Лекция 16

MIP = NEXP

(Конспект: А. Малова)

Конспект основан на лекции 26 курса Джона Каца (Даниэль Апон. **MIP** = **NEXP***) и статье U. Feige, S. Goldwasser, L. Lovasz, S. Safra, M. Szegedy. Interactive proofs and the hardness of approximating cliques. JACM, 1996 с использованием статьи L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. Computational Complexity, 1(1):3-40, 1991.*

16.1 Основные определения

Интерактивные протоколы с одним проверяющим и несколькими доказывающими являются естественным обобщением интерактивных протоколов с одним доказывающим (класс **IP**). Формально такие протоколы описывается следующим образом. Пусть проверяющий (verifier) V — вероятностная полиномиальная машина, доказывающие (provers) $P_1, \ldots, P_k$ — машины неограниченной вычислительной мощности, где k в общем случае является некоторым полиномом от длины входа. V и все P_i видят входное слово x, prover'ы пытаются убедить verifier'а, что оно принадлежит некоторому языку $\mathtt{L}$. P_i знают, как устроен V, но не могут видеть содержимое его лент, в частности, ленты со случайными битами. V может общаться с любым P_i, но никакие два доказывающих не могут общаться между собой, то есть все P_i являются *независимыми*, тогда

Определение 16.1. Язык $\mathtt{L} \in$ **MIP** (multi-prover interactive protocol), если

- $x \in \mathtt{L} \Rightarrow Pr[P_1, \ldots, P_k \text{ убедят } V] = 1$,

- $x \notin \mathtt{L} \Rightarrow \forall P'_1, \ldots, P'_k, Pr[P_1, \ldots, P_k \text{ убедят } V] \leq \frac{1}{2}$,

Заметим, что, как и в **IP**, вероятность ошибки в **MIP** можно понижать, но запросто это можно делать только *последовательным*, а не *параллельным* повторением протокола.

Также напомним определение класса **NEXP**:

Определение 16.2. $\mathbf{NEXP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTime}(2^{n^c})$

и сформулируем основную теорему, которая будет доказана в оставшихся секциях конспекта.

Теорема 16.1. $\mathbf{MIP} = \mathbf{NEXP}$.

Доказательство будет вестись аналогично доказательству того факта, что $\mathbf{IP} = \mathbf{PSPACE}$.

16.2 MIP ⊆ NEXP

Необходимо показать, что любой интерактивный протокол с несколькими доказывающими может быть смоделирован на **NEXP** машине. При доказательстве аналогичного вложения для $\mathbf{IP} \subseteq \mathbf{PSPACE}$ моделирующий алгоритм перебирает все ответы prover'ов. Но этот алгоритм не гарантирует их независимость. Предположим, что V сначала общается с prover'ом P_1, а после с prover'ом P_2, тогда в ветках перебора, соответствующих разной истории общения V и P_1 могут выбираться в качестве оптимальных разные ответы P_2 на одни и те же сообщения V. Таким образом получится, что ответы P_2 зависят не только от того, что ему сообщил V, что противоречит определению **MIP**.

Однако, что из себя представляет P_i? Это по сути некоторая *функция*, принимающая всю историю предыдущего общения V и P_i и очередное сообщение V и возвращающая ответное сообщение. Поскольку размер входных и выходных данных полиномиален, таблица значений такой функции при всех возможных аргументах имеет экспоненциальный размер. Угадаем недетерминированно таблицы значений всех P_i и, перебирая все возможные случайные биты V, оценим вероятность допуска V и вынесем соответствующий вердикт. Ответы P_i тут перебирать уже не придется — они будут браться из таблиц значений соответственно общению V и P_i.

16.3 NEXP $\subseteq$ MIP

Доказательство этого включения будет вестись аналогично доказательству включения **PSPACE** $\subseteq$ **IP**. Мы рассмотрим полную задачу в классе **NEXP** и получим сведение к булевой формуле экспоненциально большого размера. Арифметизуем полученную формулу таким образом, чтобы любой дизъюнкт булевой формулы был вычислим за полиномиальное от длины входа время. После этого мы воспользуемся **IP**-протоколом для **PSPACE** с некоторыми модификациями.

16.3.1 MIP = PPTO

Для описания протокола удобно доказать эквивалентность **MIP** другой модели вычисления — вероятностным машинам с оракулом (которому мы не доверяем).

Определение 16.3. Язык $\mathtt{L}$ принимается полиномиальной по времени вероятностной машиной M с оракулом O, если:

- $\forall x \in \mathtt{L}\ \exists O\ \Pr[V^O(x) = 1] > 1 - 2^{-\mathrm{poly}(|x|)}$,
- $\forall x \notin \mathtt{L}\ \forall O'\ \Pr[V^{O'}(x) = 1] < 2^{-\mathrm{poly}(|x|)}$.

Класс таких языков будем называть **PPTO** (Probabilistic Polynomial-Time machine with access to an Oracle $\mathtt{O}$).

Рассмотрим некоторый язык $\mathtt{L} \in$ **MIP**. Заметим, что $\mathtt{L}$ также принимается вероятностной машиной с оракулом, а именно, когда V хочет отправить сообщение P_i, он отправляет оракулу i сообщение, а также историю общения с P_i. Честный оракул поведет себя так же, как честные prover'ы, и убедит V. Теперь предположим, что нечестный оракул смог обмануть, но тогда набор $P_1, \ldots, P_k$, где P_i ведет себя так же, как оракул после принятия первым аргументом i, также смог бы обмануть V, что по определению класса **MIP** возможно только с маленькой вероятностью.

Точно также вероятностной машиной с оракулом моделируется **IP**-протокол. Однако одного prover'а недостаточно, чтобы смоделировать оракул, так как prover помнит все предыдущие запросы, в отличие от оракула. Таким образом, у prover'а больше возможности обмнануть. Тем не менее, уже двух независимых prover'ов хватает для того, чтобы смоделировать оракул. Протокол выглядит следующим образом. Сначала V отправляет P_1 все запросы, которые до этого были отправлены оракулу. Затем он случайным образом выбирает один из запросов и посылает его P_2. Если P_2 дает тот же ответ, что и P_1, и P_1 убедил V, то V принимает слово, иначе отвергает. Заметим, что если оба prover'а отвечают, как

честный оракул, то они смогут убедить V. Рассмотрим, что происходит, когда prover'ы врут. Возможны два случая:

1. Пусть не существует такого запроса, на котором бы ответ P_2 отличался бы от ответа P_1, тогда это значит, что P_1 ведет себя так же, как если бы он не знал истории сообщений, то есть как оракул. Но по определению класса **PPTO** нечестный оракул может обмануть verifier'а с маленькой вероятностью.

2. Если ответ P_2 на какой-то запрос отличается от ответа P_1, то с вероятностью хотя бы $\frac{1}{|x|^{O(1)}}$ мы случайно попадем в этот запрос, так как количество запросов не больше, чем время работы V. Таким образом, нечестные prover'ы обманут с вероятностью не более $(1 - \frac{1}{|x|^{O(1)}})$. Последовательным повторением полиномиальное число раз эту ошибку можно понизить до константной.

Таким образом, общение со всеми prover'ами мы можем заменить общением всего с одним оракулом.

16.3.2 NEXP-полная задача и арифметизация

Аналогично классу **NP** полной задачей в классе **NEXP** является выполнимость формул в 3-КНФ, но в случае класса **NEXP** формула содержит экспоненциальное число переменных и дизъюнктов. В связи с этим есть небольшая тонкость, поскольку, так как сведение должно быть полиномиальным по времени, то вернуть формулу полностью алгоритм сведения не может. Вместо этого функция сведения будет возвращать алгоритм (заданный схемой), который по номеру дизъюнкта возвращает сам дизъюнкт. Более формально это сведение описывает следующая теорема:

Теорема 16.2 (Теорема Кука-Левина для **NEXP**). *Пусть* $\mathtt{L} \in \mathbf{NEXP}$*, тогда существует такая константа* c*, что выполнены следующие условия:*

1. *Для любого входного слова* $x \in \{0,1\}^n$ *существует формула в 3-КНФ с экспоненциальным числом переменных и дизъюнктов* $\Phi_x(X(0), X(1), \ldots, X(2^{n^c})) = \bigwedge_{i=0}^{2^{n^c}-1} C_i$*, где:*

   (a) $C_i = t_1^i X(b_1^i) \vee t_2^i X(b_2^i) \vee t_3^i X(b_3^i)$,

   (b) $t_j^i \in \{0,1\}$ *(запись вида* $0X$ *обозначает* $\neg X$*, а запись* $1X$ *—* X*),*

(c) $b_j^i \in \{0,1\}^{n^c}$ *обозначают номера переменных.*

2. *Существует полиномиальный по времени алгоритм, который принимает* $x \in \{0,1\}^n$ *и выдаёт булеву схему* $g_x : \{0,1\}^{n^c} \to \{0,1\}^{3n^c+3}$, *которая обладает следующими свойствами:*

   (a) *на входе* $0 \le i < 2^{n^c}$, *схема* g_x *выдаёт* $t_1^i, t_2^i, t_3^i, b_1^i, b_2^i, b_3^i$,

   (b) $x \in \mathtt{L} \iff$ *существует подстановка значений переменным* $X(0), X(1), \ldots, X(2^{n^c})$, *выполняющая каждый дизъюнкт* C_i.

Опишем процесс арифметизации булевой формулы Φ_x. Для начала обозначим через f_x функцию, которая принимает номер дизъюнкта i, номера переменных b_1, b_2, b_3 и знаки t_1, t_2, t_3 и возвращает 1, если дизъюнкт номер i имеет вид $C_i = t_1X(b_1) \vee t_2X(b_2) \vee t_3X(b_3)$.

Далее для краткости будем обозначать тройку номеров переменных b_1, b_2, b_3 через b, а тройку знаков через t.

Обозначим через S_1 условие, которое утверждает, что если дизъюнкт встречается в формуле, то он выполним на всех значениях, заданных функцией A. То есть, говоря другими словами, это условие для всех возможных наборов i, t, b утверждает, что, если i-ый дизъюнкт описывается соответствующими t и b, то он выполним на значениях, заданных функцией A:

$$S_1 \stackrel{\text{def}}{=} \forall i, t, b : [f_x(i,t,b) = 1] \Longrightarrow [t_1A(b_1) \vee t_2A(b_2) \vee t_3A(b_3) = 1].$$

Теперь перепишем формулировку теоремы 16.2 следующим образом:

Лемма 16.1. *Рассмотрим язык* $\mathtt{L} \in \mathbf{NEXP}$ *и* $x \in \{0,1\}^n$, *тогда существуют такая константа* c *и такая функция* $f_x : \{0,1\}^{4n^c+3} \to \{0,1\}$, *вычислимая за время* $poly(n)$, *что* $x \in \mathtt{L} \iff \exists A : \{0,1\}^{n^c} \to \{0,1\} : S_1$.

Условие S_1 эквивалентно следующему условию:

$$S_2 \stackrel{\text{def}}{=} \neg\exists i, t, b : [f_x(i,t,b) = 1] \wedge \neg[t_1A(b_1) \vee t_2A(b_2) \vee t_3A(b_3) = 1].$$

Обозначим часть этого выражения как

$$S_3 \stackrel{\text{def}}{=} [f_x(i,t,b) = 1] \wedge \neg[t_1A(b_1) \vee t_2A(b_2) \vee t_3A(b_3) = 1].$$

Пусть $1 = True$ и $False = 0$, тогда верна следующая лемма.

Лемма 16.2. *Существует такой вычислимый за полиномиальное от n время полином $P_x(i, b, t, z, a)$ степени не больше, чем $poly(n)$, что для любых $i, b_j \in \{0,1\}^{n^c}$ и для любых $t_j \in \{0,1\}$ выполнено:*

$$\sum_{z \in \{0,1\}^{n^{O(1)}}} P_x(i, b, t, z, A(b_1), A(b_2), A(b_3)) = \begin{cases} 1, & \text{если } S_3 \text{ выполнено} \\ 0, & \text{в противном случае} \end{cases},$$

где $a = (a_1, a_2, a_3)$ — подстановка переменных b_1, b_2, b_3, а z обозначает дополнительные переменные.

Доказательство. Определим следующий предикат, он нам понадобится, чтобы арифметизовать выражение $[t_1 A(b_1) \vee t_2 A(b_2) \vee t_3 A(b_3) = 1]$:

$$p(t, a) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{если } (a_1, a_2, a_3) \text{ выполняет дизъюнкт для } (t_1, t_2, t_3) \\ 0, & \text{в противном случае} \end{cases}.$$

Обозначим через $q(t, a)$ полином, интерполирующий p. Заметим, что функция f_x, определенная выше, вычислима за полиномиальное время. Используя теорему Кука-Левина, мы можем построить булеву формулу. При построении этой формулы используется таблица состояний машины и ленты в каждый момент времени, имеющая полиномиальный размер. При арифметизации полученной булевой формулы, мы получим арифметическую формулу F_x, вычислимую за полиномиальное время, которая удовлетворяет следующему условию:

$$\sum_{z \in \{0,1\}^{n^{O(1)}}} F_x(i, t, b, z) = \begin{cases} 1, & \text{если } f_x(i, t, b) = True \\ 0, & \text{в противном случае} \end{cases}.$$

Здесь z пробегает по всем возможным таблицам (корректная таблица всего одна, так что в этой сумме не более одного ненулевого слагаемого). Соединяя этот результат и определение S_3, приходим к следующим равенствам:

$$\sum_{z \in \{0,1\}^{n^{O(1)}}} P_x(i, b, t, z, A(b_1), A(b_2), A(b_3)) =$$

$$\sum_{z \in \{0,1\}^{n^{O(1)}}} F_x(i, t, b, z)(1 - q(t, a)) = \begin{cases} 1, & \text{если } S_3 = True \\ 0, & \text{в противном случае} \end{cases}$$ □

Теперь пусть $True = 0$ и $False > 0$, тогда мы можем арифметизовать S_1 и S_2 следующим образом.

Лемма 16.3. *Для всех x существует вычислимый за полиномиальное время полином $P^*_x(i,b,t,z,a)$, такой что*

$$\sum_{i,b,t,z} P^*_x(i,b,t,z,A(b_1),A(b_2),A(b_3)) = \begin{cases} 0, & \text{если } S_1 = True \text{ и } \forall b \in \{0,1\}^{\cdots} A(b) \in \{0,1\} \\ > 0, & \text{в противном случае} \end{cases},$$

где $i,b,t,z \in \{0,1\}^{n^{O(1)}}$.

Доказательство. $\sum\limits_{i,b,t,z} P^2_x(i,b,t,z,A(b_1),A(b_2),A(b_3)) =$
$\sum\limits_{i,b,t}\sum\limits_{z} P^2_x(i,b,t,z,A(b_1),A(b_2),A(b_3)) =$
По предыдущей лемме это выражение равно:
$= \sum\limits_{i,b,t} \begin{cases} 1, & \text{если } S_3 = True \\ 0, & \text{в противном случае} \end{cases} = \begin{cases} 0, & \text{если } S_2 = True \\ > 0, & \text{в противном случае} \end{cases}$; для того, чтобы обеспечить $A(b) \in \{0,1\}$ добавим также члены $(A(b)(1-A(b))^2$. Сумма квадратов равна нулю тогда и только тогда, когда каждый из членов ноль; иначе она строго положительна. □

Теперь, переписывая лемму 16.1, используя лемму 16.2 и лемму 16.3, приходим к следующей теореме.

Теорема 16.3. *Пусть $\mathtt{L} \in$ **NEXP**, тогда существует такая константа c, что для всех n и для любого $x \in \{0,1\}^n$ существует вычислимый за полиномиальное время полином $P^*_x(i,b,t,z,a)$ степени не больше, чем $n^{O(1)}$, такой что*

$$x \in \mathtt{L} \iff \exists A : \{0,1\}^{n^c} \to \{0,1\} : \sum_{i,b,t,z} P^*_x(i,b,t,z,A(b_1),A(b_2),A(b_3)) = 0,$$

где $i,b,t,z \in \{0,1\}^{n^{O(1)}}$ для некоторой константы, которая экспоненциально зависит от c и $\mathtt{L}$.

Напомним, что выше мы доказали эквивалентность классов **MIP** и **PPTO**. Значение выполняющей функции A будет задано оракулом. Отметим, что назначающая функция A может быть представлена как мультилинейная (линейная по всем координатам) функция (так как $x^k = x$, если $x \in \{0,1\}$, то есть можно понизить степень каждой переменной), если оракул честный. Это позволяет нам использовать следующую лемму:

Лемма 16.4 (Schwartz—Zippel)**.** *Пусть I подмножество некоторого поля. Если f и g два различных полинома, степени d, тогда их значения совпадают не более чем в $\frac{d}{|\mathcal{I}|}$ доле всех векторов $\vec{x} \in \mathcal{I}^m$.*

Чтобы увеличить долю векторов, на которых различные функции расходятся, расширим область определения функции A с $\{0,1\}^{n^c}$ до $\mathcal{I}^{n^c}$, где $\mathcal{I} = \{0, 1, \ldots, N-1\}$ — элементы множества $\mathcal{I}$ и N достаточно велико.

Мы будем говорить, что функция $A : \mathcal{I}^{n^c} \to \mathcal{I}$ выполняет Φ_x тогда и только тогда, когда сужение функции $A|_{\{0,1\}^{n^c}}$ удовлетворяет Φ_x.

Для проверки функции, заданной prover'ом, на мультилинейность нужен отдельный протокол.

Теорема 16.4. *Пусть $A : \mathcal{I}^{n^c} \to \mathcal{I}$ произвольная функция и пусть $\varepsilon > 0$ выбран произвольно маленьким, тогда существует вероятностный тест на мультилинейность, работающий за время $poly(|x|, \frac{1}{\varepsilon})$, который получает оракульный доступ к функции A и с высокой вероятностью отвергает A, если она не мультилинейна.*

Описание этого теста и доказательство его корректности будут приведены ниже. Теперь мы готовы описать протокол.

16.3.3 Протокол

Протокол состоит из двух стадий.

1. На входе x verifier вычисляет P_x, как описано выше. Общение с prover'ами мы заменим общением с одним prover'ом и оракулом для назначающей функции $A : \mathcal{I}^{n^c} \to \mathcal{I}$. Verifier проверяет мультилинейность оракульной функции. Если тест отвергает функцию, то verifier считает, что назначающая функция не корректна, то есть не задает выполняющий набор, и также отвергает вход. В противном случае, verifier продолжает и переходит к следующему шагу.

2. В этом раунде verifier должен проверить условия теоремы 16.3. Заметим, что полином P_x зависит от полиномиального числа переменных, каждая из которых имеет полиномиальный размер. Опишем **IP**-протокол для проверки условий теоремы. Для этого обозначим $h(i, b, t, z) = P_x(i, b, t, z, A(b_1), A(b_2), A(b_3))$. Очевидно, что h также зависит от полиномиального числа переменных. Для удобства описания протокола переобозначим все переменные, от которых зависит h, через $x_1, x_2, \ldots, x_m$. Итак, verifier хочет проверить, что выполняется равенство

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_m=0}^{1} h(x_1, x_2, \ldots, x_m) = a.$$

В каждом раунде **IP**-протокола verifier будет рассматривать полином от меньшего числа переменных, который определяется следующим образом:

$$h_i(x_1, \ldots, x_i) = \sum_{x_{i+1}=0}^{1} \cdots \sum_{x_m=0}^{1} h(x_1, x_2, \ldots, x_m).$$

Очевидно, что $h_{i-1} = h_i(x_i = 0) + h_i(x_i = 1)$. Для проверки этого условия в каждом раунде verifier случайным образом выбирает случайное число $r_i \in \mathcal{I}$ и вычисляет значение $b_i = h_i(r_1, r_2, \ldots, r_i)$. Будем считать, что $b_0 = a$. Пусть для полинома $g_i(x) = h_i(r_1, r_2, \ldots, r_{i-1}, x)$ prover выдал полином g'_i, тогда verifier должен проверить, что $b_{i-1} = g'_i(0) + g'_i(1)$. Если это условие не выполняется, то verifier отвергает вход, иначе он продолжает. По окончании m раундов verifier проверяет корректность условия $b_m = h(r_1, r_2, \ldots, r_m)$. Для окончательного вычисления h verifier должен сделать три запроса к оракулу для вычисления $A(b_1), A(b_2), A(b_3)$, подставляя соответствующие значения r_j (напомним, что в текущих обозначениях b_i соответствуют каким-то x_j).

Доказательство. Если слово было из языка, то честные prover и оракул убедят verifier'а.

Пусть слово не из языка, и prover пытается доказать неверное равенство. Обозначим через d степень полинома P_x. Будем считать, что prover не может использовать для обмана полиномы степени выше, чем d, так как иначе бы verifier обнаружил это во время проверки функции A на мультилинейность. Пусть prover во время какого-то раунда **IP**-протокола обманул и выдал $g'_i \neq g_i$, и пусть он смог обмануть так, чтобы пройти в следующий раунд (то есть условие $b_{i-1} = g'_i(0) + g'_i(1)$ выполнено), тогда по лемме 16.4 с вероятностью $1 - \frac{d}{|\mathcal{I}|}$ значения этих двух полиномов разойдутся, то есть $b_i = g'_i(r_i) \neq g_i(r_i)$. Значит, мы продолжим доказывать неверное равенство. По окончании всех раундов с вероятностью $1 - \frac{dm}{|\mathcal{I}|}$ мы по-прежнему будем иметь неверное равенство, а значит, verifier обнаружит это во время последнего раунда при проверке $b_m = h(r_1, r_2, \ldots, r_m)$. □

16.4 Тест на мультилинейность

Сначала введем несколько определений. Расстоянием между двумя функциями $f_1, f_2 : \mathcal{I}^m \to \mathcal{I}$ называется величина $\Delta(f_1, f_2) = \frac{|\{\vec{y} | f_1(\vec{y}) \neq f_2(\vec{y})\}|}{|\{\text{все } \vec{y}\}|}$, то есть доля векторов из $\mathcal{I}^m$, на которых значения функций расходятся.

Расстоянием от функции f до множества всех мультилинейных функций, определенных на $\mathcal{I}^m$, обозначим его через ML, называется величина: $\Delta_{ML}(f) = \min_{f^* \in ML} \Delta(f, f^*)$. Набор из трех различных векторов x, y, z будем называть *тройкой в направлении* i, если эти три вектора отличаются только в i-й координате.

Заметим, что функция мультилинейна тогда и только тогда, когда она линейна по всем возможным тройкам. Будем называть тройку f-линейной или хорошей, если существует такая функция g, которая линейна по i-й координате и совпадает с функцией f в этих трех точках.

Теорема 16.5. *Пусть $\delta = \Delta_{ML}(f)$, где $\delta > \frac{1}{10}$, тогда вероятность найти не f-линейную тройку не меньше, чем $\frac{c}{n}$, где c — некоторая константа.*

Доказательство. Рассмотрим два случая в зависимости от величины δ:

1. Пусть $\delta \in [\frac{1}{10}; \frac{9}{10}]$. Обозначим через f^* ближайшую к f мультилинейную функцию.

   Вероятность $Pr[(x, y, z)$ — не f-линейная$] \geq$
   $3Pr[(x, y, z)$ — не f-линейна и $f(x) = f^*(x)$, $f(y) \neq f^*(y)]$, где вероятность берется по случайным точкам на одной прямой. Далее, эта вероятность $= 3(Pr[f(x) = f^*(x)$ и $f(y) \neq f^*(y)] -$
   $Pr[(x, y, z)$ — хорошая и $f(x) = f^*(x)$ и $f(y) \neq f^*(y)])$.

   Для того, чтобы оценить эту разность, рассмотрим следующий способ генерации случайных точек на одной прямой. Случайным образом выберем две точки p и q и координату i и определим x и y следующим образом:
   $x = (p_1, \ldots, p_{i-1}, p_i, q_{i+1}, \ldots, q_m)$, $y = (p_1, \ldots, p_{i-1}, q_i, q_{i+1}, \ldots, q_m)$.

   С вероятностью δ мы выбрали точку p так, что $f(p) = f^*(p)$, и с вероятностью $1 - \delta$ мы выбрали точку q так, что $f(q) \neq f^*(q)$. Если оба эти события произошли, то существует такая координата i, в которой $f(x) = f^*(x)$ и $f(y) \neq f^*(y)$. Для того чтобы это показать, рассмотрим следующую последовательность точек:
   $(p_1, p_2, \ldots, p_m), (q_1, p_2, \ldots, p_m), \ldots (q_1, q_2, \ldots, q_m)$. В точке p значения функций совпадают, а в точке q — нет, значит, найдутся две такие соседние точки в последовательности, что значения функций f и f^* в первой из этих точек совпадают, а в следующей нет. Осталось заметить, что x и y выбираются как две соседние точки. Вероятность выбрать координату, обладающую таким свойством, составляет хотя бы $\frac{1}{m}$.
   Суммарно получаем, $Pr[f(x) = f^*(x)$,$f(y) \neq f^*(y)] \geq \delta(1 - \delta)\frac{1}{m}$.

Пусть значения функций f и f^* не совпали в точках y, z, тогда существует единственный x, в котором значения функций совпадают и тройка (x, y, z) — хорошая. Предположим, что такой x не единственный и существует какой-то x', для которого выполнено все тоже самое, но тогда тройка (x, y, x') тоже хорошая, и значения функций совпадают в двух точках из трех, чего быть не может. Вероятность попасть в такой не превосходит $\frac{1}{|\mathcal{I}|}$.

Таким образом, всю разность можно оценить снизу, как $3(\delta(1-\delta)\frac{1}{m} - \frac{1}{|\mathcal{I}|})$. При достаточно большом размере множества $\mathcal{I}$ эта вероятность $\geq \frac{c'}{n}$.

2. Пусть $\delta \geq \frac{9}{10}$. Будем вести индукцию по m, чтобы показать, что вероятность найти плохую тройку, хотя бы $(1 - \frac{1}{|\mathcal{I}|})^{m-1}\frac{c'}{m} > \frac{c}{m}$. Здесь c' обозначает константу, полученную при доказательстве первого пункта. База индукции при $m = 1$ соответствует тому, что вероятность найти плохую тройку хотя бы c', то есть константа, которую мы можем подобрать так, чтобы $c' < \frac{9}{10}$.

   Для того, чтобы сделать индукционный переход, нам потребуется следующая лемма:

   Лемма 16.5. *Обозначим через f_a функцию f, у которой первая координата результата заменяется на a, аналогично определяется функция f_b, тогда если существуют такие a, b, что $\Delta_{ML}(f_a) < \frac{1}{10}$, а также доля не f-линейных троек среди троек, у которых первая координата хотя бы у одной из точек a, меньше $\frac{1}{3}$, и аналогичное выполнено для b, то $\Delta_{ML}(f) < \frac{9}{10}$.*

   Доказательство. Рассмотрим функции f_a^* и f_b^* и обозначим через $f^* = f_a^*(x) + \frac{x_1 - a}{b-a}(f_b^*(x) - f_a^*(x))$ мультилинейную функцию, которая является одновременно продолжением функций f_a^* и f_b^*. При первой координате, равной a, f^* совпадает с f_a^*, а при первой координате, равной b — с функцией f_b^*. Докажем, что $\Delta(f, f^*) < \frac{9}{10}$, то есть вероятность, что в случайной точке f и f^* совпадут, больше $\frac{1}{10}$. Для этого выберем случайные r и r', которые отличаются только в первой координате, а также рассмотрим точки r_a, r_b. Все эти четыре точки отличаются только по первой координате. Тогда с вероятностью $1 - 2\frac{1}{10} - 2\frac{1}{3} > \frac{1}{10}$ произойдет следующее. Тройки (r_a, r, r') и (r_b, r, r') окажутся хорошими (так как доля плохих троек составляет $\frac{1}{3}$), функция $f(r_a) = f^*(r_a)$ (что то же самое, что f_a совпадет с f_a^* в r_a) и, аналогично, $f(r_b) = f^*(r_b)$. □

Зафиксируем первую координату, вероятность того, что первая координата у всех одинаковая, равна $\frac{m-1}{m}$. Если $\Delta_{ML}(f) > \frac{9}{10}$, то условие леммы нарушается. Отсюда следует, что точек, обладающих свойствами из условия леммы, может быть не больше одной. Эту точку мы рассматривать не будем. Выберем произвольную точку b, в которой нарушены условия леммы. Вероятность попасть в эту точку составляет $1 - \frac{1}{|\mathcal{I}|}$. Теперь рассмотрим три случая, которые соответствуют возможным нарушениям условия леммы:

(a) Пусть $\Delta_{ML}(f_b) > \frac{9}{10}$, тогда по индукционному предположению вероятность найти плохую тройку больше, чем $\frac{m-1}{m}(1-\frac{1}{|\mathcal{I}|})(1-\frac{1}{\mathcal{I}})^{m-2}\frac{c'}{m-1} = (1-\frac{1}{|\mathcal{I}|})^{m-1}\frac{c'}{m}$, получили требуемое.

(b) Пусть $\frac{1}{10} \leq \Delta_{ML}(f_b) \leq \frac{9}{10}$, тогда вероятность найти плохую тройку составляет хотя бы $\frac{m-1}{m}(1-\frac{1}{|\mathcal{I}|})\frac{c'}{m-1} \geq (1-\frac{1}{|\mathcal{I}|})^{m-1}\frac{c'}{m}$ (по первому пункту доказательства этой теоремы).

(c) Пусть доля нелинейных троек $\geq \frac{1}{3}$, значит, вероятность попасть в плохую тройку хотя бы $\frac{1}{3}$, таким образом вероятность найти плохую тройку хотя бы $\frac{m-1}{m}(1-\frac{1}{|\mathcal{I}|})\frac{1}{3} \geq (1-\frac{1}{|\mathcal{I}|})^{m-1}\frac{c'}{m}$.

□

Теперь мы готовы описать тест на мультилинейность:

1. Случайным образом выбираются cm троек и спрашивается значение функции f на каждой из них, таким образом по теореме 16.5 мы получим константную вероятность ошибки.

2. Тест принимает функцию f, тогда и только тогда, когда все тройки f-линейны.