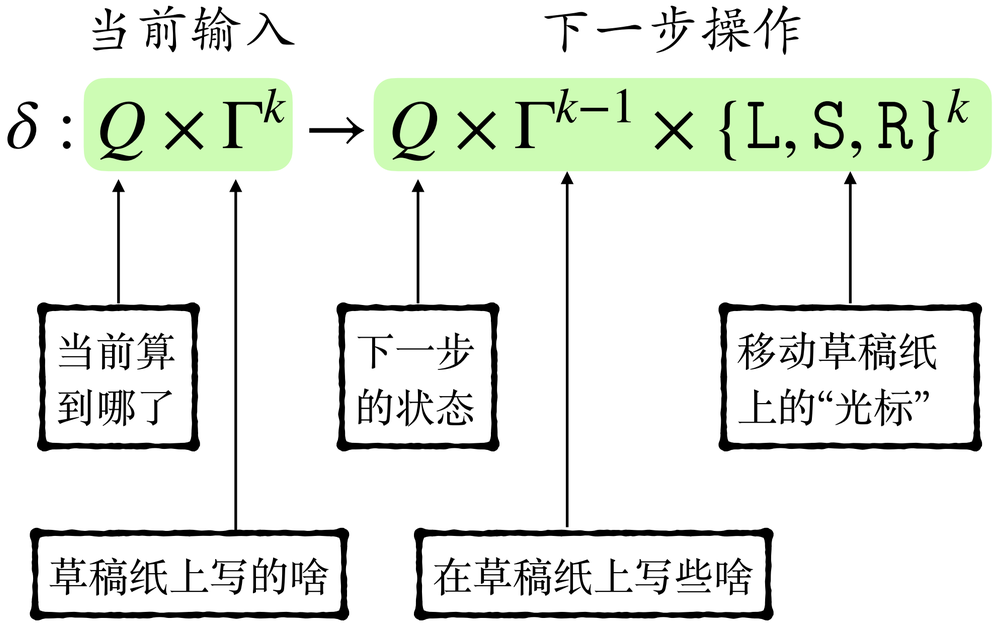
> For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

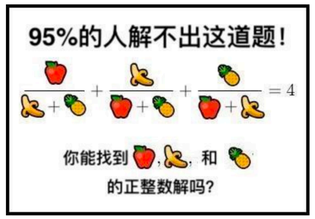
- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



函数的可计算性

函数的可计算性

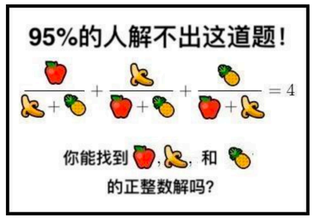
什么是算法



你能写一个程序/算法解这类问题吗?

函数的可计算性

什么是算法



= 154476802108746166441951315019919837485664325669656317006266348982532020352779993

= 368751317941299998271978115652254748254929799689719709962831374716372246340555793

= 437361267792869725786125260237139015281653755816161638186214379933784234677720363

你能写一个程序/算法解这类问题吗?

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

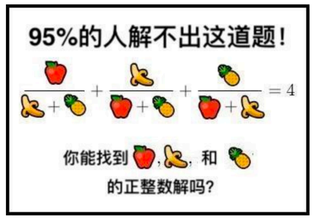
https://en.wikipedia.org/wiki/Entscheidungsproblem

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

函数的可计算性

什么是算法



你能写一个程序/算法解这类问题吗?

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

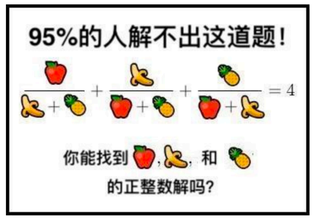
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

问题：每一个函数都是可计算的吗?

函数的可计算性

什么是算法



你能写一个程序/算法解这类问题吗?

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

问题：每一个函数都是可计算的吗?

几乎所有的函数都是不可计算的：函数的个数是不可数的，不同程序的个数是可数的。

题外话：如何定义比较无穷集合的大小?

题外话：如何定义比较无穷集合的大小?

$\mathbb{N}$: 自然数集

$\mathbb{Q}$: 有理数集

$\mathbb{R}$: 实数集

题外话：如何定义比较无穷集合的大小?

$\mathbb{N}$: 自然数集

$\mathbb{Q}$: 有理数集

$\mathbb{R}$: 实数集

哪个集合最大?

题外话：如何定义比较无穷集合的大小?

$\mathbb{N}$: 自然数集

$\mathbb{Q}$: 有理数集

$\mathbb{R}$: 实数集

考虑映射 $f: A \to B$

哪个集合最大?

题外话：如何定义比较无穷集合的大小?

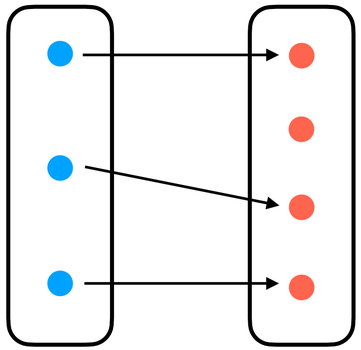
$\mathbb{N}$: 自然数集

$\mathbb{Q}$: 有理数集

$\mathbb{R}$: 实数集

哪个集合最大?

考虑映射 $f: A \to B$



单射

题外话：如何定义比较无穷集合的大小?

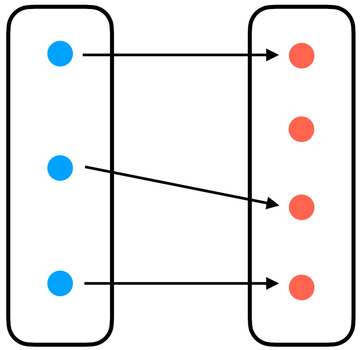
$\mathbb{N}$: 自然数集

$\mathbb{Q}$: 有理数集

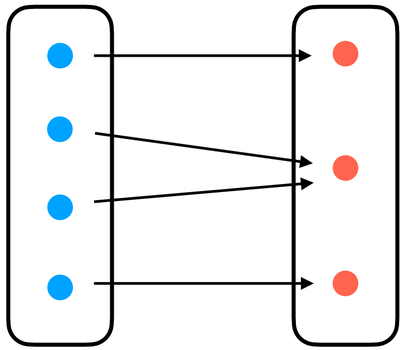
$\mathbb{R}$: 实数集

哪个集合最大?

考虑映射 $f: A \to B$



单射



满射

题外话：如何定义比较无穷集合的大小?

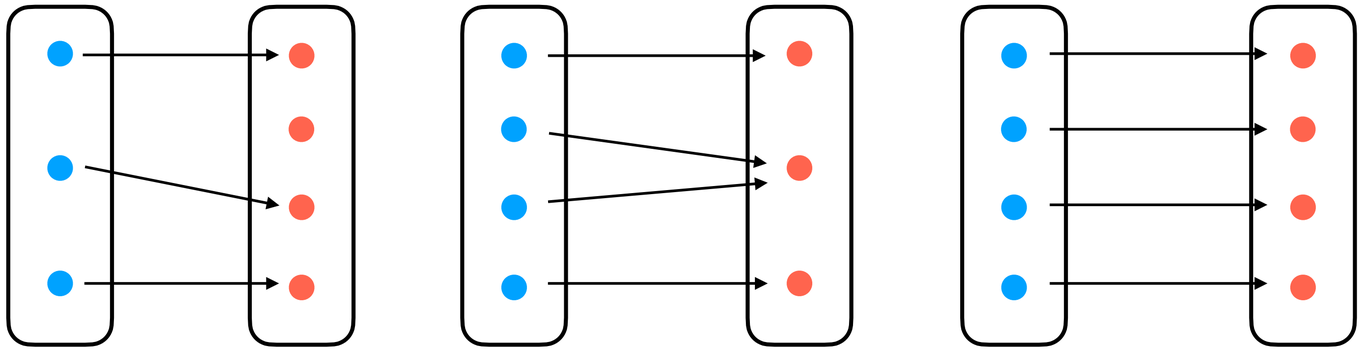
$\mathbb{N}$: 自然数集

$\mathbb{Q}$: 有理数集

$\mathbb{R}$: 实数集

哪个集合最大?

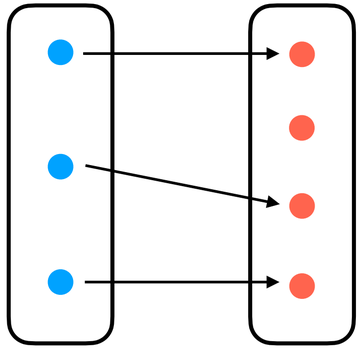
考虑映射 $f: A \to B$



单射　　满射　　双射

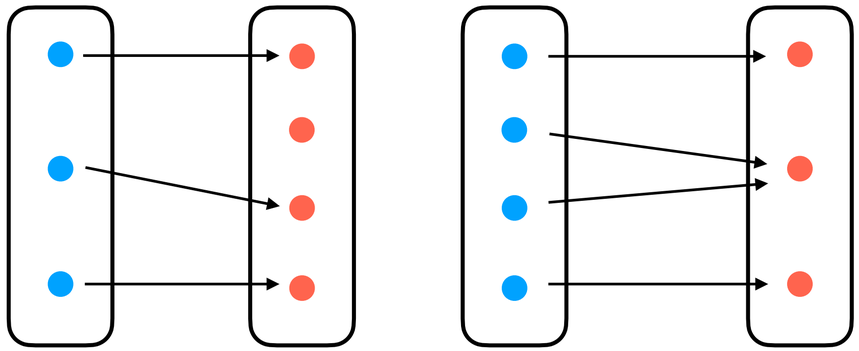
考虑映射 $f: A \to B$

考虑映射 $f: A \to B$



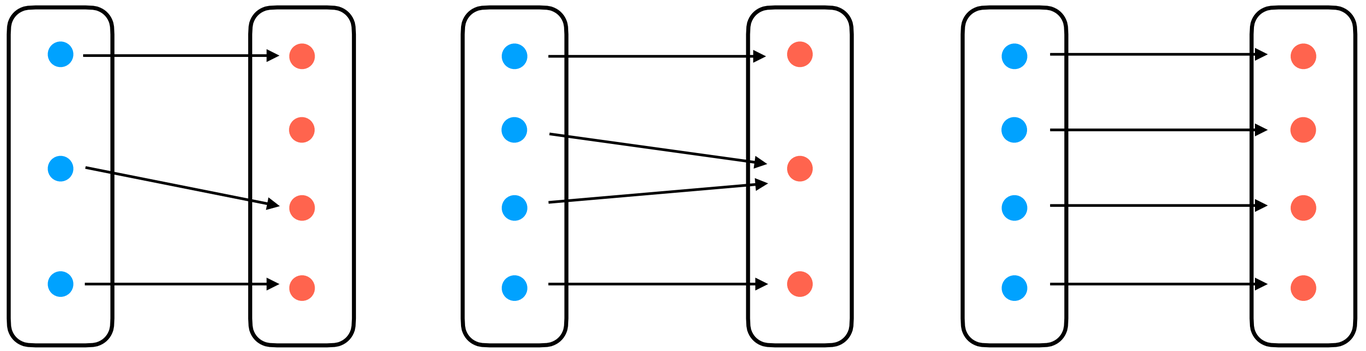
单射

考虑映射 $f: A \to B$



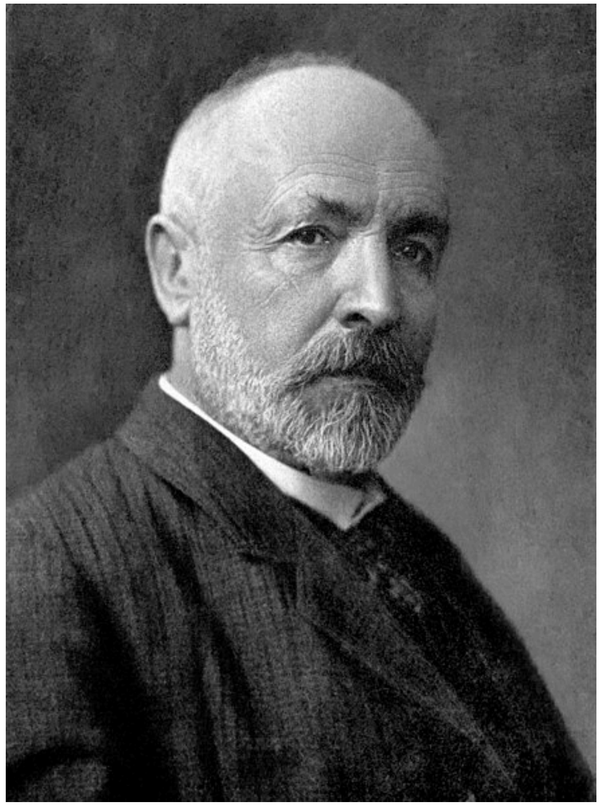
单射　　满射

考虑映射 $f: A \to B$



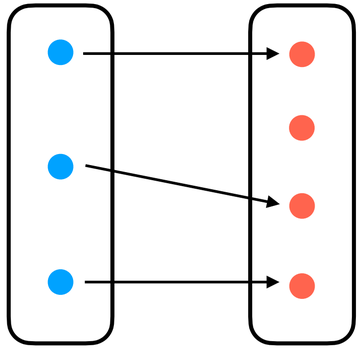
单射　　满射　　双射

考虑映射 $f: A \to B$

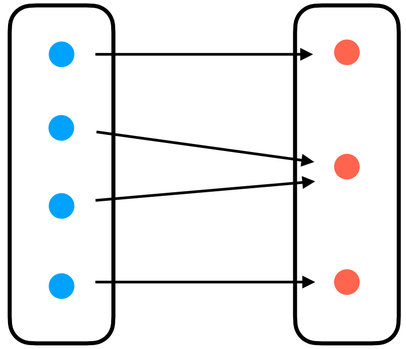


Georg Cantor

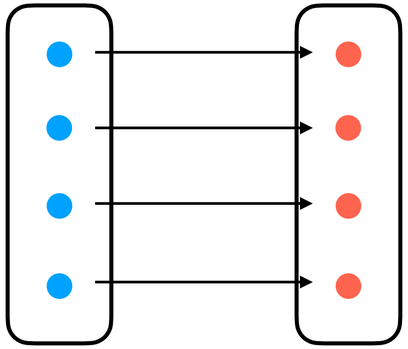
From: https://en.wikipedia.org/wiki/Georg_Cantor



单射

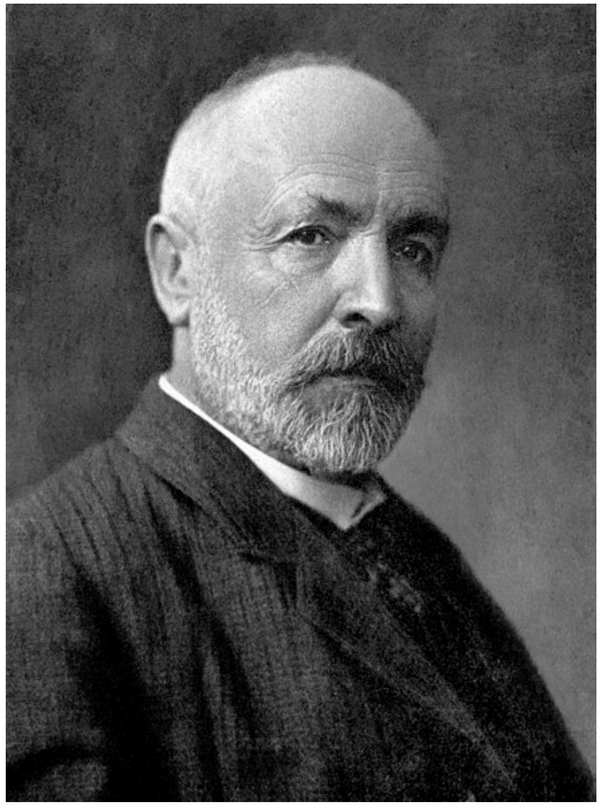


满射



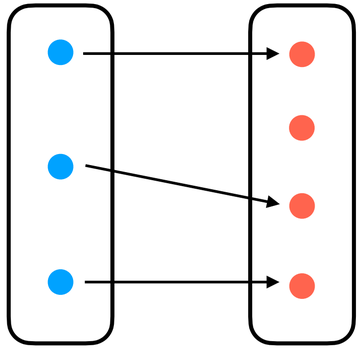
双射

考虑映射 $f: A \to B$

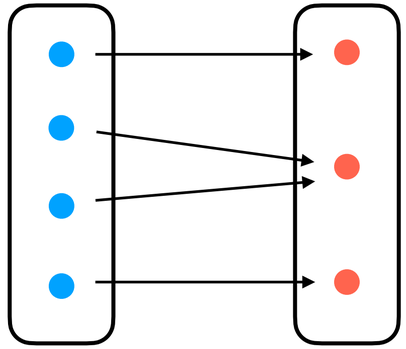


Georg Cantor

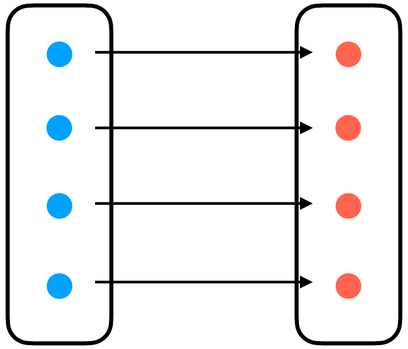
From: https://en.wikipedia.org/wiki/Georg_Cantor



单射



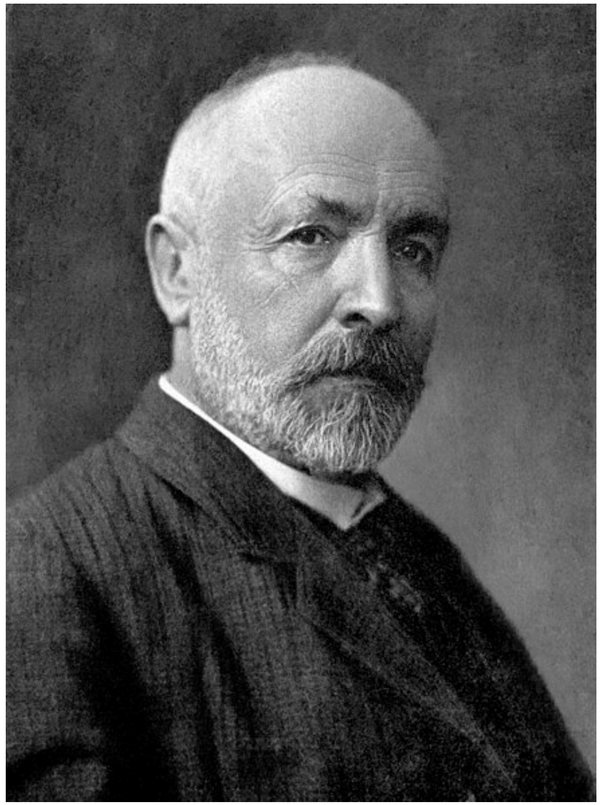
满射



双射

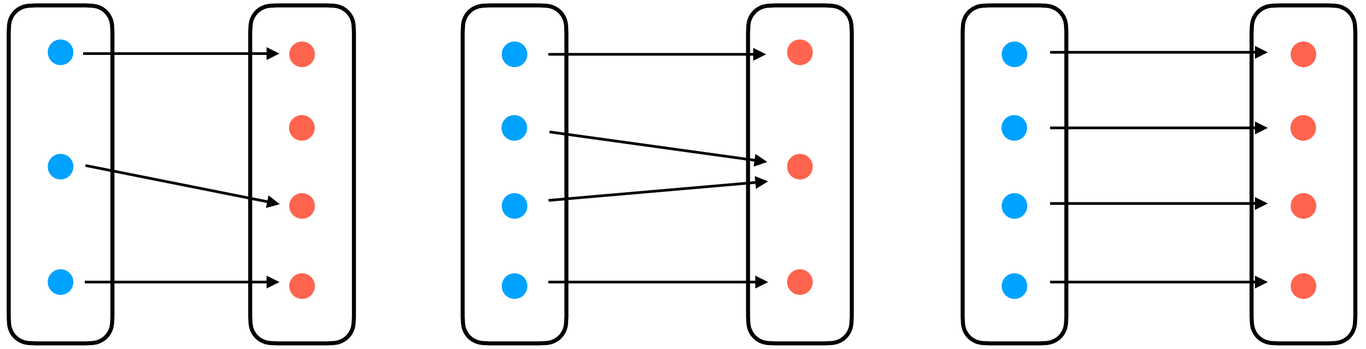
- 如果存在单射 $f: A \to B$，则 $|A| \leq |B|$

考虑映射 $f: A \to B$



Georg Cantor

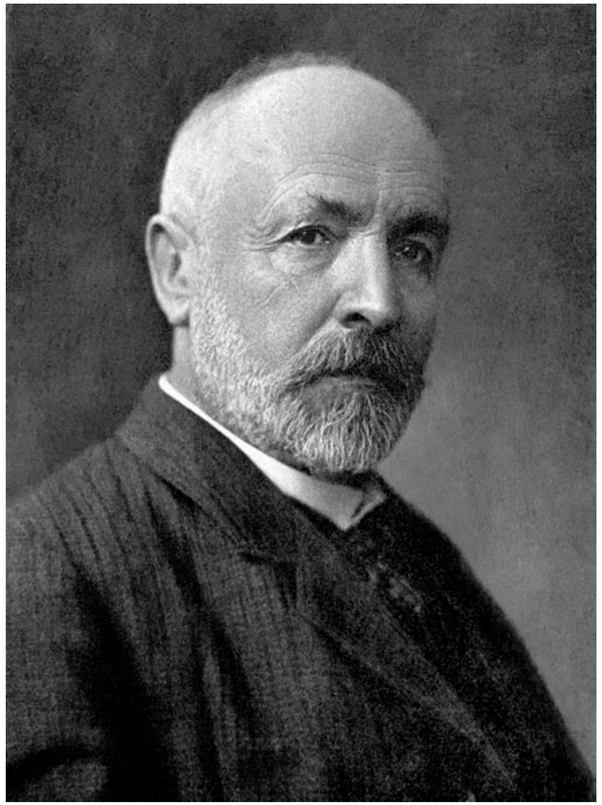
From: https://en.wikipedia.org/wiki/Georg_Cantor



单射　　满射　　双射

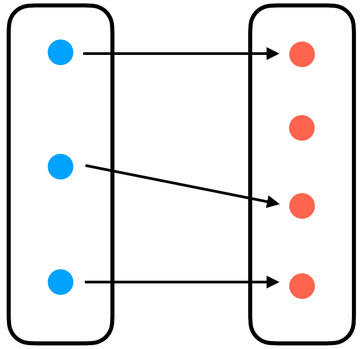
- 如果存在单射 $f: A \to B$，则 $|A| \leq |B|$
- 如果存在满射 $f: A \to B$，则 $|A| \geq |B|$

考虑映射 $f: A \to B$

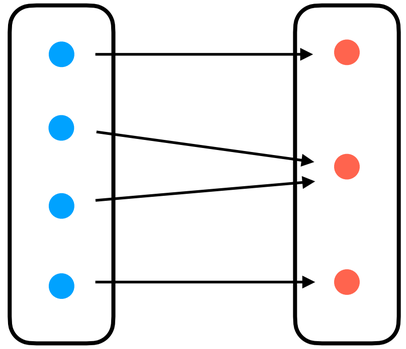


Georg Cantor

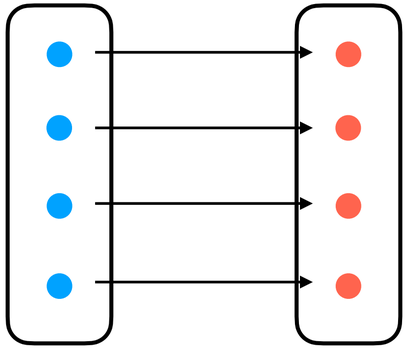
From: https://en.wikipedia.org/wiki/Georg_Cantor



单射



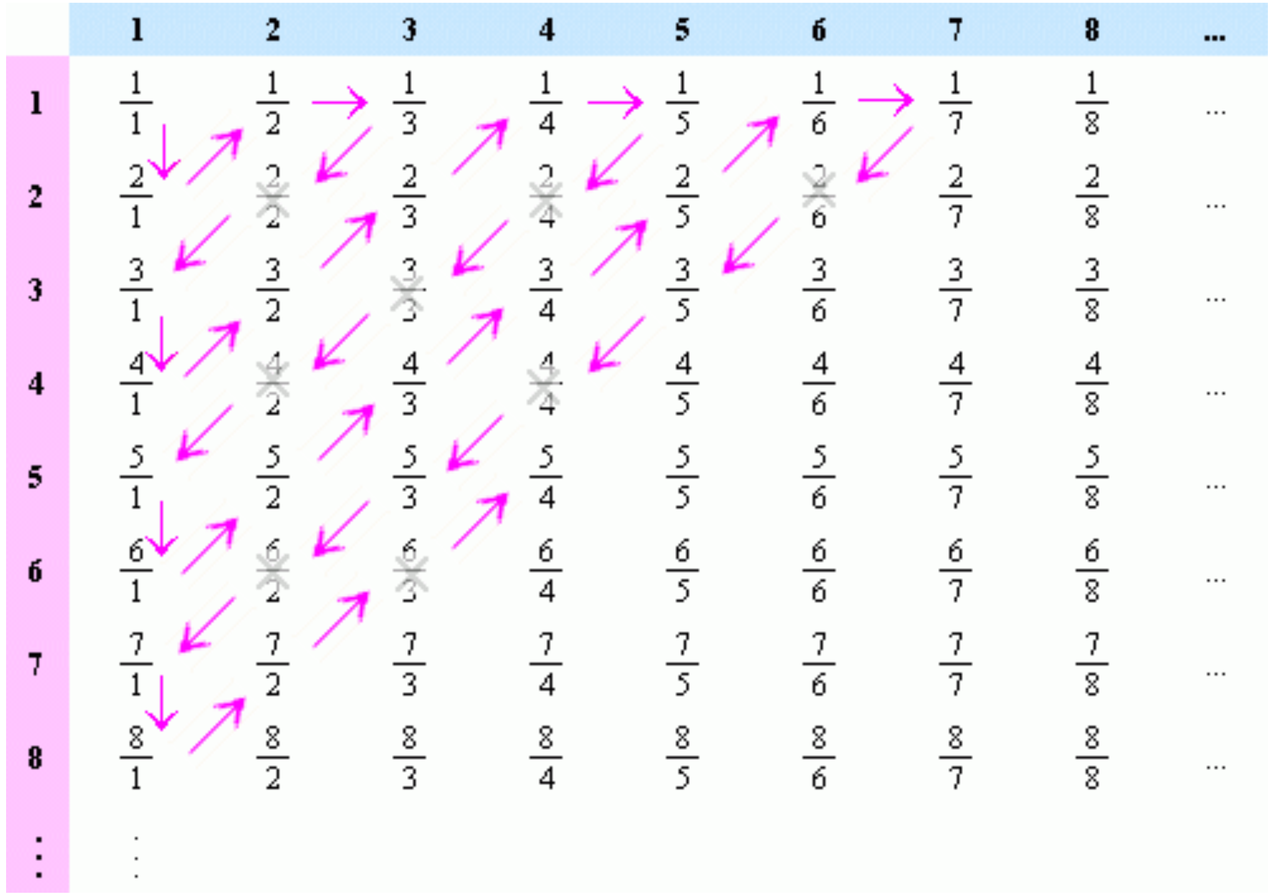
满射



双射

- 如果存在单射 $f: A \to B$，则 $|A| \leq |B|$
- 如果存在满射 $f: A \to B$，则 $|A| \geq |B|$
- 如果存在双射 $f: A \to B$，则 $|A| = |B|$

$|\mathbb{Q}| = |\mathbb{N}|$



https://www.homeschoolmath.net/teaching/rational-numbers-countable.php

对角线方法（Diagonalization）

对角线方法（Diagonalization）

定理：$|\mathbb{N}| < |\mathbb{R}|$

对角线方法（Diagonalization）

定理：$|\mathbb{N}| < |\mathbb{R}|$

假设存在$\mathbb{N} \to [0,1]$的双射f...

对角线方法（Diagonalization）

定理：$|\mathbb{N}| < |\mathbb{R}|$

假设存在$\mathbb{N} \to [0,1]$的双射f…

$$
\begin{aligned}
1 &\mapsto 0.r_{11}r_{12}r_{13}\dots \\
2 &\mapsto 0.r_{21}r_{22}r_{23}\dots \\
&\vdots \\
j &\mapsto 0.r_{j1}r_{j2}\dots
\end{aligned}
$$

对角线方法（Diagonalization）

定理：$|\mathbb{N}| < |\mathbb{R}|$

假设存在$\mathbb{N} \to [0,1]$的双射f…

$$
\begin{aligned}
1 &\mapsto 0.r_{11}r_{12}r_{13}\dots \\
2 &\mapsto 0.r_{21}r_{22}r_{23}\dots \\
&\vdots \\
j &\mapsto 0.r_{j1}r_{j2}\dots
\end{aligned}
$$

$r_{jk} \in \{0,1,\dots,9\}$

对角线方法（Diagonalization）

定理：$|\mathbb{N}| < |\mathbb{R}|$

假设存在$\mathbb{N} \to [0,1]$的双射f…

$1 \mapsto 0.r_{11}r_{12}r_{13}\cdots$
$2 \mapsto 0.r_{21}r_{22}r_{23}\cdots$
$\vdots$
$j \mapsto 0.r_{j1}r_{j2}\cdots$

$r_{jk} \in \{0,1,\ldots,9\}$

	1	2	3	4	…
1	r_{11}	r_{12}	r_{13}	r_{14}	⋯
2	r_{21}	r_{22}	r_{23}	r_{24}	⋯
3	r_{31}	r_{32}	r_{33}	r_{34}	⋯
⋮	⋮	⋮	⋮	⋮	
j	r_{j1}	r_{j2}	r_{j3}	r_{j4}	⋯
⋮					

考察实数 $x = 0.r_1r_2r_3\ldots$满足

考察实数 $x = 0.r_1 r_2 r_3 \ldots$ 满足

$$\text{任意 } j \in \mathbb{N},\quad r_j = \begin{cases} 6, & \text{if } r_{jj} = 7, \\ 7, & \text{if } r_{jj} \neq 7. \end{cases}$$

考察实数 $x = 0.r_1r_2r_3\ldots$满足

$$\text{任意 } j \in \mathbb{N},\quad r_j = \begin{cases} 6, & \text{if } r_{jj} = 7, \\ 7, & \text{if } r_{jj} \neq 7. \end{cases}$$

则任意$j \in \mathbb{N}, f(j) \neq x$

考察实数 $x = 0.r_1 r_2 r_3 \ldots$ 满足

$$\text{任意 } j \in \mathbb{N},\quad r_j = \begin{cases} 6, & \text{if } r_{jj} = 7, \\ 7, & \text{if } r_{jj} \neq 7. \end{cases}$$

则任意$j \in \mathbb{N}, f(j) \neq x$

	1	2	3	4	…
1	r_{11}	r_{12}	r_{13}	r_{14}	…
2	r_{21}	r_{22}	r_{23}	r_{24}	…
3	r_{31}	r_{32}	r_{33}	r_{34}	…
⋮	⋮	⋮	⋮	⋮	
j	r_{j1}	r_{j2}	r_{j3}	r_{j4}	…r_{jj}
⋮					

考察实数 $x = 0.r_1 r_2 r_3 \ldots$ 满足

$$\text{任意 } j \in \mathbb{N},\quad r_j = \begin{cases} 6, & \text{if } r_{jj} = 7, \\ 7, & \text{if } r_{jj} \neq 7. \end{cases}$$

则任意$j \in \mathbb{N}, f(j) \neq x$

与 f 是双射矛盾！

	1	2	3	4	…
1	r_{11}	r_{12}	r_{13}	r_{14}	…
2	r_{21}	r_{22}	r_{23}	r_{24}	…
3	r_{31}	r_{32}	r_{33}	r_{34}	…
⋮	⋮	⋮	⋮	⋮	
j	r_{j1}	r_{j2}	r_{j3}	r_{j4}	…r_{jj}
⋮					

考察实数 $x = 0.r_1 r_2 r_3 \ldots$ 满足

$$\text{任意 } j \in \mathbb{N},\quad r_j = \begin{cases} 6, & \text{if } r_{jj} = 7, \\ 7, & \text{if } r_{jj} \neq 7. \end{cases}$$

则任意$j \in \mathbb{N}, f(j) \neq x$

与 f 是双射矛盾！

	1	2	3	4	…
1	r_{11}	r_{12}	r_{13}	r_{14}	…
2	r_{21}	r_{22}	r_{23}	r_{24}	…
3	r_{31}	r_{32}	r_{33}	r_{34}	…
⋮	⋮	⋮	⋮	⋮	
j	r_{j1}	r_{j2}	r_{j3}	r_{j4}	…r_{jj}
⋮					

$\mathbb{N}$: 可数无穷大

任何比$\mathbb{N}$大的无穷集合：不可数无穷大

停机问题（Halting Problem）

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

考虑如下函数：

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

考虑如下函数：

$$\texttt{Halt}(x, y) = \begin{cases} 0, & \text{如果 } P_x(y) \text{有死循环;} \\ 1, & \text{如果 } P_x(y) \text{没有死循环.} \end{cases}$$

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

考虑如下函数：

$$\texttt{Halt}(x, y) = \begin{cases} 0, & \text{如果 } P_x(y) \text{有死循环;} \\ 1, & \text{如果 } P_x(y) \text{没有死循环.} \end{cases}$$

定理：Halt 函数是不可计算的。

停机问题不可计算的证明

停机问题不可计算的证明

对角线方法！

停机问题不可计算的证明

对角线方法！

假设停机问题可以计算，即存
在程序 P: $P(x, y) = \texttt{Halt}(x, y)$

停机问题不可计算的证明

对角线方法！

假设停机问题可以计算，即存
在程序 P: $P(x, y) = \texttt{Halt}(x, y)$

把 $\texttt{Halt}(x, y)$ 写成表格

停机问题不可计算的证明

对角线方法！

假设停机问题可以计算，即存在程序 P: $P(x, y) = \texttt{Halt}(x, y)$

把 $\texttt{Halt}(x, y)$ 写成表格

	1	2	3	4	…
1	h_{11}	h_{12}	h_{13}	h_{14}	…
2	h_{21}	h_{22}	h_{23}	h_{24}	…
3	h_{31}	h_{32}	h_{33}	h_{34}	…
⋮	⋮	⋮	⋮	⋮	
x	h_{x1}	h_{x2}	h_{x3}	h_{x4}	…
⋮					

停机问题不可计算的证明

对角线方法！

假设停机问题可以计算，即存在程序 P: $P(x, y) = \texttt{Halt}(x, y)$

把 $\texttt{Halt}(x, y)$ 写成表格

$h_{xy} = \texttt{Halt}(x, y)$

	1	2	3	4	…
1	h_{11}	h_{12}	h_{13}	h_{14}	…
2	h_{21}	h_{22}	h_{23}	h_{24}	…
3	h_{31}	h_{32}	h_{33}	h_{34}	…
⋮	⋮	⋮	⋮	⋮	
x	h_{x1}	h_{x2}	h_{x3}	h_{x4}	…
⋮					

可以构造程序 B 满足

可以构造程序 B 满足

任意 $x \in \mathbb{N}$, $h_{x(B),x} = 1 - h_{x,x}$

可以构造程序 B 满足

任意 $x \in \mathbb{N}$, $h_{x(B),x} = 1 - h_{x,x}$

则任意 $x \in \mathbb{N}$, $P_x \neq B$

可以构造程序 B 满足

任意 $x \in \mathbb{N}$, $h_{x(B),x} = 1 - h_{x,x}$

则任意 $x \in \mathbb{N}$, $P_x \neq B$

	1	2	3	4	…	x
1	h_{11}	h_{12}	h_{13}	h_{14}	…	
2	h_{21}	h_{22}	h_{23}	h_{24}	…	
3	h_{31}	h_{32}	h_{33}	h_{34}	…	
⋮	⋮	⋮	⋮	⋮		
x	h_{x1}	h_{x2}	h_{x3}	h_{x4}	…	h_{xx}
⋮						

可以构造程序 B 满足

任意 $x \in \mathbb{N}$, $h_{x(B),x} = 1 - h_{x,x}$

则任意 $x \in \mathbb{N}$, $P_x \neq B$

```
Program B:
int main(int x) {
    If P(x,x)=0 then
        return 1;
    else
        for(;;); // loop forever
}
```

	1	2	3	4	…	x
1	h_{11}	h_{12}	h_{13}	h_{14}	…	
2	h_{21}	h_{22}	h_{23}	h_{24}	…	
3	h_{31}	h_{32}	h_{33}	h_{34}	…	
⋮	⋮	⋮	⋮	⋮		
x	h_{x1}	h_{x2}	h_{x3}	h_{x4}	…	h_{xx}
⋮						

归约（Reduction）

一名数学家去应聘消防员

面试官：“街边一栋房子着火了怎么办？”

数学家讲了如何灭火。

消防员：“那房子没着火呢？”

数学家：把房子点着

一名数学家去应聘消防员

面试官："街边一栋房子着火了怎么办？"

数学家讲了如何灭火。

消防员："那房子没着火呢？"

数学家：把房子点着

设 $f, g : \mathbb{N} \to \mathbb{N}$ 为两个计算问题

一名数学家去应聘消防员

面试官："街边一栋房子着火了怎么办？"

数学家讲了如何灭火。

消防员："那房子没着火呢？"

数学家：把房子点着

设 $f, g : \mathbb{N} \to \mathbb{N}$ 为两个计算问题

"f 可（图灵）归约至 g"，记作 $f \leq_T g$

一名数学家去应聘消防员

面试官："街边一栋房子着火了怎么办？"

数学家讲了如何灭火。

消防员："那房子没着火呢？"

数学家：把房子点着

设 $f, g : \mathbb{N} \to \mathbb{N}$ 为两个计算问题

"f 可（图灵）归约至 g"，记作 $f \leq_T g$

如果有一个程序 B 能计算 g，则存在一个允许使用 B 作子程序的程序 A 来计算 f

假设$f \leq_T g$，则：

假设 $f \leq_T g$，则：

- 如果 g 可计算，则 f 可计算

假设 $f \leq_T g$，则：

- 如果 g 可计算，则 f 可计算
- 如果 f 不可计算，则 g 不可计算

假设 $f \leq_T g$，则：

- 如果 g 可计算，则 f 可计算
- 如果 f 不可计算，则 g 不可计算

“f 的难度 $\leq$ g 的难度”

假设 $f \leq_T g$，则：

- 如果 g 可计算，则 f 可计算
- 如果 f 不可计算，则 g 不可计算

“f 的难度 $\leq$ g 的难度”

例：证明函数

$$\texttt{TMEQ}(x,y) := \begin{cases} 1, & \text{任意} z \in \mathbb{N}, P_x(z) = P_y(z), \\ 0, & \text{otherwise.} \end{cases}$$

是不可计算的。

假设 $f \leq_T g$，则：

- 如果 g 可计算，则 f 可计算
- 如果 f 不可计算，则 g 不可计算

“f 的难度 $\leq$ g 的难度”

例：证明函数

$$\texttt{TMEQ}(x,y) := \begin{cases} 1, & \text{任意} z \in \mathbb{N}, P_x(z) = P_y(z), \\ 0, & \text{otherwise.} \end{cases}$$

是不可计算的。

从停机问题归约，即证明

$\texttt{Halt} \leq_T \texttt{TMEQ}$

假设有一个程序 B 能够解 TMEQ，如何使用它来解 Halt？

假设有一个程序 B 能够解 TMEQ，如何使用它来解 Halt？

- 给定输入 $x, y \in \mathbb{N}$，目标是判定 $P_x(y)$是否有死循环

假设有一个程序 B 能够解 TMEQ，如何使用它来解 Halt？

- 给定输入 $x, y \in \mathbb{N}$，目标是判定 $P_x(y)$是否有死循环
- 与以往不同，我们具有“超能力”，即判定两个程序是否等价

假设有一个程序 B 能够解 TMEQ，如何使用它来解 Halt？

- 给定输入 $x, y \in \mathbb{N}$，目标是判定 $P_x(y)$是否有死循环
- 与以往不同，我们具有“超能力”，即判定两个程序是否等价
- 构造程序P_1满足对于任意输入$z \in \mathbb{N}$, $P_1(z) = P_x(y)$

假设有一个程序 B 能够解 TMEQ，如何使用它来解 Halt？

- 给定输入 $x, y \in \mathbb{N}$，目标是判定 $P_x(y)$是否有死循环
- 与以往不同，我们具有“超能力”，即判定两个程序是否等价
- 构造程序P_1满足对于任意输入$z \in \mathbb{N}$, $P_1(z) = P_x(y)$
- 构造程序P_2满足对于任意输入$z \in \mathbb{N}$, $P_2(z)$均死循环

假设有一个程序 B 能够解 TMEQ，如何使用它来解 Halt？

- 给定输入 $x, y \in \mathbb{N}$，目标是判定 $P_x(y)$是否有死循环
- 与以往不同，我们具有“超能力”，即判定两个程序是否等价
- 构造程序P_1满足对于任意输入$z \in \mathbb{N}$, $P_1(z) = P_x(y)$
- 构造程序P_2满足对于任意输入$z \in \mathbb{N}$, $P_2(z)$均死循环
- 输出 $B(x(P_1), x(P_2))$

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

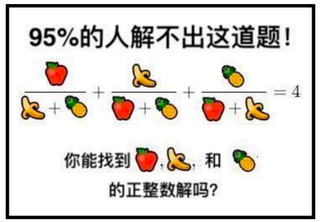
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

什么是算法



你能写一个程序/算法
解这类问题吗？

🍎 = 154476802108746166441951315019919837485664325669565431700026634898253202035277999
🍌 = 36875131794129999827197811565225474825492979968971970996283137471637224634055579
🍊 = 43736126779286972578612526023713901528165375581616136186214379933784234677720361

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

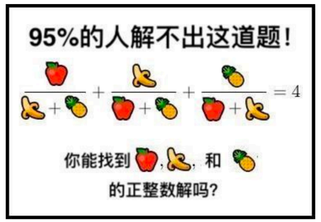
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

什么是算法



你能写一个程序/算法解这类问题吗?

= 154476802108746166441951315019919837485664325669565431700026634989253202035277999

= 366751317941299998271978115652254748254929799689719709962831374716372246340555579

= 4373612677926869725786125260237139015281655755816161361862143799337842346772036

希尔伯特第十问题：是否存在算法判断丢番图方程是否有解?

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

https://en.wikipedia.org/wiki/Entscheidungsproblem

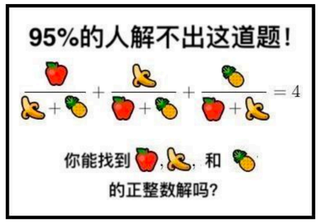
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

什么是算法



你能写一个程序/算法解这类问题吗?

= 154476802108746166441951315019919837485664325669565431700026634898253202035277999

= 36875131794129999827197811565225474825492979968971970996283137471637224634055579

= 4373612677926869725786125260237139015281655755816161361862143799337842346772036

希尔伯特第十问题：是否存在算法判断丢番图方程是否有解?

Matiyasevich 定理（1970）：不存在求解丢番图方程的通用算法。

图灵归约定义了一种计算难度上的等价关系：

图灵归约定义了一种计算难度上的等价关系：

- $f \equiv_T g$ 当且仅当 $f \leq_T g$ 且 $g \leq_T f$

图灵归约定义了一种计算难度上的等价关系：

- $f \equiv_T g$ 当且仅当 $f \leq_T g$ 且 $g \leq_T f$

计算难度等价类的结构？

图灵归约定义了一种计算难度上的等价关系：

- $f \equiv_T g$ 当且仅当 $f \leq_T g$ 且 $g \leq_T f$

计算难度等价类的结构？

假设拥有解决停机问题的超能力，是否能解决所有问题？

图灵归约定义了一种计算难度上的等价关系：

- $f \equiv_T g$ 当且仅当 $f \leq_T g$ 且 $g \leq_T f$

计算难度等价类的结构？

假设拥有解决停机问题的超能力，是否能解决所有问题？

否（Counting Argument）

图灵度

图灵度

- 给定一个具体问题 f，等价类 $[f]$ 为其图灵度

图灵度

- 给定一个具体问题 f，等价类 $[f]$ 为其图灵度
- 图灵度上具有偏序关系：$[f] \leq [g]$ iff $f \leq_T g$

图灵度

- 给定一个具体问题 f，等价类 $[f]$ 为其图灵度
- 图灵度上具有偏序关系：$[f] \leq [g]$ iff $f \leq_T g$
- 设 f 为可计算问题，我们知道 $[f] <_T [\mathtt{Halt}]$

图灵度

- 给定一个具体问题 f，等价类 $[f]$ 为其图灵度
- 图灵度上具有偏序关系：$[f] \leq [g]$ iff $f \leq_T g$
- 设 f 为可计算问题，我们知道 $[f] <_T [\texttt{Halt}]$

(Friedberg and Muchnik (1965)) 存在 $[f]$ 与 $[\texttt{Halt}]$ 之间的图灵度

计算问题分类

计算问题分类

计算问题分类

- 可计算理论/递归论：研究不可计算函数

计算问题分类

- 可计算理论/递归论：研究不可计算函数

这个问题能不能计算？

计算问题分类

- 可计算理论/递归论：研究不可计算函数

这个问题能不能计算？

图灵度的结构

计算问题分类

- 可计算理论/递归论：研究不可计算函数

  这个问题能不能计算？

  图灵度的结构

- 计算复杂性/算法：研究可计算函数

计算问题分类

- 可计算理论/递归论：研究不可计算函数

  这个问题能不能计算？

  图灵度的结构

- 计算复杂性/算法：研究可计算函数

  这个问题能多快计算？

计算复杂性

计算复杂性

考察可计算函数 $f: \{0,1\}^* \to \{0,1\}^*$

计算复杂性

考察可计算函数 $f: \{0,1\}^* \to \{0,1\}^*$

- 设 P是计算 f 的一个程序，$\mathrm{T}_P: \mathbb{N} \to \mathbb{N}$ 为一个函数 （如n^2，2^n等）

计算复杂性

考察可计算函数 $f: \{0,1\}^* \to \{0,1\}^*$

- 设 P是计算 f 的一个程序，$\mathrm{T}_P: \mathbb{N} \to \mathbb{N}$ 为一个函数 （如n^2，2^n等）

- f 在 P 上的运行时间不超过 T_P 指的是对于任意 x，$P(x)$在 $\mathrm{T}_P(|x|)$步内停止并输出

计算复杂性

考察可计算函数 $f: \{0,1\}^* \to \{0,1\}^*$

- 设 P是计算 f 的一个程序，$\mathrm{T}_P: \mathbb{N} \to \mathbb{N}$ 为一个函数 （如n^2，2^n等）
- f 在 P 上的运行时间不超过 T_P 指的是对于任意 x，$P(x)$在 $\mathrm{T}_P(|x|)$步内停止并输出
- f 的计算复杂性（Informal）： $\min\{T_P \mid P \text{ computes } f\}$

重见对角线方法：时间分层定理

重见对角线方法：时间分层定理

时间分层定理：如果 $T_1(n) \ll \dfrac{T_2(n)}{\log T_1(n)}$，则存在可在 $T_2(n)$ 内计算但无法在 $T_1(n)$ 内计算的函数

重见对角线方法：时间分层定理

时间分层定理：如果 $T_1(n) \ll \frac{T_2(n)}{\log T_1(n)}$，则存在可在 $T_2(n)$ 内计算但无法在 $T_1(n)$ 内计算的函数

证明：对角线方法：对于任何 $x \in \mathbb{N}$，高效的模拟 $P_x(x)$。