колмогоровская сложность

колмогоровская сложность

- интерпретатор (декомпрессор, способ описания): алгоритм вход и выход — двоичные слова (строки)
- для данного интерпретатора I:

$$K_I(x) = \min\{|p| : I(p) = x\} \quad [\min(\varnothing) = +\infty]$$

колмогоровская сложность

- интерпретатор (декомпрессор, способ описания): алгоритм вход и выход — двоичные слова (строки)
- для данного интерпретатора I:

$$K_I(x) = \min\{|p| : I(p) = x\} \quad [\min(\varnothing) = +\infty]$$

- сравнение: I не хуже I', если

$$\exists c \, \forall x \, [K_I(x) \leq K_{I'}(x) + c]$$

- теорема оптимальности Соломонова – Колмогорова: существует оптимальный интерпретатор (не хуже любого)
- фиксируем оптимальный интерпретатор I и определяем колмогоровскую сложность K как K_I

свойства

- $K(x) \leq |x| +$ для некоторого c и для любых слов x (сравнение с тривиальным декомпрессором $l(p) = p$)

свойства

- $K(x) \leq |x| +$ для некоторого c и для любых слов x
  (сравнение с тривиальным декомпрессором $l(p) = p$)
- $K(x) \approx n$ для большинства слов длины n
  доля тех слов x длины n, для которых $K(x) < n - d$. не больше 2^{-d}
  (все эти слова имеют в качестве программ разные слова длины $< n - d$, которых $1 + 2 + 4 + \ldots + 2^{n-d-1} < 2^{n-d}$)

свойства

- $K(x) \leq |x| +$ для некоторого c и для любых слов x
  (сравнение с тривиальным декомпрессором $I(p) = p$)
- $K(x) \approx n$ для большинства слов длины n
  доля тех слов x длины n, для которых $K(x) < n - d$. не больше 2^{-d}
  (все эти слова имеют в качестве программ разные слова длины $< n - d$, которых $1 + 2 + 4 + \ldots + 2^{n-d-1} < 2^{n-d}$)
- алгоритмы не прибавляют информации:
  для всякого алгоритма A существует константа c:

$$K(A(x)) \leq K(x) + c$$

  при всех x.
  (Сравнение интерпретатора $I(p)$ с $A(I(p))$)

«теорема Гёделя в форме Чейтина»

функция K не вычислима
более того: *не существует вычислимой неограниченной нижней оценки для K* (вычислимой частичной функции, которая не превосходит K там, где определена, и значения которой не ограничены)

«теорема Гёделя в форме Чейтина»

функция K не вычислима
более того: *не существует вычислимой неограниченной нижней оценки для K* (вычислимой частичной функции, которая не превосходит K там, где определена, и значения которой не ограничены)

минимальное натуральное число,
которое нельзя задать миллионом слов

пусть K' — вычислимая неограниченная нижняя оценка
$A(n)$ — первое слово z, для которого обнаружилось значение $K'(z) > n$:

$$n < K'(A(n)) \leq K(A(n)) < K(n) + O(1) < \log n + O(1)$$

busy beavers

- $BB(n)$: максимальное время работы (оптимального) интерпретатора на программах длины не более n

busy beavers

- $BB(n)$: максимальное время работы (оптимального) интерпретатора на программах длины не более n
- $B(n)$: максимальное натуральное число сложности не больше n

busy beavers

- $BB(n)$: максимальное время работы (оптимального) интерпретатора на программах длины не более n
- $B(n)$: максимальное натуральное число сложности не больше n
- $$BB(n) \leq B(n+c); \quad B(n) \leq BB(n+c)$$

busy beavers

- $BB(n)$: максимальное время работы (оптимального) интерпретатора на программах длины не более n
- $B(n)$: максимальное натуральное число сложности не больше n
- $$BB(n) \leq B(n+c); \quad B(n) \leq BB(n+c)$$
- $BB(n)$ алгоритмически определяется самой долгоиграющей программой длины не более n, поэтому $K(BB(n)) \leq n + O(1)$, то есть $BB(n) \leq B(n + O(1))$

busy beavers

- $BB(n)$: максимальное время работы (оптимального) интерпретатора на программах длины не более n
- $B(n)$: максимальное натуральное число сложности не больше n
- $$BB(n) \leq B(n+c); \quad B(n) \leq BB(n+c)$$
- $BB(n)$ алгоритмически определяется самой долгоиграющей программой длины не более n, поэтому $K(BB(n)) \leq n + O(1)$, то есть $BB(n) \leq B(n + O(1))$
- $B(n-c) \leq BB(n)$: любое число $u > BB(n)$ имеет сложность больше $n - c$

busy beavers

- $BB(n)$: максимальное время работы (оптимального) интерпретатора на программах длины не более n
- $B(n)$: максимальное натуральное число сложности не больше n
- $$BB(n) \leq B(n+c); \quad B(n) \leq BB(n+c)$$
- $BB(n)$ алгоритмически определяется самой долгоиграющей программой длины не более n, поэтому $K(BB(n)) \leq n + O(1)$, то есть $BB(n) \leq B(n + O(1))$
- $B(n-c) \leq BB(n)$: любое число $u > BB(n)$ имеет сложность больше $n - c$

  зная u, можно запустить интерпретатор на всех входах, где он работает не больше u (слишком длинные входы не успеет прочесть), и найти слово z, которого нет не выходе; тогда $K(z) > n$ и по свойству неувеличения информации $K(u) > n - c$

приближённое решение проблемы остановки

приближённое решение проблемы остановки

- проблема остановки: по программе p узнать, остановится ли она, то есть определено ли $I(p)$

приближённое решение проблемы остановки

- проблема остановки: по программе p узнать, остановится ли она, то есть определено ли $I(p)$
- если бы она была алгоритмически разрешимой, то функция K была бы вычислимой

приближённое решение проблемы остановки

- проблема остановки: по программе p узнать, остановится ли она, то есть определено ли $I(p)$
- если бы она была алгоритмически разрешимой, то функция K была бы вычислимой
- но, может быть, есть алгоритм, который решает проблему остановки для «почти всех» p?

приближённое решение проблемы остановки

- проблема остановки: по программе p узнать, остановится ли она, то есть определено ли $I(p)$
- если бы она была алгоритмически разрешимой, то функция K была бы вычислимой
- но, может быть, есть алгоритм, который решает проблему остановки для «почти всех» p?
- нет: для больших n доля ошибок на словах длины n отделена от нуля

точная формулировка

точная формулировка

- пусть D — любой алгоритм, определённый на всех словах и дающий ответы «да» и «нет»; обозначим $\varepsilon_D(n)$ долю слов длины n, где D даёт неверный ответ для проблемы остановки. Тогда $\varepsilon_D(n) \geq \varepsilon$ для некоторого $\varepsilon > 0$ и для всех достаточно больших n.

точная формулировка

- пусть D — любой алгоритм, определённый на всех словах и дающий ответы «да» и «нет»; обозначим $\varepsilon_D(n)$ долю слов длины n, где D даёт неверный ответ для проблемы остановки. Тогда $\varepsilon_D(n) \geq \varepsilon$ для некоторого $\varepsilon > 0$ и для всех достаточно больших n.
- основная лемма: пусть $h(n)$ — число программ длины не больше n, на которых I останавливается. Тогда $K(h(n)) = n + O(1)$.

доказательство леммы

доказательство леммы

- очевидная часть: $h(n) \leq 2^{n+1}$, поэтому $K(h(n)) \leq n + O(1)$.

доказательство леммы

- очевидная часть: $h(n) \leq 2^{n+1}$, поэтому $K(h(n)) \leq n + O(1)$.
- менее очевидная часть: зная n и $h(n)$, можно найти список всех останавливающихся программ, все их выходы и слово сложности не меньше n, так что $K(\langle n, h(n)\rangle) \geq n - O(1)$.

доказательство леммы

- очевидная часть: $h(n) \le 2^{n+1}$, поэтому $K(h(n)) \le n + O(1)$.
- менее очевидная часть: зная n и $h(n)$, можно найти список всех останавливающихся программ, все их выходы и слово сложности не меньше n, так что $K(\langle n, h(n)\rangle) \ge n - O(1)$.
- технический приём: как избавиться от n. Пусть p программа для $h(n)$ длины $n - d$. Запишем самоограниченный код для d перед p, по этим данным можно получить d, p, n, $h(n)$ и наконец слово сложности не меньше n, поэтому

$$n \le O(\log d) + n - d + O(1); \quad d - O(\log d) \le O(1); \quad d \le O(1).$$

вывод теоремы из леммы

вывод теоремы из леммы

- D — приближённый всюду определённый алгоритм для проблемы остановки

вывод теоремы из леммы

- D — приближённый всюду определённый алгоритм для проблемы остановки
- e_n — число слов длины n, где D работает неверно

вывод теоремы из леммы

- D — приближённый всюду определённый алгоритм для проблемы остановки
- e_n — число слов длины n, где D работает неверно
- d_n — число слов длины n, где D даёт ответ «да»

вывод теоремы из леммы

- D — приближённый всюду определённый алгоритм для проблемы остановки
- e_n — число слов длины n, где D работает неверно
- d_n — число слов длины n, где D даёт ответ «да»
- $|d_n - h(n)| \leq e_n$

вывод теоремы из леммы

- D — приближённый всюду определённый алгоритм для проблемы остановки
- e_n — число слов длины n, где D работает неверно
- d_n — число слов длины n, где D даёт ответ «да»
- $|d_n - h(n)| \leq e_n$
- d_n можно вычислить по n, поэтому если бы e_n было мало, то $h(n)$ можно было бы задать существенно короче, чем n битами

вывод теоремы из леммы

- D — приближённый всюду определённый алгоритм для проблемы остановки
- e_n — число слов длины n, где D работает неверно
- d_n — число слов длины n, где D даёт ответ «да»
- $|d_n - h(n)| \leq e_n$
- d_n можно вычислить по n, поэтому если бы e_n было мало, то $h(n)$ можно было бы задать существенно короче, чем n битами
- тот же приём: если e_n записывается $n - d$ битами, то слово (самоограниченное описание d)+(запись e_n со знаком) имеет длину $O(\log d) + n - d$ и позволяет восстановить d, $n - d$, n, d_n и $h(n)$, так что его длина не меньше $n - O(1)$ и $n - d + O(\log d) \geq n - O(1)$, откуда $d = O(1)$.

теорема Клини – Арсланова

теорема Клини – Арсланова

- программы для функций (одной переменной)

теорема Клини – Арсланова

- программы для функций (одной переменной)
- $p \equiv p'$, если p и p' задают одну и ту же функцию

теорема Клини – Арсланова

- программы для функций (одной переменной)
- $p \equiv p'$, если p и p' задают одну и ту же функцию
- Клини: не существует вычислимой функции h без неподвижной точки (для которой $h(p) \not\equiv p$ для всех p)

теорема Клини – Арсланова

- программы для функций (одной переменной)
- $p \equiv p'$, если p и p' задают одну и ту же функцию
- Клини: не существует вычислимой функции h без неподвижной точки (для которой $h(p) \not\equiv p$ для всех p)
- Арсланов: если некоторый перечислимый оракул A вычисляет функцию h без неподвижной точки, то A полный (=позволяет решать проблему остановки)

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

(d) с помощью X по любому n можно указать слово сложности больше n.

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

(d) с помощью X по любому n можно указать слово сложности больше n.

Очевидное: $(c) \Rightarrow (b)$

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

(d) с помощью X по любому n можно указать слово сложности больше n.

Очевидное: $(c) \Rightarrow (b)$
Почти очевидное: $(c) \Leftrightarrow (d)$

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

(d) с помощью X по любому n можно указать слово сложности больше n.

Очевидное: $(c) \Rightarrow (b)$
Почти очевидное: $(c) \Leftrightarrow (d)$
$(b) \Rightarrow (a)$: если $p(\cdot)$ — программа со входом, то $p(\Lambda)$ — программа без входа; если y отлична от её выхода, то программа, которая на всех входах даёт y, не эквивалентна p

что это за оракулы?

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

(d) с помощью X по любому n можно указать слово сложности больше n.

Очевидное: $(c) \Rightarrow (b)$
Почти очевидное: $(c) \Leftrightarrow (d)$
$(b) \Rightarrow (a)$: если $p(\cdot)$ — программа со входом, то $p(\Lambda)$ — программа без входа; если y отлична от её выхода, то программа, которая на всех входах даёт y, не эквивалентна p
план: $(a) \Rightarrow (b) \Rightarrow (c)$

$(a) \Rightarrow (b)$

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть)

...

$(a) \Rightarrow (b)$

Лемма. Следующие свойства оракула X равносильны:

(a) X вычисляет функцию без неподвижной точки;

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть)

...

(без X) можно по программе без входа алгоритмически указать слово, которое эквивалентно её выходу, если он существует

этого достаточно: после этого X указывает неэквивалентное слово, которое поэтому не может быть её выходом

почему верно: пусть $p()$ программа без входа.

рассмотрим программу q:

- взять вход x
- запустить $p()$ и результат (если будет) обозначить u
- применить программу u ко входу x

если $p()$ даёт результат, то он эквивалентен q (а q мы написали по тексту $p()$ алгоритмически, без запуска)

$(b) \Rightarrow (c)$

Лемма. Следующие свойства оракула X равносильны:

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

$(b) \Rightarrow (c)$

Лемма. Следующие свойства оракула X равносильны:

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

- пусть есть программы $p_1, \ldots, p_k$ без входа, считаем, что их выходами являются последовательности длины k (перекодируем)

$(b) \Rightarrow (c)$

Лемма. Следующие свойства оракула X равносильны:

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

- пусть есть программы $p_1, \ldots, p_k$ без входа, считаем, что их выходами являются последовательности длины k (перекодируем)
- p'_i: программа, которая запускает p_i и потом даёт i-й член выхода

$(b) \Rightarrow (c)$

Лемма. Следующие свойства оракула X равносильны:

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

- пусть есть программы $p_1, \ldots, p_k$ без входа, считаем, что их выходами являются последовательности длины k (перекодируем)
- p'_i: программа, которая запускает p_i и потом даёт i-й член выхода
- q_i: число, заведомо отличающееся от выхода p'_i

$(b) \Rightarrow (c)$

Лемма. Следующие свойства оракула X равносильны:

(b) с помощью X можно по программе без входа указать слово, отличающееся от её выхода (если он есть);

(c) с помощью X можно по набору программ без входа указать слово, отличающееся от всех их выходов (те, которые есть)

- пусть есть программы $p_1, \ldots, p_k$ без входа, считаем, что их выходами являются последовательности длины k (перекодируем)
- p'_i: программа, которая запускает p_i и потом даёт i-й член выхода
- q_i: число, заведомо отличающееся от выхода p'_i
- $(q_1, \ldots, q_k)$: последовательность, заведомо отличающаяся от выходов всех p_i

что осталось доказать в теореме Арсланова

если с помощью перечислимого оракула можно по любому n найти слово сложности больше n, то A полный

что осталось доказать в теореме Арсланова

если с помощью перечислимого оракула можно по любому n найти слово сложности больше n, то A полный

- с помощью оракула A найдём слово сложности больше n

что осталось доказать в теореме Арсланова

если с помощью перечислимого оракула можно по любому n найти слово сложности больше n, то A полный

- с помощью оракула A найдём слово сложности больше n
- посмотрим, принадлежность каких чисел множеству A использована в этом вычислении

что осталось доказать в теореме Арсланова

если с помощью перечислимого оракула можно по любому n найти слово сложности больше n, то A полный

- с помощью оракула A найдём слово сложности больше n
- посмотрим, принадлежность каких чисел множеству A использована в этом вычислении
- пусть N — число шагов, нужных, чтобы они появились в перечислении A

что осталось доказать в теореме Арсланова

если с помощью перечислимого оракула можно по любому n найти слово сложности больше n, то A полный

- с помощью оракула A найдём слово сложности больше n
- посмотрим, принадлежность каких чисел множеству A использована в этом вычислении
- пусть N — число шагов, нужных, чтобы они появились в перечислении A
- тогда $N \geq B(n - O(1))$: любое число, большее N, позволяет перечислить приближение к A, которое можно использовать для нахождения слова сложности больше n

что осталось доказать в теореме Арсланова

если с помощью перечислимого оракула можно по любому n найти слово сложности больше n, то A полный

- с помощью оракула A найдём слово сложности больше n
- посмотрим, принадлежность каких чисел множеству A использована в этом вычислении
- пусть N — число шагов, нужных, чтобы они появились в перечислении A
- тогда $N \geq B(n - O(1))$: любое число, большее N, позволяет перечислить приближение к A, которое можно использовать для нахождения слова сложности больше n
- значит, $N \geq BB(n - O(1))$ и его можно использовать для решения проблемы остановки для всех слов до длины $n - O(1)$.