Конспекты по дискретной матемтике

Анатолий Коченюк, Георгий Каданцев, Константин Бац

2022 год, семестр 4

Последний семестр дискретной математики. Две больших темы: производящие функции (комбинаторика) и введение в теорию вычислимости.

1 Производящие фукнции

Рассмотрим последовательности $\{a_n\}_{n\in\mathbb{N}}, \{b_n\}_{n\in\ltimes} \subset \mathbb{R}(\mathbb{C})$. Назовём эти последовательности A и B и будем почленную сумму обозначать кратко $A+B$. Это несколько неудобно и неестественно, об этих конвенциях нужно договариваться.

Вместо этого давайте рассмотрим формальный степенной ряд, у которого члены последовательности это коэффициенты ряда.

$$A(t) = a_0 + a_1 t + a_2 t^2 + \dots a_n t^n + \dots$$

Тогда почленная сумма последовательностей будет соотвествовать обычной сумме рядов $A(t) + B(t)$.

Чтобы сдвинуть последовательность на 1 вправо, можно просто умножить степенной ряд на x.

Можем рассмотреть степенной ряд-композицию $A(t^2) = a_0 + 0t + a_1 t^2 + \dots$. Это степенной ряд, соответвующий последовательности $a_0, 0, a_1, 0, a_2 \dots$.

Таким образом, мы можем "оперировать" над последовательностью как единым целым, и это очень удобно.

Мы не рассматриваем степенные ряды с стороны, с которой на них смотрит мат. анализ: как способ приблизить фукнцию, с некоторым радиусом сходимости и т.д. У нас степенные ряды *формальные* и не всегда (всегда не) должны пониматься как функции, в которой в переменную можно подставить значение.

$\mathbb{R}[x]$ — кольцо многочленов с коэффициентами из кольца R, состоящий из формальных многочленов. $\mathbb{R}[x]^+$ — множество формальных степенных рядов.

Определение 1.0.1. Формальный степенной ряд $A(t)$ последовательности $\{a_n\}_{n\in\mathbb{N}}$ называется производящей функцией (generating function).

Название неудачное. Оно связано с другими корнями понятия производящей функции (они нужны не только в комбинаторике).

Определена сумма производящих функций и произведение

$$A(t)B(t) = C(t) \quad c_n = \sum_{i=0}^{n} a_i b_{n-i}$$

Несмотря на то, что мы работаем с бесконечным по размеру обьектом, нам необходимо только конечное число элементов, чтобы посчитать каждый отдельынй его член. Этот раздел дискретной математики не любит предельных переходов.

Определено умножение на скаляр.

$$\lambda A(t) = C(t) \quad c_n = \lambda a_n$$

Определено даже деление!

$$\frac{A(t)}{B(t)} = C(t);\ b_0 \neq 0 \quad c_n = \frac{a_n - \sum\limits_{i=0}^{n-1} c_i b_{n-i}}{b_0}$$

Так можно посчитать, например, что

$$C(t) = \frac{1}{1-t} = 1 + t + t^2 + \dots t^n + \dots; \quad a_n = 1$$

Мы записали короткой (конечной) производящей функцией бесконечную последовательность. Более того, мы можем эту запись взять и производить с ней операции (умножать и складывать с другими производящими функциями).

$$\frac{1}{1-2t} = 1 + 2t + 4t^2 + \dots + 2^n t^n; \quad c_n = 2^n$$

Обобщая мы видим, что

$$\frac{1}{1-bt} = \sum_{n=0}^{\infty} b^n t^n = C(bt)$$

Вообще говоря,

$$A(t) = \sum a_n t^n \quad A(bt) = \sum a_n b^n t^n$$

Замечание. Если $b_0 = \pm 1$, $a_i, b_i \in \mathbb{Z}$, тогда $C = \frac{A}{B}$ с целочисленными коэффициентами $c_i \in \mathbb{Z}$.

$$\frac{1}{1-t-t^2} = 1 + t + 2t^2 + \dots + F_n t^n + \dots$$

Мы одной дробью породили целую последовательность Фиббоначи!

А как быть, если мы хотим взять последовательность и найти представление для её производящей функции? Мы можем поступить так.

$$F_0 = 1,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2}$$

Отсюда $F(t) + F(t)t = \frac{F(t)-1}{t}$ (следует из операций над производящими функциями и рекурентным соотношением последовательности фиббоначи).

А что с дифференцированием? Обыкновенная операция взятия производной (формального) степенного ряда позволяет нам умножать член последовательности на его номер.

$$A(t) \to A(t)' \cdot t$$

Эту операцию можно производить многократно, получая последовательноть членов исходной п-ти в k степени.

Как найти представление производящей функции для последовательности $a_n = n$?

$$a_n = n * 1$$

Производящая функция для п-ти единиц это $\frac{1}{1-t}$. Тогда

$$A(t) = \left(\frac{1}{1-t}\right)' t = \frac{t}{(1-t^2)}$$

Формальное деление это подтверждает.

А что с интегрированием? С интегрированием всё не очень мило.

А что с композицией?

$$C(t) = A(B(t))$$

Здесь много проблем доставляет свободый коэффициент у B. Давайте его уберем — $b_0 = 0$. Теперь мы можем посчитать

$$c_n = \sum_{k=0}^{n} a_k \sum_{n=i_1+i_2+\dots i_k} b_{i_1} b_{i_2} \dots b_{i_k}$$

Пример с доминошками.

Пример с деревьями.

1.1 Линейные рекуррентые последовательности. Регулярные производящие функции

Определение 1.1.1 (Линейные рекуррентные последовательности)**.** Пусть даны первые k членов последовательности $a_0, a_1, \ldots, a_{k-1}$. А все следующае члены определяются, как линейная комбинации k предыдущих.

$$a_n = a_{n-1} \cdot c_1 + a_{n-2} \cdot c_2 + \cdots + a_{n-k} \cdot c_k.$$

Такаяя последовательность называется **линейной рекурентной последовательностью**.

Пример. Числа Фибоначч. $F_0 = 1,\ F_1 = 1, \forall\ n \geqslant 2\ :\ F_n = F_{n11} + F_{n-2}$

$$\frac{1}{1-t-t^2} = \sum_{i=0}^{\infty} F_i t^i.$$

Обозначим $F(t) = \sum\limits_{i=0}^{\infty} F_i t^i = F_0 t^0 + F_1 t^1 + \sum\limits_{i=2}^{\infty} F_i t^i =$
$= 1 + t + \sum\limits_{i=2}^{\infty} F_{i-1} t^i + \sum\limits_{i=2}^{\infty} F_{i-2} t^i = 1 + t + t \cdot \sum\limits_{i=1}^{\infty} F_i t^i + t^2 \sum\limits_{i=0}^{\infty} F_i t^i =$
$= 1 + t + t \cdot (F(t) - 1) + t^2 \cdot F(t) \implies F = 1 + t \cdot F + t^2 F \implies$
$F(t) = \dfrac{1}{1-t-t^2}.$

Теорема 1.1.1. Пусть есть линейная рекуррентная последовательность порядка k:
$a_0, a_1, \ldots, a_{k-1}, \ldots.$

Даны $a_0, \ldots, a_{k-1},\ \forall n \geqslant k\ :\ a_n = \sum\limits_{i=1}^{k} a_{n-i} \cdot c_i.$

Тогда $A(t) = \sum\limits_{i=0}^{\infty} a_i t^i = \dfrac{P(t)}{Q(t)}$ — рациональная функция, где
$Q(t) = 1 - c_1 t - c_2 t^2 - \cdots - c_k t^k$, а $P(t) = \ldots.$

Доказательство. Обозначим

$$A(t) = \sum_{i=0}^{\infty} a_i t^i = \sum_{i=0}^{k-1} a_i t^i + \sum_{n=k}^{\infty} a_i t^n.$$

Сразу заменим последнюю сумму предположеним из теоремы, получим

$$A(t) = \sum_{i=0}^{k-1} a_i t^i + \sum_{n=k}^{\infty} t^n \sum_{i=1}^{k} a_{n-i} \cdot c_i = S + \sum_{i=1}^{k} c_i \sum_{n=k}^{\infty} a_{n-i} t^n = S + \sum_{i=1}^{k} c_i \cdot t^i \cdot \sum_{n=k-1}^{\infty} a_n t^n =$$

$$= S + \sum_{i=1}^{k} c_i \cdot t^i \cdot (A(t) - A_{k-1}(t)) = X.$$

Пусть $C(t) = \sum\limits_{k=1}^{k} c_i t^i$, тогда $Q(t) = q - C(t)$. $\quad X = S + C(t) \cdot A(t) - \sum\limits_{k=1}^{k} c_i t^i A_{k-i}(t) = Y.$

Пусть $F(t) \,\%\, t^k = \sum\limits_{i=0}^{k-1} f_i t^i$, тогда $A_{k-i}(t) = A(t) \,\%\, t^{k-1}$.

$$\sum_{k=1}^{k} c_i t^i A_{k-i}(t) \cdot Ak - i(t) = A(t) \,\%\, t^{k-i} = (C(t) \cdot A(t)) \,\%\, t^k \implies$$

$$A(t) = \sum_{i=0}^{k-1} a_i t^i + C(t) \cdot A(t) - (C(t) \cdot A(t)) \,\%\, t^k \implies A(t)(1 - C(t)) = ((1 - C(t)) \cdot A(t)) \,\%\, t^k$$

$$\Longrightarrow A(t) = \frac{P(t)}{Q(t)}, \quad \text{где } Q(t) = 1 - C(t) = 1 - c_1 t - c_2 t^2 - \dots - c_k t^k,$$

$$P(t) = \left(\left(\sum_{i=0}^{k-1} a_i t^i \right) \cdot Q(t) \right) \mod t^k.$$

∎

Пример. Для чисел фибоначчи: $a_0 = a_1 = 1, \ c_1 = c_2 = 1 \Longrightarrow$

$$A(t) = \frac{(1+t)\cdot(1-t-t^2) \mod t^2}{1-t-t^2}.$$

$$a_0 = 6, \ a_1 = -3, \ c_1 = c_2 = 1 \quad \Longrightarrow \quad A(t) = \frac{(6-3t)\cdot(1-t) \mod t^2}{1-t-t^2} = \frac{6-9t}{1-t-t^2}.$$

Доказательство в обратном направлении. Частный случай:

$$\frac{1}{1-C(t)} = A(t), \ A(t)\cdot(1-C(t)) = 1,$$

$$t^0 = a_0 = 1, \ t^1 : a_1 \cdot 1 - a_0 c_1 = 0, \ t^2 : a_2 \cdot 1 - a_1 \cdot c_1 - a_0 c_2 = 0.$$

Посмотрим на некоторую производящую функцию, например $\frac{1-3t+6t^3}{1-t-t^2-t^4}$. Понимаем, что $a_n = a_{n-1} + a_{n-2} + a_{n-4}$.

$$a_0 = 1, \ a_1 = 1 - 3 = -2, \ a_2 = 1 - 2 = -1, \ = -1 - 2 + 6 = 3$$

$$A(t)\cdot Q(t) = P(t). \quad \sum_{i=0}^{n} q_i \cdot a_{n-i} = p_n \quad a_n = p_n - \sum_{i=1}^{k} q_i \cdot a_{n-i}.$$

∎

Пусть $a_0, a_1, \dots, a_{k-1}, \ \forall n \geqslant k \ : a_n \sum_{i=1}^{k} a_{n-i} \cdot c_i$.

Задача: посчитать a_n.
Можно явно за $\mathcal{O}(n \cdot k)$.
Можно через возведение матрицы в степень за $\mathcal{O}(k^3 \log_2 n)$.
Потом мы научимся делать это за $\mathcal{O}(k^2 \log_2 n)$.

На самом деле, для одной и той же числовой последовательности можно получить несколько производящих функций.

$$A(t) = \frac{P(t)}{Q(t)} \cdot \frac{Q(-t)}{Q(-t)} = \frac{P(t)\cdot Q(-t)}{Q(t)\cdot Q(-t)}.$$

Например, для чисел Фибоначчи:

$$\frac{1}{1-t-t^2} \cdot \frac{1+t+t^2}{1+t+t^2} = \frac{1+t-t^2}{1-3t^2+t^4}. \quad F_n = F_{n-2}\cdot 3 - F_{n-4}.$$

Теорема 1.1.2. Для производящих функций, задающих рекурентные соотношения эквивалентны следующие высказывания

- $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots +$
- $A(t) = \frac{P(t)}{Q(t)}$
- $a_n = \sum_{i=1}^{b} p_i(n) \cdot r^i$, где $r_i \in \mathbb{C}$

$Q(t) = 1 - c_1t - c_2t^2 - \dots - c_kt^k$

$P(t)$ определяет то, как надо подправить первые члены, чтобы получились те, которые нужны.

$A(t)Q(t) - P(t) = 0$.

Как посчитать r?

Пусь $Q(t) = 1 - rt$,

$a_n = r \cdot a_{n-1}$

$a_m = r \cdot a_{m-1}$

$a_{m+1} = r \cdot a_m$

$\dots$

$a_n = r^n \cdot \frac{a_{m-1}}{r^{m-1}}$

Пусть $Q(t) = (1 - r_1t)(1 - r_2t),\ \ r_1 \neq r_2$.

Лемма 1.1.2.1. $Q(t) = \prod\limits_{i=1}^{n}(1 - r_it)$, где $r_i \neq r_j$ $\dfrac{P(t)}{Q(t)} = \sum\limits_{i=1}^{n} \dfrac{P_i(t)}{1 - r_it}$

$Q(t) = \sum\limits_{i=1}^{n} d_ir_i^n$.

r_i — числа, обратные корням многочлена Q. Если степень Q равно k, то Q имеет ровно k корней (с учетом кратности).

Таким образом, $Q(t) = q_k\prod\limits_{i=1}^{k}(t - t_i) = (-1)^kq_k\prod\limits_{i=1}^{k}\left(1 - \dfrac{t}{t_i}\right) \cdot t_i =$

$= \left[(-1)^kq_k \cdot \prod\limits_{i=1}^{k} t_k\right] \prod\limits_{i=1}^{k}(1 - r_it) = \alpha\prod\limits_{i=1}^{k}(1 - R_it)$.

Почему нет корня 0? Потому что $Q(t)$ имеет вид $Q(t) = 1 - c_1t - \dots$.

Пример. Рассмотрим числа Фибоначчии. $F(t) = \dfrac{1}{1 - t - t^2}$.

Корни $t_{1,2} = \dfrac{-1 \pm \sqrt{1+4}}{2}$, обратные корини $r_{1,2} = \dfrac{1 \mp \sqrt{5}}{2}$. Обратные корни разные — нам очень приятно.

$$Q(t) = \left(1 - \frac{1-\sqrt{5}}{2}\right)\left(1 - \frac{1+\sqrt{5}}{2}\right).$$

$$\frac{1}{(1 - r_1t)(1 - r_2t)} = \frac{c_1}{1 - r_1t} + \frac{c_2}{1 - r_2t}, \quad c_1(1 - r_2t) + c_2(1 - r_1t) = 1 \implies$$

$$\begin{cases} c_1 + c_2 = 1 \\ c_1(-r_2) + c_2(-r_1) = 0 \end{cases} \implies c_2 = \frac{-r_2}{r_1 - r_2} = \frac{-1-\sqrt{5}}{2 \cdot (-\sqrt{5})} = \frac{5+\sqrt{5}}{10}, \quad c_1 = \frac{5-\sqrt{5}}{10}.$$

$$a_n = c_1r_1^n = \frac{5-\sqrt{5}}{10} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n b_n = c_2r_2^n = \frac{5+\sqrt{5}}{10} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n \implies$$

$$f_n = \frac{5-\sqrt{5}}{10} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n + \frac{5+\sqrt{5}}{10} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right).$$

Замечание. Если λ — единсвенный минимальный по модулю комплексный корень $Q(t)$, $A(t) = \dfrac{P(t)}{Q(t)}$, то $a_n = \Theta\left(\dfrac{1}{\lambda^n}\right)$.

$\dfrac{1}{(1 - rt)^2} = \dfrac{1}{1 - 2rt + r^2t^2}$. $a_0 = 1,\ a_1 = 2r,\ \ a_2 = 3r^2,\ a_3 = 4r^3,\quad \dots,\ a_n = (n+1)r^n$.

$\dfrac{1}{r}\left(r^nt^n\right)' = \dfrac{1}{r}\sum nr^nt^{n-1} = \sum nr^{n-1}t^{n-1} = \sum(n+1)r^nt^n$.

Лемма 1.1.2.2. ${\frac{1}{1-rt}}^s = \sum\limits_{n=0}^{\infty} p_s(n)r^nt^n$

Доказательство. Докажем по индукции.

1. База. $s = 0$ — просто

2. Переход. Далее много формул. $\left(\frac{1}{(1-rt)^s}\right)' = \frac{-r(-s)}{(1-rt)^{s+1}}$. $\left(\sum\limits_{n=0}^{\infty} p_s(n) r^n t^n\right)' = \sum\limits_{n=1}^{\infty} n p_s(n) r^n t^{n-1} = \sum\limits_{n=0}^{\infty} (n+1) p_s(n+1) r^{n+1} t^n$. $\frac{1}{(1-rt)^{s+1}} = \sum\limits_{n=0}^{\infty} \frac{n+1}{s} p_s(n+1) r^n t^n$, $p_{s+1}(n) = p_s(n+1)\frac{n+1}{s} = \sum\limits_{i=0}^{s-1} p_{s,i}(n+1)^i \frac{n+1}{s}$.

$$p_{s,i} = \frac{a_{s,i}}{b}, \quad b = s!, \ a_{s,i} \in \mathbb{Z}$$

■

Теорема 1.1.3. Пусть $A(t) = \dfrac{P(t)}{Q(t)}$, r_i — обратный корень кратности s_i Q_i, количество различных корней b. Тогда начиная с некоторого места (но точно, начиная с k) $a_n = \sum\limits_{i=1}^{b} p_i(n) r_i^n$, $\quad \deg p_i = s_i - 1$, $\quad \sum\limits_{i=1}^{b} s_i = k$.

Доказательство.

$$Q(t) = \prod_{i=1}^{b} (1 - r_i t)^{s_i}, \quad \frac{P(t)}{Q(t)} = \sum_{i=1}^{b} \frac{P_i(t)}{(1 - r_i t)^{s_i}}.$$

■

Если λ_i — единственный минимальный комплексный корень $Q(t)$ кратности s_i. Тогда $a_n = \Theta\left(\dfrac{n^{s_i-1}}{\lambda_i^n}\right)$.

Пример. $a_n = n^3$, $\quad a_n = 4 \cdot a_{n-1} - 6 \cdot a_{n-2} + 4 \cdot a_{n-3} - a_{n-4}$.

Подберем поправку первых членов: $P(n) = t + 4t^2 + t^3$.

Утверждение 1.1.1. Асимптотическое поведение рекуррентности не зависит от начальных значений, оно зависит только от коэффициентов соотношений.

Утверждение 1.1.2. Пусть $\lambda_1, \lambda_2, \ldots, \lambda_z$ — минимальные корни максимальной кратности.

$\lambda_j = \dfrac{e^{i\varphi_j}}{r}$. $\varphi_j = \dfrac{p_j}{q_j} \cdot 2\pi$.

Пусть $\overline{q} = LCM(q_j)$. Тогда последовательность a_i имеет асимптотическое поведение при $i \% \overline{q} = const$.

1.2 Комбинаторика и производящие функции

Пример. Замещение прямоугольника $2 \times n$ доминошками вида 1×2.

$$\frac{1}{1 - t - t^2} = 2 + t + t^2 + t^2 + t^3 + t^3 + t^3 + \ldots = 1 + t + 2t^2 63t^3 + \ldots + F_n t^n$$

Комбинаторные обьекты это конструкции, которые состоят из атомов и разных связей атомов между собой. Под атомом мы понимаем некоторую неделимую часть комб. обьекта. Давайте все наши комбинаторные обьекты сложим.

В этой сумме заменим каждый атом на t^{ω}, где ω — вес данного атома. Потом t^{ω} атомов одного обьекта перемножим.

Вес обьекта — сумма весов его атомов.

Пример. A – множество комбинаторных объектов. Давайте их просуммируем.

$\Delta_1 + \Delta_2 + \Delta_3 + \ldots$

атом (неделимое) – то, что мы считаем.

$t^{\omega(\Delta_1)} + t^{\omega(\Delta_2)} + t^{\omega(\Delta_3)} + \ldots = \sum\limits_{n=0}^{\infty} a_n t^n = A(t)$ — производящая функция для объектов веса t.

Базовые комбинаторные объеты

Определение 1.2.1 (Базовые объекты). $U = \{u\} \quad \omega(u) = 1 \quad u(t) = t$ – производящая функция для этих комбинаторных объектов

$B = \{a, b\} \quad \omega(a) = \omega(b) = 1 \quad B(t) = 2t$

$E = \{\varepsilon\} \quad \omega(\varepsilon) = 0 \quad E(t) = 1$

$E_k = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k\} \quad E_k(t) = k$

$D = \{a, A\} \quad \omega(a) = 1 \quad \omega(A) = 2 \quad D(t) = t + t^2$

Операции конструирования

Определение 1.2.2 (Дизъюнктное объединение). A, B – множества комбинаторных объектов и $A \cap B = \varnothing$

Пусть $C = A \cup B$. Тогда производящая функция

$$\begin{aligned} C(t) &= A_1 + A_2 + \ldots + B_1 + B_2 + \ldots \\ &= t^{\omega(A_1)} + t^{\omega(A_2)} + \ldots + t^{\omega(B_1)} + t^{\omega(B_2)} + \ldots \\ &= A(t) + B(t) \end{aligned}$$

.

Определение 1.2.3 (Упорядоченная пара (прямое произведение)). Пусть $A, B, \quad A(t), B(t)$ — их производящая функция. Определим пару C, как $A \times B = C = \{\langle a, b\rangle \mid a \in A, b \in B\}$.

$C_n = C \cap \{x \mid \omega(x) = n\} \quad \langle a, b\rangle\, \omega(a) = i \quad \omega(b) = j \quad i + j = n \quad j = n - i$.

$C_n = \cup A_i \times B_{n-i}$. $c_n = \sum\limits_{i=0}^{n} a_i \cdot b_{n-i} \implies C(t) = A(t) \cdot B(t)$.

Замечание (Комбинаторный мысл прямого произведения). Пусть у нас есть объекты $A = A_1 + A_2 + \ldots + A_k + \ldots, \quad B = B_1 + B_2 + \ldots + B_k + \ldots$.

$(A_1 + A_2 + \ldots) \cdot (B_1 + B_2 + \ldots) = A_1 \cdot B_1 + A_1 \cdot B_2 + \ldots + A_2 \cdot B_1 + A_2 \cdot B_2 + \ldots$.

$\langle a, b\rangle \quad t^{\omega(a)} t^{\omega(b)} = t^{\omega(a)+\omega(b)}$.

Определение 1.2.4 (Последовательность (sequence)). Определим последовательность из A, как $SeqA = \{[], [A_1], [A_2], \ldots, [A_1, A_2], [A_1, A_3], \ldots\}$.

$SeqA = [] \cup A_1 \cdot ([] + [A_1] + [A_2] + \ldots) + A_2 \cdot ([] + \ldots) = 1 + A \times SeqA$.

$B = SeqA \implies B(t) = 1 + A(t)B(t) \implies B(t) = \frac{1}{1-A(t)}$.

Определение 1.2.5 (Последовательность (sequence), второй способ). $SeqA = A^0 \cup A^1 \cup A^2 \cup \ldots \cup A^k \cup \ldots$

A^i – декартова степень, последовательности длины i

$$B(t) = A(t)^0 + A(t)^1 + A(t)^2 + \ldots + A(t)^k + \ldots = \frac{1}{1 - A(t)}$$

Пример. $SeqU = \{[], [u], [u, u], [u, u, u], \ldots\} = N$. $n_k = 1, \; U(t) = 1 \implies SeqU = \dfrac{1}{1-t}$.

$SeqB = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\} = C$. $c_n = 2^n, \quad B(t) = 2t \implies C(t) = \dfrac{1}{1-2t}, \quad c_n = 2 \cdot c_{n-1}$.

$C_n = 2^n \quad B(t) = 2t \quad C(t) = \frac{1}{1-2t}$

$C_n = 2C_{n-1}$
$SeqE = \{[], [\varepsilon], [\varepsilon, \varepsilon], \ldots\} = C$
$E(t) = 1 \quad C(t) = \frac{1}{1-E(t)} = \frac{1}{1-1} = \frac{1}{0}$ ☹
$C_0 = +\infty??$

Пример. $C = SeqD = \{\varepsilon, a, aa, aA, A, Aa, AA, \ldots\}$
$C(t) = \frac{1}{1-D(t)} = \frac{1}{1-t-t^2}$
a – одна вертикальная доминошка, вес 1. A – две горизонтальные доминошки, вес 2.
$C = aC + AC \quad C(t) = tC(t) + t^2C(t)$

Определение 1.2.6 (Множество)**.** Обозначается Set или $PSet$.
$B = \{a, A\} \quad SetB = \{\varnothing, \{a\}, \{A\}, \{a, A\}\}$.
$C = SetA$.
$a \in A \quad B_a = \varepsilon + a$ – либо берём, либо не берём. C – дкартово произведение по всем a.
$C(t) = \prod\limits_{a \in A} \left(1 + t^{\omega(a)}\right) = \prod\limits_{n=0}^{\infty} (1 + t^n)^{a_n}$.

Пример. Возьмем $U = \{u\}$. $SetU = C = \{\varnothing, \{u\}\}$. Найдем $C(t)$.

$$C(t) = \prod_{n=0}^{+\infty} (1 + t^n)^a_n = (1 + t)^1 = 1 + t.$$

Пусть $B = \{a, A\}$, $\quad C = SetB$. Заметим, что $b_1 = 1, b_2 = 1$.

$$C(t) = \prod_{n=0}^{+\infty} (1 + t^n)^{b_n} = (1 + t)(1 + t^2) = \underbrace{1}_{\varnothing} + \underbrace{t}_{a} + \underbrace{t^2}_{A} + \underbrace{t^3}_{a,A}.$$

$\prod\limits_{n=0}^{\infty} (1 + t^n)^{a_n} = (a + t_0)^{a_0} \cdot \prod\limits_{n=1}^{\infty} (1 + t^n)^{a_n} = 2^{a_0} \prod\limits_{n=1}^{\infty} (1 + t^n)^{a_n}$.

Определение 1.2.7 (Мультимножество)**.** Обозначается $MSetA$.
Мы можем включить каждый объект $0, 1, 2, \ldots$.
$\varepsilon + a + aa + \ldots = Seq\{a\}$.
$a_1 \in A,\ a_2 \in A \implies Seq\{a_1\} \times Seq\{a_2\} = MSet\{a_1, a_2\}$.
$MSetA = \prod\limits_{a \in A} Set\{a\}$.

$$C(t) = \prod_{a \in A} Seq\{a\} = \prod_{a \in A} \frac{1}{1 - t^{\omega(a)}} = \prod_{n=1} \left(\frac{1}{1 - t^n}^{a_n}\right) = \prod_{n=1}^{\infty} (1 - t^n)^{-a_n}.$$

Пример. $U = \{u\} \quad C = MSetU \quad C(t) = \prod\limits_{n=1}^{\infty} (1 - t^n)^{-u_n} = (1 - t)^{-1} = \frac{1}{1-t}$.
$B = \{a, A\} \quad C = MSetB \quad b_1 = 1 = b_2$.
$C(t) = \prod\limits_{n=1}^{\infty} = (1 - t)^{-1}(1 - t^2)^{-1} = \frac{1}{(1-t)(1-t^2)}$.
Ассимптотика C_n.
$Q(t) = (1 - t)(1 - t^2) = (1 - t^2)(1 + t)$.
Корни: $t = \pm 1$. Обратные корни $r = \pm 1$ Кратность $r_1 = 1 \quad s_1 = 2 \qquad r_2 = -1 \quad s_2 = 1$.
$(a_n + b) \cdot 1^n + c \cdot (-1)^n$
$\frac{1}{2}n + \frac{1}{2}(-1)^n + const$

1.2.1 Циклы (cycle)

$B = \{a, b\} \quad CycB = \{\varepsilon, a, b, ab, aa, bb, aaa, aab, abb, bbb, aaaa, aaab, aabb, abab, abbb, bbbb, \dots\}$

Раньше мы называли такие комбинаторные объекты *ожерельями.*

$C = CycB = \bigcup_{k=1}^{\infty} (CycA)_k$.

$C(t) = \sum\limits_{k=1}^{\infty} C_k(t), \quad C_k(t)$ — производящая функция длинны k.

$C_k(t)$ – последоватльности длины k с точностью до циклического сдвига.

S_k – последовательности длины $k \quad = (A(t))^k \quad C_k(t)/G \quad G$ – группа циклических сдвигов.

$C_{k,n} = \dfrac{1}{k} \cdot \sum\limits_{i=0}^{k-1} |I(i)|$.

Количество классов эквивалентности по лемме Бёрнсайда равно $\gcd(i, k)$. Внутри класса одинаковые объекты. Размер класса $\dfrac{k}{\gcd(i, k)}$.

n кратно $\dfrac{k}{\gcd(i, k)}$. $S_{\gcd(i,k)} \dfrac{n \cdot \gcd(i, k)}{k}$.

$$C_{n,k} = \frac{\sum\limits_{j=0}^{k-1} S_{\gcd(i,k), \frac{n \cdot gcd(i,k)}{k}}}{k}$$

2 Формальные языки

Как объяснить комьютеру, что я вляется словом в нашем языке.

Пусть $L \subset L^*$. Мы знаем два способа задания языка.

1. ДКА = Р.В.

2. КСГ = МП-автомат.

А что мы вообще сделать с помощью компьютера? Ну что-то нельзя сделать из-за фикики: путешествовать во времени, ... А что нельзя сделать с помощью математики?

Есть два способа рассказать компьютеру о языке:

- Научить его распознавать слово из языка.

  Например, ДКА. Нужен мета-язык описания конечных автоматов и само описание автомата.

- Научить компьютер пораждать слова из языка.

  У нас есть также мета-язык описания пораждения языка и само описание.

  Например, даем компьютеру парсер регулярных выражений, само регулярное выражение, компьютер строит дерево разбора и генерирует нам слова.

На сколько сильно мы можем усложнить граматику нашего языка. Регулярные языки не умели геренрировать палиндромы, КСГ не умели генерировать $1^n 2^n 3^n$. А на сколько еще мы можем усложнить наши описания?

Вообще, языков может быть 2^{Σ^*} — несчетное количество. Но для нас это не страшно, почти все из них описать не возможно. А что можно описать? Ну пусть то, что умеет понимать компьютер. **А что такое вообще компьютер?**

Для осознания мощности компьютеров существуют модели. Мы будем изучать **Машину Тьюринга**. Машина Тьюринга основывается на ленте. Так же существует, например, машина Маркова, там немного другой принцип.

Метод описания

Пусть у нас есть *соременный x86* компьютер. Как описать модель такого комьютера? Ну есть почти неограниченная память и есть какие-то операции (сложение, умножение, вызов функции). Что такое программа для такого компьютера? Программа — $\underbrace{\quad}_{\text{строка}} \in \Pi^*$. На самом деле можно записать в битовом формате $\mathbb{B} = \{0, 1\}$, то есть $\underbrace{\quad}_{\text{Description}} \in \mathbb{B}^*$. На самом деле, описание программ самый мощный инструмен. Пусть есть описание конченого автомата, запишим его в константу, добавим код имплементации конечного автомата, получим описание языка от автомата на языке программ.

Утверждение 2.0.1 (Тезиси Тьюринга–Чёрча)**.** Все, что можно выразить на «обычном компьютере» можно выразить на Машине Тьюринга.

Замечание. Почему это не утверждение или теорема? Надо доопределить понятие «обычный комьютер» и тогда получится какое-то утверждение.

А что вообще может выдавать программа? *Comilation error* не больная проблема, давайте в таком случае программа будет делать что-то конкретное, например *No*. Если *Runtime error*, путь тоже будем возвращать что-то конкретное. *Memory limit* — не проблема, попросим пользователя добавить память или подождем, пока изобретут комьютер лучше. Самое интересное — *Time limit*, это обозначают $\bot$. Таким, образом наша программа будет возвращать три значения $И, Л, \bot$. Тогда хочется добавить ограничение на время исполненя, правда, это немного ограничит класс зязыков.

Определение 2.0.1. Язык L разрешимый (рекурсивный), если $\exists$ программа p $\forall x \in L \implies p(x) = 1, \quad x \notin L \implies p(x) = 0$.

Определение 2.0.2. Язык L полуразрешимый (перечислимый, рекурсивнно перечислимый), если $\exists$ программа p $\forall x \in L \iff p(x) = 1$.

На самом деле полуразрешимые описания языков — мкисмальные по мощности. Разрешимый — максимальный по мощности прикладной способ описания.

Существуют не разрешимые и не полуразрешимые языки.

Метод порождения

Пусть у нас есть компьютер, который по опсанию выводит список слов. Можно выводить первые n слов.

Опять же, понятно, что описание с помощью компьютера макисмальное по мощности.

Определение 2.0.3. Язык L перечеслимым, если можно написать программу, которая выодит его слова.

Теорема 2.0.1. L полуразрешим $\iff$ L перечислим.

Определение 2.0.4. Градуированный лексикографический порядк — перечисление в порядке увеличения длинны, а среди слов с равной длинны лексикографически.

Доказательство. $\implies$

Неверный подход 1.

```
for (x ∈ Σ*)
    if p(x):
        print(x)
```

Не верный поход, попытка 2

```
for (TL = 1; True; TL++)
    for (x ∈ \Sigma^{*})
        if p |_T (x):
            print(x)
```

Подход правильный:

```
    for (TL = 1; True; TL++)
        for (x ∈ Σ*[: TL])
            if p |_T (x):
                if (x ∉ was):
                    print(x)
                    was.add(x)
```

$\Longleftarrow$

Пусть у нас есть q — перечислитель L.

```
p(x):
    while q.next() != x:
        pass
    return True
```

■

Пример непреичлимого языка

Программа набор из 0 и 1. Пусть A — предикат. $L_A = \{p \mid A(p)\}$ — формальный язык.

Определение 2.0.5 (Универсальный язык)**.** $U = \{\langle p, x\rangle \mid p(x) = 1\}$.

Замечание. U — полуразрешим.

Доказательство. Давайте сделаем

```
inU(⟨p,x⟩):
    return p(x)
```

■

Теорема 2.0.2. U не разрешим.

Доказательство. Пусть есть функция $\mathrm{inU}(\langle p, x\rangle)$ — всегда завершается.

```
q(x):
    if inU(⟨x,x⟩):
        return 0
    else:
        return 1
```

Посчитаем q(q).

Если $\mathrm{q(q)} = 1 \implies \mathrm{inU}(\langle q, q\rangle) = \mathrm{false} \implies \mathrm{q(q)} = 0$ (плохо).

Пусть $\mathrm{q(q)} = 0 \implies \mathrm{inU}\ (\langle q, q\rangle) = \mathrm{true} \implies \mathrm{q(q)} = 1$ (тоже плохо).

■

Теорема 2.0.3. Если A и $\overline{A}$ — полуразрешим $\implies A$ — разрешим.

Утверждение 2.0.2. A разрешим $\implies \overline{A}$ — разрешим.

Доказательство теоремы. .

```
inA(x):
    for (TL = 1; +∞):
        if p |_TL(x):
            return 1
        if q |_TL(x):
            return 0
```

■

Не полуразрешим

Пример. Язык дополнение до U не полуразрешим.

На самом деле эти множества биективны:

- Строки Σ^*
- Программы Prog
- Числа $\mathbb{N}^+$

$\text{Prog} \underset{\text{id}}{\iff} \Sigma^* \underset{\text{grad order}}{\iff} \mathbb{N}^+.$

- Полуразрешимые языки $\iff$ вычислимые функции $A \subset \mathbb{N} \to \{0, 1\}$.
- Разрешимые языки $\iff$ всюду определенные (Hall) вычислимые функции $\mathbb{N} \to \{0, 1\}$.

2.1 А обязательно ли разрешать компьютеру зависать?

Определение 2.1.1. Язык программирования называется **полным** для A, если для любого перечислимого языка из A можно задать его описание на этом языке.

Иными словами, любую вычислимую функцию можно задать при помощи этого языка программирования.

Определение 2.1.2. Язык называется **вычислимым** (компилируемым), если по описанию и словы мы можем сказать подходит или нет.

Определение 2.1.3. Язык программирования **независающий**, если $\forall$ программы и $\forall$ слова он не зависнет.

Теорема 2.1.1. Не существует полного для разрешимых языков вычислимого, не зависающего метаязыка описания.

То есть, не существует вычилимой нумерации всех всюду определенных вычислимых функций.

Доказательство. Предположим, что она (нумирация) существует. Вот эта нумерация $f_1, f_2, f_3, \dots$.

	x_1	x_2	x_3	x_4
f_1				
f_2				
f_3				
f_4				

$g(i) = f_i(x_i) + 1, \quad g$ — всюдуопределенная, вычислимая.

$g(i) =! f_i(x_i)$ ■

Следствие 2.1.1.1. На конечном автомате нельзя интерпретировать конечный автомат.

2.2 Разрешимость

У нас есть пример неразрешимого языка. Это язык $U = \{\langle p, x\rangle \mid p(x) = 1\}$. А если мы хоти проверить еще какой-то язык? Можно ли куда-то замести под ковер рассужения?

Определение 2.2.1. m — сведение (исторически many to one reduction, но в реальности это оказалось не удачным, поэтому называют mappng сведение).

Говорят язык $A \leqslant_m B$, если существует всюдуопределенная вычислимая функция f, такая, что $x \in A \iff f(x) \in B$.

Теорема 2.2.1. Если A не разрешимый, $A \leqslant_m B \implies B$ — не разрешимый.

Доказательство. Предположим B разрешимый, то есть есть in$B(x)$. Тогда A разрешается программой

```
inA(x):
    return inB(f(x))
```

■

Пример (Задача останова)**.** Пусть $HALT = \{p \mid p(\epsilon)$ не зависает $\}$.

Сведем U к $HALT$.

```
f (⟨p,x⟩):
    return "q(y):
        def p = ...
        if p(x) != 1:
            while True:
                pass
    "
```

Заметим, что f всюду определлнная и вычислимая.

f является m сведением U к HALT.

$q \in HALT \iff \langle p, x\rangle \in U$.

Разминка перед Т. У-Р. Пусть мы хотим проанализировать поведение программы.

$A = \{p \mid p$ чему-то удовлетворяет, но не зависает$\}$. Тогда A не зарешим.

Доказательство. Также напишим

```
f(⟨p,x⟩):
    return "q(y):
        if p(x) = 1:
            сделай то, что удовлетворяет A
        else: while True: pass
    "
```

■

2.3 Теорема Успенского-Райса

E = множества перечислимых языков.

c: $\Sigma \equiv$ char, s: $\Sigma^* \equiv$ string, L $\subset \Sigma^* \equiv$ set<string> = lang, $2^{\Sigma^*} \equiv$ set<lang>, $E \subset 2^{\Sigma^*} \equiv$ set<lang>.

Свойство перечислимых языков:

$A \subset E \equiv$ set<lang>

$L \in A \to L$ удовлетворяет A

$L \notin A \to L$ не удовлетворяет A.

$L(A) = \{p \mid L(p) \in A\}$

$L(p) = \{x \mid p(x) = 1\}$.

Теорема 2.3.1 (Успенского-Райса)**.** Язык любого нетривиального свойства перечислимых языков не разрешим.

$A = \varnothing \quad L(A) = \varnothing, \qquad A = E \quad L(A) = \Sigma^*$. Есть $A \neq \varnothing$, $A \neq E \implies L(A)$ не разрешим.

Доказательство. Пусть $\varnothing \notin A$. Пусть какой-то язык $X \in A$.

$L(A)$ — разрешим $inA(p)$. Напишем следующий код, разрешающий U.

```
in U(⟨p,x⟩):
    q = "q(y):
            if p(x) = 1:
                return inX(y)
            else:
                return 0
        "
    return inA(q)
```

Утверждается, что получился разрешитель для U. Если $p(x) = 1$, то $L(q) = X \in A$. А если $p(x) = 0$ или $p(x)$ зависает, то $L(q)\varnothing$.

Таким образом, $\langle p, x\rangle \in U \iff q \in L(A)$.

То есть f m–сводит U к $L(A)$. ■

2.4 Теорема о рекурсии

Бывают программы, что

```
q - разрещитель
    f(x): ...
    if q(f):
        while True: pass
    else:
        return 1
```

Пример (игрушечный)**.** Что выведет эта инструкция?

```
Написать 2 раза, второй раз в ковычках:
"Написать 2 раза, второй раз в ковычках:"
```

Действительно, получится:

```
Написать 2 раза, второй раз в ковычках:
"Написать 2 раза, второй раз в ковычках:"
```

То есть в каком-то месте программы мы можем вывести исходный код программы.

```
f(...):
.
.   getSource()
.
.
```

Программа выведет:

```
.
getSource():
.
.
.
.
```

Наиболее интересное применение — quine — программа, которая выводит свой исходный код.

Как же это делать?

```
f(...):
.
.
.   getSource()
.

getSource():
    s ⟵ getAuxSource()
    return s + "getAuxSource():" +
    "   return \"" + s + "\""

getAuxSource():
    return "f(...):
```

```
            ...
            getSource():
                s ⟵ getAuxSource()
                return s + \"getAuxSource():\" +
                    \"    return \\\"\" + s + \"\\\"\"
            "
```

Замечание. Заметим, что весь код программы состоит из того, что выведет getAuxSource, плюс определение getSource.

Теорема 2.4.1 (О рекурсии)**.** Пусть $V(x, y)$ – вычислимая функция $\implies \exists$ вычислимая функция $f : f(y) = V(f, y)$.

Благодаря этой теореме можно проще доказывать неразрешимость некоторых языков.

Пример. $\mathrm{HALT} = \{p \mid p(\varepsilon) \neq \perp\}$ неразрешим

Доказательство. Пусть h — разрещитель HALT.

```
p():
    if h(p) == 1 :
        while True: pass
    else:
        return 1
```

■

Пример (Теорема Успенского–Райса)**.** $A \subseteq RE$, $A \neq \varnothing$, $A \neq RE$.
$L(A) = \{p \mid L(p) \in A\}$ — неразрешим.

Доказательство. Пусть inL_A — разрешитель.
Пусть $\underbrace{M}_{inM(x)} \in A$, $\underbrace{N}_{inN(x)} \notin A$.

```
f(x):
    if (inL_A(getSource())):
        return inN(x)
    else:
        return inM(x)
```

■

Замечание. Такая подростковая программа: послушаем родителей и сделаем наоборот.

Еще примеры невычислимых функций

Пример. $K(x)$ — Колмогоровская сложность
$K(x) = \min$ длинна программы p, что $p() = x$.
К сожалению, нельзя написать программу, которая бы оптимально кодировалоа строку.

Утверждение 2.4.1. $K(x)$ — невычислима

Доказательство. Пусть $K(x)$ вычислима, напишем код.

```
p():
    for x \in Σ*:
        if k(x) > |getSource()|:
            return x
```

Мы нашли программу, у котороой колмогоровская сложность больше, чем сложность программы p, которая без проблем выводит эту строчку. ■

Пример (Busy Beaver (BB(n)))**.** Эта функция принимает число n и возвращает максимальное число шагов, которое делает программы длинны n (из n строчек или символов).

Утверждение 2.4.2. $BB(n)$ невычилима. Напишем программу, которая будет работать дольше.

```
f():
    for i = 0 .. BB(|getSource()|):
        pass
    return
```

2.5 Проблемы вычислимости

Когда мы говорим про вычислимость, мы представляем программу на *c++*, *python*, *псевдоязыке*. Но с математикикой это не очень удобно связывать.

Задача 1 (Проблема Соответствия Поста)**.** Есть число n. Далее следует n строк с числами $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$.

Задача, вывести `YES` или `NO` и сертификат, если существует последовательность индесов такая, что $a_{i_1}, a_{i_2}, a_{i_3}, \ldots a_{i_n} = b_{j_1}, b_{j_2}, b_{j_3}, \ldots, b_{j_n}$.

Задача 2 (Проблема полимомно)**.** Полимино — плоская геометрическая фигура, состоящая из n одноклеточных квадратов, соединенных по сторонам.

Дано n, w, h — количество типов полимино, высота и ширина. Далее n фигурок полимино.

Требуется понять, можно ли замостить четверть плоскости такими фигурами.

Определение 2.5.1. [Машиана тьюринга]. Σ — входной алфавит, $\Pi \supset \Sigma$. $B \in \Pi \backslash \Sigma$ — пробел.

Q — множество состояний. $Y \in Q$, $N \in Q, Y \neq N$. $S \in Q$ — стартовое состояние.

$\delta : (\backslash\{Y, N\}) \times \Pi \to Q \times P \times \{\leftarrow, \to\}$.

Мгновенное описание Машины Тьюринга $\alpha\#_p\beta$.

$\vdash$ — переходит за один шаг. $\alpha\#_p\beta \vdash \xi\#_q\eta$, если

$$\begin{cases} \alpha = \xi\alpha, \eta = ad\beta, \quad \delta(p, c) = \langle q, d, \leftarrow \rangle, \\ \xi = \alpha d, \eta = \beta', \quad \delta(q, d, \to), \\ \xi = \alpha, \eta = d\beta', \quad \delta(q, d, \cdot) \end{cases}$$

Сейчас мы попробуем приблизить машину тьюринга к современному компьютору.

М.Т. $\to$ многодорожечная М.Т. $\to$ многоленточная М.Т $\xrightarrow[\text{яма с крокодилами и Тезис Т.Ч.}]{}$. современный компьютор.

А потом мы попробуем упростить нашу машину тьюринга.

Двух счетчиковая машина $\leftarrow$ трех счетчиковая машина $\leftarrow$ двустековая машина $\leftarrow$ М.Т.

А также мы докажем эфкивавалентность М.Т.

1. Машине Мароква,
2. Грамматикам нулевого порядка,
3. Кл. Автомат.

Определение 2.5.2. Многодорожечная Машина Тьюринга

Пусть у нас есть не одна дорожка, а несколько, то есть мы можем не портить исходное слово.

Утверждение 2.5.1. М.Д.М.Т. $\approx$ М.Т.

Доказательство. Действительно, $\Pi' = \Pi^k$. ∎

Утверждение 2.5.2. Можно считать, что лента односторонне бесконечная и нет пробелов B.

Доказательство. Чтобы не ставить пробелы добавим сурогатный пробел $\{\overline{B}\} \cup \Pi = \Pi'$.

Потом отрежем один бесконечный край ленты, развернем и приклеим к началу со сдвигом на один. Получим ленту с двумя бесконечными в одну сторону дорожками.

При этом вычислительную мощность мы не потеряли.

∎

Определение 2.5.3. Многоленточная Машина Тьюринга

Пусть у нас есть k лент и на каждой из них своя головка. То есть $\delta : Q^k \to Q \times \Pi^k \times \{\leftarrow, \to, \cdot\}^k$.

Утверждение 2.5.3. М.Л.М.Т. $\approx$ М.Т.

Доказательство. Докажем эквивалентность М.Л.М.Т. $2k$–дорожечной МТ. На нечетных дорожках будут данные, на четных дорожках будем хранить положение головки в $i/2$–й ленте.

Переход будем делать по стадиям пока не подвиним все головки.

$Q' = Q \times \Pi^k \times 2^k \cup$ (специальные состояния для обратного прохода + служебные). ■

Замечание. Пусть М.Л.М.Т сделала t шагов для симуляции 1 шага. Требуется $\mathcal{O}(t)$ шагов 1 л М.Т.

А для всех t шагов потребуется $\mathcal{O}(t^2)$.

Теорема 2.5.1 (Тьюринга-Черча)**.** L — полуразрешимый $\iff \exists$ М.Т. $L(m) = L$.

Усилинная версия. Замедление происходит не более, чем в полином раз.

Теперь давай в обратную сторону.

Пусть у нас есть недетерменированная М.Т. $\delta : Q \times \Pi \to P_{<\infty}\left(Q \times \Pi \times \{\leftarrow, \to, \cdot\}\right)$.

Определение 2.5.4. k–стековая машина $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Pi^k \to P_{<+\infty}\left(Q \times (\Pi^*)^k\right)$

Теорема 2.5.2. L — перечислим $\iff$ разрешим на 2–стековой машине.

Определение 2.5.5. k–счетчиковая машина:

$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \{= 0, > 0\}^k \to P_{<+\infty}\left(Q \times \{+1, -1, +0\}^k\right)$.

Теорема 2.5.3. k–стековая машина $\implies$ $k + 1$–счетчиковая машина

Теорема 2.5.4. k–счетчиковая машина $\implies$ 2–счетчиковая машина.

Другие вычислители, эквивалентные Машине Тьюринга.

Вообще, приимущество М.Т. в ее локальности. То есть $p \vdash q$, растояние Левенштейна между p и q равно $\mathcal{O}(1)$.

Определение 2.5.6. Нормальные алгорифмы Мароква (Машина Мароква).

У нас есть строковый регистр (строка данная на вход). Программа для Машины Маркова: переходы $s_1 \to t_1, s_2 \to t_2, \ldots, s_k \to \mathbb{STOP}, \ldots$. Просматриваем список правил, берем первое встретившееся правило, где $s_i = s$ и заменяем s на t_i.

Теорема 2.5.5 (Маркова – Тьюринга)**.** М.М. $\equiv$ М.Т.

Доказательство. В одну сторону $\pm$ понятно. А как на Машине Маркова сделать М.Т.?

Добавим последнее правило $\epsilon \to \#_s$.

$\delta(p, c) = (q, d, \rightarrow)$ переделаем в $\#_p c \to d\#_q$.
$\beta(p, c) = (q, d, \leftarrow)$ переделаем в набор правил для всех a $a\#_p c \to \#_q ad$.
$\#_Y \to Y$, $\#_N \to N$ $\#_{\mathbb{STOP}} \to \mathbb{STOP}$.

$\blacksquare$