Оглавление

Введение

Одним из стандартных способов задания функций k-значной логики являются поляризованные полиномиальные формы (ППФ), которые также называются обобщенными формами Рида-Мюллера, или каноническими поляризованными полиномами. В ППФ каждая переменная имеет определенную поляризацию. Длиной полиномиальной формы называется число попарно различных слагаемых в ней. Длиной функции f в классе ППФ называется наименьшая длина среди длин всех поляризованных полиномиальных форм, реализующих f. Функция Шеннона $L_k^K(n)$ длины определяется как наибольшая длина среди всех функций k-значной логики в классе K от n переменных, если K опущено, то подразумевается класс ППФ. Практическое применение ППФ нашли при построении программируемых логических матриц (ПЛМ) [1,2], сложность ПЛМ напрямую зависит от длины ППФ, по которой она построена. Поэтому в ряде работ исследуется сложность ППФ различных функций [3–9].

В 1993 В. П. Супрун [3] получил первые оценки функции Шеннона для функций алгебры логики :

$$L_2(n) \geqslant C_n^{[\frac{n}{2}]},$$

$$L_2(n) < 3 \cdot 2^{n-1},$$

где $[a]$ обозначает целую часть a.

Точное значение функции Шеннона для функций алгебры логики в 1995 г. было найдено Н. А. Перязевым [4] :

$$L_2(n) = \left[\frac{2^{n+1}}{3}\right].$$

Функции k-значных логик являются естественным обобщением функций алгебры логики. Для функций k-значной логики верхняя оценка функции Шеннона была получена в 2002 г. С. Н. Селезневой [5] :

$$L_k(n) < \frac{k(k-1)}{k(k-1)+1} k^n,$$

в 2015 году она была улучшена [12]

$$L_k(n) < \frac{k(k-1)-1}{k(k-1)} k^n.$$

При построении ПЛМ рассматривают и другие полиномиальные формы. Например класс обобщенных полиномиальных форм. В классе обобщенных полиномиальных форм, в отличие от класса поляризованных полиномиальных форм, переменные могут иметь различную поляризацию в разных слагаемых. В статье К. Д. Кириченко [6], опубликованной в 2005 г., был предложен метод построения обобщенных полиномиальных форм из которогоследует $L_2^{\text{О.П.}}(n) = O(\frac{2^n}{n})$.

В работах [7, 10] получено, что $L_k^{\text{О.П.}}(n) = O(\frac{k^n}{n})$.

В 2012 г. Н. К. Маркеловым была получена нижняя оценка функции Шеннона для функции трехзначной логики в классе поляризованных полиномов [8]:

$$L_3(n) \geqslant \left[\frac{3}{4} 3^n\right],$$

в [11] эта оценка была достигнута на симметрических функциях.

Основные определения

Пусть $k \geqslant 2$ – натуральное число, $E_k = \{0, 1, \ldots, k-1\}$. Весом набора $\alpha = (a_1, \ldots, a_n) \in E_k^n$ назовем число $|\alpha| = \sum\limits_{i=1}^{n} a_i$. Моном $\prod\limits_{a_i \neq 0} x_i^{a_i}$ назовем соответствующим набору $\alpha = (a_1, \ldots, a_n) \in E_k^n$ и обозначим через K_α. По определению положим, что константа 1 соответствует набору из всех нулей. Функцией k-значной логики называется отображение $f^{(n)} : E_k^n \to E_k$, $n = 0, 1, \ldots$. Множество всех функций k-значной логики обозначим через P_k , множество всех функций k-значной логики, зависящих от переменных $x_1, \ldots, x_n$, обозначим через P_k^n. Функция $j_i(x) = \begin{cases} 1, \text{ если } x = i; \\ 0, \text{ если } x \neq i. \end{cases}$

Если k – простое число, то каждая функция k-значной логики $f(x_1, \ldots, x_n)$ может быть однозначно задана формулой вида

$$f(x_1, \ldots, x_n) = \sum_{\alpha \in E_k^n : c_f(\alpha) \neq 0} c_f(\alpha) K_\alpha \ ,$$

где $c_f(\alpha) \in E_k$ – коэффициенты, $\alpha \in E_k$, и операции сложения и умножения рассматриваются по модулю k. Это представление функций k-значной логики называется ее полиномом по модулю k. При простых k однозначно определенный полином по модулю k для функции k-значной логики f будем обозначать через $P(f)$.

Определим поляризованные полиномиальные формы по модулю k. Поляризованной переменной x_i с поляризацией d, $d \in E_k$, назовем выражение вида $(x_i + d)$. Поляризованным мономом по вектору поляризации δ, $\delta = (d_1, \ldots, d_n) \in E_k^n$, назовем произведение вида $(x_{i_1} + d_{i_1})^{m_1} \cdots (x_{i_r} + d_{i_r})^{m_r}$, где $1 \leqslant i_1 < \ldots < i_r \leqslant n$, и $1 \leqslant m_1, \ldots, m_r \leqslant k-1$. Обычный моном является мономом, поляризованным по вектору $\tilde{0} = (0, \ldots, 0) \in E_k^n$.

Выражение вида $\sum_{i=1}^{l} c_i \cdot K_i$, где $c_i \in E_k \setminus \{0\}$ – коэффициенты, K_i – попарно различные мономы, поляризованные по вектору $\delta = (d_1, \ldots, d_n) \in E_k^n$, $i = 1, \ldots, l$, назовем поляризованной полиномиальной нормальной формой (ППФ) по вектору поляризации δ. Мы будем считать, что константа 0 является ППФ по произвольному вектору поляризации. Заметим, что при простых k для каждого вектора поляризации каждую функцию k-значной логики можно однозначно представить ППФ по этому вектору поляризации [5]. При простых k однозначно определенную ППФ по вектору поляризации $\delta \in E_k^n$ для функции $f \in P_k^n$ будем обозначать через $P^\delta(f)$.

Длиной $l(p)$ ППФ p назовем число попарно различных слагаемых в этой ППФ. Положим, что $l(0) = 0$. При простых k длиной функции k-значной логики в классе ППФ называется величина $l^{\text{ППФ}}(f) = \min\limits_{\delta \in E_k^n} l(P^\delta(f))$.

Функция k-значной логики $f(x_1, \ldots, x_n)$ называется симметрической, если

$$f(\pi(x_1), \ldots, \pi(x_n)) = f(x_1, \ldots, x_n)$$

для произвольной перестановки π на множестве переменных $\{x_1, \ldots, x_n\}$. Множество всех симметрических функций k-значной логики обозначим через S_k. Симметрическая функция $f(x_1, \ldots, x_n)$ называется периодической с периодом $\tau = (\tau_0 \tau_1 \ldots \tau_{T-1}) \in E_k^T$, если $f(\alpha) = \tau_j$ при $|\alpha| = j \pmod T$ для каждого набора $\alpha \in E_k^n$. При этом число T называется длиной периода. Периодическую функцию k-значной логики $f(x_1, \ldots, x_n)$ с периодом $\tau = (\tau_0 \tau_1 \ldots \tau_{T-1}) \in E_k^T$ будем обозначать через $f^{(n)}_{(\tau_0 \tau_1 \ldots \tau_{T-1})}$. Понятно, что такое обозначение полностью определяет эту функцию.

Введем функцию $\text{rol}_i(\alpha) \in E_k^n \times E_k \to E_k^n$, производящую циклический сдвиг вектора α влево. Пусть $\alpha = (a_1, \ldots, a_n)$, тогда $\text{rol}_i(\alpha) = (a_{(1+i) \mod T}, \ldots, a_{(n+i) \mod T})$, где T – длина вектора α.

Функция $\mathrm{ror}_i(\alpha) \in E_k^n \times E_k \to E_k^n$ производит циклический сдвиг вектора α вправо. Пусть T – длина вектора α, тогда $\mathrm{ror}_i(\alpha) = \mathrm{rol}_{(-i) \mod T}(\alpha)$.

Цикличиской матрицей, порожденной ветором $\beta = (b_0,b_1,\ldots,b_{n-1})$, называется матрица вида

$$B = \begin{pmatrix} b_0 & b_1 & \ldots & b_{n-1} \\ b_{n-1} & b_0 & \ldots & b_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ b_1 & b_2 & \ldots & b_0 \end{pmatrix}.$$

Каждая строка, начиная со второй, получается циклическим сдвигом предыдущей вправо на один элемент.

В [9] дается описание быстрого алгоритма построения поляризованного полинома по заданному вектору функции и поляризации d. Данный алгоритм можно использовать для получения периодов, участвующих в разложении периодической симметрической функции $f^{(n+1)}$ с периодом τ.

$$A^{(d)} \cdot F(\tau) = \begin{pmatrix} \tau_0 \\ \tau_1 \\ \vdots \\ \tau_{k-1} \end{pmatrix}, \text{ где}$$

$$f_\tau^{(n+1)} = f_{\tau_{k-1}}^{(n)}(x_{n+1}+d)^{k-1} + f_{\tau_{k-2}}^{(n)}(x_{n+1}+d)^{k-2} + \ldots + f_{\tau_0}^{(n)} \qquad (1)$$

$$A = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & k-1 & -(2^{k-2}) \bmod k & \ldots & -((k-1)^{k-2}) \bmod k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k-1 & k-1 & k-1 & \ldots & k-1 \end{pmatrix}$$

$A^{(d)}$ получается из матрицы A циклическим сдвигом столбцов A влево на d столбцов или построчным применением функции rol_d к матрице A.

$$F(\tau) = \begin{pmatrix} rol_0(\tau) \\ rol_1(\tau) \\ \vdots \\ rol_{k-1}(\tau) \end{pmatrix}$$

$F(\tau)$ – матрица, в которой в i-той строке стоит сдвиго влево периода τ на i элементов.

Например в трехзначной логике для функции с периодом [1,1,2,2] получим

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 2 & 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 2 & 2 & 1 \\ 2 & 2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 0 & 2 & 0 \\ 2 & 1 & 1 & 2 \end{pmatrix}$$

Пусть матрица A состоит из строк a_i, a'_i – строка, полученна из a_i, добавлением нулей справа так, что длина строки a'_i равна T. Тогда матрица A_i состоит из T цикличиских сдвигов вправо строки a'_i:

$$A_i = \begin{pmatrix} ror_0(a'_i) \\ ror_1(a'_i) \\ \vdots \\ ror_{T-1}(a'_i) \end{pmatrix},$$

матрица A_i – циклическая.

Аналогичным образом из матрицы $A^{(d)}$ строятся матрицы $A_i^{(d)}$. Матрица A_0 – единичная матрица размера $T \times T$. Для всех τ_i из формулы (1) верно $\tau_i = A_i \cdot \tau$. Период τ называется вырожденным, если $l(f_\tau^{(n)}) = \bar{o}(k^n)$ при $n \to$

∞. Период τ называется сложным, если $\exists n_0, c > 0 : \forall n \geqslant n_0 \Rightarrow l(f_\tau^{(n)}) \geqslant c \cdot k^n$. Обозначим через $\tilde{n}$ – константный период, вида [n,n,. . . ,n], где $n \in E_k$.

Введем индуктивно множество периодов, порожденных периодом τ за i шагов – $\Pi_i(\tau)$. В данном случае матрицы рассматриваются, как множества строк.

$$\Pi_0(\tau) = \{\tau\}$$

$$\Pi_1(\tau) = \bigcup_{d \in E_k} A^{(d)} \cdot F(\tau)$$

$$\Pi_{i+1}(\tau) = \bigcup_{\theta \in \Pi_i(\tau)} \Pi_1(\theta)$$

Обозначим через $\Pi(\tau)$ такое $\Pi_i(\tau)$, что $\Pi_i(\tau) = \Pi_{i+1}(\tau)$.

Постановка задачи

1. Нахождение периодов, являющихся сложными, в k-значных логиках.
2. Отыскание критериев сложности периодов.
3. Получение нижних оценок длин некоторых симметрических функций k-значных логик в классе ППФ.

Результаты

Теорема 1. *Если* $\tilde{0} \notin \Pi(\tau)$, *то* τ *сложный период.*

Доказательство. Теорема 1 является следствием теоремы 2. □

Теорема 2. *Если* $\tilde{0} \notin \Pi(\tau)$, *то* $\forall \sigma \in \Pi(\tau) \forall n \geqslant n_0 \Rightarrow l(f_\sigma^{(n)}) \geqslant c \cdot k^n$, *где*

T – *длина периода* τ
$k^{n_0} > T > k^{n_0-1}$, *нетрудно заметить, что* $n_0 = \lceil \log_k T \rceil$
$c = \frac{1}{k^{n_0}}$

Доказательство. Доказательство проведем индукцией по n.
Базис индукции:
$l(f_\sigma^{(n_0)}) \geqslant 1 = \frac{1}{k^{n_0}} \cdot k^{n_0}$
Индуктивный переход:
Предположим, что теорема верна для $n \geqslant n_0$, докажем ее для $n+1$.
Для любой поляризации $d \in E_k$ функция $f_\sigma^{(n+1)}$ выражается следующим полиномом:

$$f_\sigma^{(n+1)} = f_{\sigma_{k-1}}^{(n)} \cdot (x_{n+1} + d)^{k-1} + \ldots + f_{\sigma_1}^{(n)} \cdot (x_{n+1} + d) + f_{\sigma_0}^{(n)}, \text{ где}$$

функции $f_{\sigma_i}^{(n)}$ могут быть различными при разных поляризациях d. По предположению индукции $\forall i \in E_k \Rightarrow l(f_{\sigma_i}^{(n)}) \geqslant \frac{1}{k^{n_0}} \cdot k^n$, следовательно $l(f_{\sigma_i}^{(n+1)}) \geqslant k \cdot \frac{1}{k^{n_0}} \cdot k^n = \frac{1}{k^{n_0}} \cdot k^{n+1}$ □

Теорема 3. *Все периоды длины 4 в трехзначной логике, отличные от* $[0,0,0,0]$, $[1,0,1,0]$, $[2,0,2,0]$, $[0,1,0,1]$, $[1,1,1,1]$, $[2,1,2,1]$, $[0,2,0,2]$, $[1,2,1,2]$, $[2,2,2,2]$ *являются сложными.*

Доказательство. Все матрицы $A_0^{(d)}$ являются невырожденными, поэтому из них нельзя получить нулевой вектор умножением на ненулевой.

Рассмотрим матрицы $A_i^{(d)}$, где $d \in [0..2]$, $i \in [1,2]$.

$$\begin{pmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 1 \\ 1 & 0 & 0 & 2 \\ 2 & 1 & 0 & 0 \end{pmatrix}$$

Базисным вектором ядра этой матрицы является вектор $[1, 1, 1, 1]$. Следовательно при умножении этой матрицы на вектор x будет нулевой вектор $\Leftrightarrow x \in \{[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2]\}$.

$$\begin{pmatrix} 2 & 2 & 2 & 0 \\ 0 & 2 & 2 & 2 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 0 & 2 \end{pmatrix}$$

Базисным вектором ядра этой матрицы является вектор $[1, 1, 1, 1]$. Следовательно при умножении этой матрицы на вектор x будет нулевой вектор $\Leftrightarrow x \in \{[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2]\}$.

$$\begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 1 \\ 1 & 0 & 0 & 2 \end{pmatrix}$$

Базисным вектором ядра этой матрицы является вектор $[1, 1, 1, 1]$. Следовательно при умножении этой матрицы на вектор x будет нулевой вектор $\Leftrightarrow x \in \{[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2]\}$.

$$\begin{pmatrix} 2 & 2 & 2 & 0 \\ 0 & 2 & 2 & 2 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 0 & 2 \end{pmatrix}$$

Базисным вектором ядра этой матрицы является вектор $[1, 1, 1, 1]$. Следовательно при умножении этой матрицы на вектор x будет нулевой вектор $\Leftrightarrow x \in \{[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2]\}$.

$$\begin{pmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{pmatrix}$$

Базисными векторами ядра этой матрицы является векторы $[0, 1, 0, 1], [1,0,1,0]$. Следовательно при умножении этой матрицы на вектор x будет нулевой вектор $\Leftrightarrow x \in \{[0, 0, 0, 0],\ [1, 0, 1, 0],\ [2, 0, 2, 0],\ [0, 1, 0, 1],\ [1, 1, 1, 1],\ [2, 1, 2, 1],\ [0, 2, 0, 2],\ [1, 2, 1, 2],\ [2, 2, 2, 2]\}$.

$$\begin{pmatrix} 2 & 2 & 2 & 0 \\ 0 & 2 & 2 & 2 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 0 & 2 \end{pmatrix}$$

Базисным вектором ядра этой матрицы является вектор $[1, 1, 1, 1]$. Следовательно при умножении этой матрицы на вектор x будет нулевой вектор $\Leftrightarrow x \in \{[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2]\}$. □

Следующее утверждение было получено с использованием программы на Sage, листинг приведен на странице **??**.

Утверждение. *Все периоды длины 7 в пятизначной логике, кроме константных, являются сложными.*

Теорема 4. *Произведение цикличиских матриц коммутативно.*

Доказательство. Рассмотрим циклические матрицы размера $n \times n$ A и B, порожденные векторами $(a_0,a_1,\ldots,a_{n-1})$ и $(b_0,b_1,\ldots,b_{n-1})$ соответственно

$$A = \begin{pmatrix} a_0 & a_1 & \ldots & a_{n-1} \\ a_{n-1} & a_0 & \ldots & a_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \ldots & a_0 \end{pmatrix}, B = \begin{pmatrix} b_0 & b_1 & \ldots & b_{n-1} \\ b_{n-1} & b_0 & \ldots & b_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ b_1 & b_2 & \ldots & b_0 \end{pmatrix}$$

и матрицы $C = A \cdot B$ и $C' = B \cdot A$. Докажем, что $c_{ij} = c'_{ij}$. Все операции сложения и вычитания в индексах будут производится по модулю n.

Пусть $j = l - i$, где l – некоторое число из $[0..n-1]$. Рассмотрим i строку матрицы A $(a_{-i},a_{1-i},\ldots,a_{n-1-i})$ и j столбец матрицы B $(b_{l-i},b_{l-1-i},\ldots,b_{l+1-i})$, а также i строку матрицы B $(b_{-i},b_{1-i},\ldots,b_{n-1-i})$ и j столбец матрицы A $(a_{l-i},a_{l-1-i},\ldots,a_{l+1-i})$. При вычислении c_{ij} a_{m-i} будет умножаться на b_{l-m-i}, $\forall m \in [0..n-1]$, а при вычислении c'_{ij} $b_{(l-m)-i}$ будет умножаться на $a_{l-(l-m)-i} = a_{m-i}$. □

Теорема 5. *Если* $\tilde{0} \in \Pi_1(\tau)$, *то* τ *вырожденный период.*

Доказательство.

$$f_\tau^{(n+1)} = f_{\tau_{k-1}^{(d)}}^{(n)} \cdot (x_{n+1} + d)^{k-1} + \ldots + f_{\tau_1^{(d)}}^{(n)} \cdot (x_{n+1} + d) + f_{\tau_0^{(d)}}^{(n)} \qquad (2)$$

$\tau_i^{(d)} = A_i^{(d)} \cdot \tau$. Если существует $\tau_j^{(d')} = \tilde{0}$, то для любых периодов $\tau_i^{(d)} \neq 0$ тоже будет существовать нулевой период в разложении.

Обозначим $\tau_i^{(d)}$ за σ, получим для σ разложение (2). Тогда

$$\sigma_j^{(d')} = A_j^{(d')} \cdot \tau_i^{(d)} = A_j^{(d')} \cdot A_i^{(d)} \cdot \tau \underset{\text{по теореме 4}}{=} A_i^{(d)} \cdot A_j^{(d')} \cdot \tau = A_i^{(d)} \cdot \tilde{0} = \tilde{0}.$$

Это значит, что при любой поляризации будет существовать нулевой период в разложении. Следовательно $l(f_\tau^{(n)}) \leqslant (k-1)^n = \bar{o}(k^n)$.

□

После запуска программы на различных числовых интервалах была выдвинута следующая гипотеза. *Все неконстантные периоды лдины l в k-значной логике будут сложными $\Leftrightarrow$ все простые делители $l > k$.*

Заключение

В работе получены следующие результаты:

1. Доказана теорема о нижней оценки длины функций, задаваемых сложными периодами.

2. Получено достаточное условие сложности периодов.

3. Были найдены все сложные периоды длины четыре в трехзначной логике и длины семь в пятизначной.

4. Написаны программы для работы с периодами в которых реализованы функции:

   - Проверка периода на сложность
   - Построение периодов, порожденных данным
   - Построение специальных матриц, описанных в основных определениях, и функций для работы с ними.
   - Вывод полиномов функций по векторам периодов,
   - Построение ядер, описанных матриц,
   - Проверка выдвинутой гипотезы.

Приложение

В приложении приводится код программ, использованных для получения некоторых результатов. Также программы сами по себе представляют интерес, так как могут быть использованы для более широкого ряда задач, чем требовалось при выполнении курсовой работы.

```
package Polynomial;

use feature 'say';
use List::Util qw(all);
use Data::Printer;
use Data::Dumper;
use Math::Prime::Util qw(
    binomial
    znprimroot
    is_primitive_root
    powmod
    is_prime
    invmod
);

use Storable 'dclone';
use Carp;

use Moose;
use Moose::Util::TypeConstraints;
use MyTypes;
use namespace::clean;

Moose::Exporter->setup_import_methods(
    as_is => [qw(binom_mod)],
);

has 'k' => (
    is      => 'ro',
    isa     => 'Prime',
```

```
    required => 1,
);

has 'd' => (
    is      => 'rw',
    isa     => 'Uint',
    default => 0,
);

has 'str' => (
    is      => 'bare',
    reader  => 'init_str',
    isa     => 'Str',
    required => 1,
);

has 'gen' => (
    is => 'rw',
    isa => 'Str',
);

has 'funcs' => (
    is      => 'ro',
    writer  => '_funcs',
    isa     => 'ArrayRef[Str]',
);

has 'polynomial' => (
    is       => 'rw',
    isa      => 'ArrayRef[HashRef[Uint]]',
    lazy     => 1,
    builder  => '_build_polynomial',
);

sub BUILD {
    my $self = shift;
    $self->polynomial; #build polynomial
}
```

```
use overload
    '""' => \&_print,
    '='  => 'clone',
    '*=' => '_overload_mul_eq',
    '*' => '_overload_mul',
    '+' => '_overload_add',
    '+=' => '_overload_add_eq',
    fallback => 1;

sub _print {
    my $self = shift;
    $self->show(sep => ' + ', noblank => 1);
}

sub to_csv {
    my $self = shift;
    $self->show(sep => ';');
}


sub to_tex {
    my $self = shift;
    $self->show(sep => '+', noblank => 1, mul => '',
        before => '$', after => '$');
}


sub _overload_mul_eq {
    my ($self, $c) = @_;
    my $polynomial = $self->polynomial;
    while (my ($i, $coeff)  = each @$polynomial) {
        for my $f (keys %$coeff) {
            ($polynomial->[$i]{$f} *= $c) %= $self->k;
        }
    }
    return $self;
}
```

```
sub _overload_mul {
    my ($self, $c) = @_;
    my $tmp = dclone $self;
    $tmp *= $c;
    return $tmp;
}

sub _overload_add_eq {
    my ($self, $other) = @_;
    croak "k must be equal in summands" unless $self->k == $other->k;
    my $spolynomial = $self->polynomial;
    my $opolynomial = $other->polynomial;
    $self->_funcs(_union($self->funcs, $other->funcs));
    while (my ($i, $coeff)  = each @$opolynomial) {
        for my $f (keys %$coeff) {
            $spolynomial->[$i]{$f} += $coeff->{$f};
            $spolynomial->[$i]{$f} %= $self->k;
        }
    }
    return $self;
}

sub _overload_add {
    my ($self, $other) = @_;
    my $tmp = dclone $self;
    $tmp += $other;
    return $tmp;
}

sub zeros {
    my $self = shift;
    my $res = [];

    while (my ($i, $x) = each @{$self->polynomial}) {
        push @$res, $i if all {$_ == 0} values %$x;
    }
```

```
    return $res;
}

sub len {
    my $self = shift;
    my $sum = 0;

    for my $h (@{$self->polynomial}) {
        $sum += (grep {$h->{$_} > 0} keys %$h) > 0;
    }

    return $sum;
}

sub clone {
    my $self = shift;
    my $res = dclone $self;
    return $res;
}

sub polarize {
    my ($self, $d) = @_;
    my $k = $self->k;

    my $tmp = $self->clone;
    my $res = _polar($k, $self->polynomial, $d - $self->d, $self->funcs);
    $tmp->polynomial($res);
    $tmp->d($d);
    return  $tmp;
}

sub _build_polynomial {
    my $self = shift;
    my $summandst = resummand();
    my $function = $self->init_str;
    $function =~ tr/ //d;
    $function =~ s/- (?= [a-z] | \( )/-1/gx;
    die "Bad string" unless $function =~ /^ $summandst (\+ $summandst)*+ $/x;
```

```
my ($k, $d) = ($self->k, $self->d);
my $polar_polys = {};
my %all_funcs;

while ($function =~ /$summandst/g) {
    my %h = %+;
    $h{d} //= 0;
    $h{d} =~ s/\s//g;
    $h{d} += 0;
    $h{c} //= 1;
    $h{c} =~ s/\s//g;
    $h{c} += 0;
    $h{pow} //= 1;
    $h{pow} = 0 unless defined $h{x};
    $h{funcs} //= 'Int';
    $h{funcs} =~ tr/*//d;
    delete $h{x};

    my $funcs = {};
    for (split /\+/, $h{funcs}) {
        my ($c, $f) = /(-?\d+)?(.+)/;
        $c //= 1;
        $funcs->{$f} = $c * $h{c} % $k;
        $all_funcs{$f} = 1;
    }

    for my $f (keys %$funcs) {
        no warnings 'uninitialized';
        ($polar_polys->{$h{d}}[$h{pow}]{$f} += $funcs->{$f}) %= $k;
    }
}

$self->_funcs([sort keys %all_funcs]);

while (my ($i, $v) = each %$polar_polys) {
    $polar_polys->{$i} = _polar($k, $v, $d - $i, $self->funcs);
}
```

```
    my $poly = _modulo_sum($k, values %$polar_polys);

    for my $coeff (@$poly) {
        $coeff->{Int} //= 0;
    }

    return $poly;
}

sub show {
    my $self = shift;
    my $k = $self->k;
    my $res = '';

    my %args =  @_;

    if ($args{tex}) {
        $args{mul} //= '';
    } else {
        $args{mul} //= '*';
    }

    $args{sep} = '' if $args{only_functions};
    $args{one_function} //= $args{only_functions};

    my $c_sub = sub {
        $_ = $_[0];

        my $f = $args{one_function};
        tr/()//d;
        my @a = split /\s*\+\s*/;
        if (@a > 2) {
            # Do nothing
            $_ = $_[0];
        } elsif (@a == 1) {
            my $f0 = $self->funcs->[0];
            s/$f0/f0/g;
```

```
            my $f1 = $self->funcs->[1];
            s/$f1/f1/g;
        } else { # @a == 2
            $a[0] =~ s/^(\d)*//;
            my $c1 = $1 // 1;
            $a[1] =~ s/^(\d)*//;
            my $c2 = $1 // 1;
            my $cf = "$f@{[invmod($c1, $k)*$c2 % $k + 1]}";
            $cf = "$c1$args{mul}$cf" if $c1 > 1;
            $_ = $cf;
        }

        $_[0] = $_;
    };

    if ($args{one_function}) {
        $res = $self->_show_polynomial(c_sub => $c_sub, %args);
    } else {
        $res = $self->_show_polynomial(%args);
    }

    return $res;
}

sub _show_polynomial {
    my $self = shift;
    my %args = (
        sep            => ' + ',
        tex            => 0,
        before         => '',
        mul            => '*',
        after          => '',
        noblank        => 0,
        c_sub          => sub {$_[0]},
        default        => '',
        only_functions => 0,
        around         => ['', ''],
        @_
```

```
);

$args{around} = [$args{around}, $args{around}] unless ref($args{around});

my ($k, $d) = ($self->k, $self->d);
my @polynomial = @{$self->polynomial};
my @res;
my @funcs;

while (my ($i,$s) = each @polynomial) {
    my @keys = sort grep {$s->{$_} > 0} keys %{$s};
    my $coeff = $args{default};
    my $blank = not @keys;

    unless ($blank) {
        $coeff = '';
        my @coeffs;
        for my $f (@keys) {
            if ($f eq 'Int') {
                push @coeffs, $s->{$f};
            } elsif ($s->{$f} == 1) {
                push @coeffs, $f;
            } else {
                push @coeffs, "$s->{$f}$args{mul}$f";
            }
        }

        $coeff = join(' + ', @coeffs);
        $coeff = "($coeff)" if @coeffs > 1;

        $args{c_sub}->($coeff);
        if ($args{tex}) {
            $coeff =~ s/([a-w])(\d+)/$1_$2/g;
        }

        if ($args{only_functions} and (not $blank or not $args{noblank})) {
            $coeff =~ s/^\d+\Q$args{mul}\E//;
            $coeff = $args{around}[0] . $coeff . $args{around}[1]
```

```
                unless $blank;
            unshift @res, $coeff;
        }

        $coeff = '' if $coeff eq '1' and $i > 0;
        $coeff = $coeff . $args{mul} if $i > 0 and $coeff;

        if ($i > 1) {
            if ($d > 0) {
                $coeff .= "(x+$d)^$i";
            } else {
                $coeff .= "x^$i";
            }
        } elsif ($i == 1) {
            if ($d > 0) {
                $coeff .= "(x+$d)";
            } else {
                $coeff .= "x";
            }
        }

        if ((not $blank or not $args{noblank}) and not $args{only_functions}) {
          $coeff = $args{around}[0] . $coeff . $args{around}[1] unless $blank;
          unshift @res, $coeff;
        }
    }

    $args{before} . join($args{sep}, @res) . $args{after};
}
__PACKAGE__->meta->make_immutable;

sub _union {
    my %union;
    for my $e (@{$_[0]}, @{$_[1]}) {$union{$e}++}

    return [sort keys %union];
}
```

```
sub _modulo_sum {
    my $k = shift;
    my $res = [];
    for my $v (@_) {
        while (my ($i, $coeff) = each @$v) {
            for my $f (keys %$coeff) {
                ($res->[$i]{$f} += $coeff->{$f}) %= $k;
            }
        }
    }

    return $res;
}

sub _polar {
    my ($k, $polynomial, $d, $funcs) = @_;
    $d %= $k;
    my $a = $polynomial;
    my $res;

    for (my $pow = 0; $pow < $k; ++$pow) {
        no warnings 'uninitialized';
        my $h = $a->[$pow];
        for my $f (@{$funcs}) {
            my $c = 0;
            for (my $i = $pow; $i < $k; ++$i) {
                my $binom = 0 + "@{[binomial($i, $pow) % $k]}";
                $c += $a->[$i]->{$f} * $binom * powmod(-$d, $i-$pow, $k);
            }
            $c %= $k;
            $res->[$pow]->{$f} = $c;
        }
    }

    return $res;
}
```

```
sub resummand {
    my $number = qr/\-?\d++/;
    my $polar = qr/(?:-|\+)?\d++/;

    my $var = qr/(?<x>x)/;
    my $varst = qr{
            \($var(?<d>$polar)\)
        |
            \((?<d>$number)\+$var\)
        |
            $var
    }x;

    my $expst = qr/$varst\^(?<pow>$number)/;

    my $func = qr/[a-w]\d*+/x;
    my $c_func = qr/(?: $number \*? )? [a-w]\d*+/x;
    my $funcst = qr{
            (?<funcs> $c_func )
        |
            \( (?<funcs> $c_func (?: \+ $c_func)*+ ) \)
    }x;

    my $summand = qr/(?: $funcst \*? )? (?: $expst|$varst )/x;
    my $summandst = qr{
            (?: (?<c>$number) \*? )?? $summand
        |
            (?<c>$number)?? $funcst
        |
            (?<c>$number)
    }x;

    return $summandst;
}



sub prime_root {
```

```
    my $k = shift;
    my $res;
    ELEM:
    for my $i (2..$k-1) {
        for my $e (2..$k-2) {
            next ELEM if $i ** $e  % $k == 1;
        }
        $res = $i;
        last ELEM;
    }

    return $res;
}
```

Список литературы

1. Угрюмов Е. П. Цифровая схемотехника. СПб.: БХВ-Петербург, 2004.

2. Sasao T., Besslich P. On the complexity of mod-2 sum PLA's // IEEE Trans.on Comput. 39. N 2. 1990. P. 262–266.

3. Супрун В. П. Сложность булевых функций в классе канонических поляризованных полиномов // Дискретная математика. 5. №2. 1993. С. 111–115.

4. Перязев Н. А. Сложность булевых функций в классе полиномиальных поляризованных форм // Алгебра и логика. 34. №3. 1995. С. 323–326.

5. Селезнева С. Н. О сложности представления функций многозначных логик поляризованными полиномами. Дискретная математика. 14. №2. 2002. С. 48–53.

6. Кириченко К. Д. Верхняя оценка сложности полиномиальных нормальных форм булевых функций // Дискретная математика. 17. №3. 2005. С. 80–88.

7. Селезнева С. Н. Дайняк А. Б. О сложности обобщенных полиномов k-значных функций // Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. №3. 2008. С. 34–39.

8. Маркелов Н. К. Нижняя оценка сложности функций трехзначной логики в классе поляризованных полиномов // Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. №3. 2012. С. 40–45.

9. Селезнева С. Н. Маркелов Н. К. Быстрый алгоритм построения векторов коэффициэнтов поляризованных полиномов k-значных функций // Ученые записки Казанского университета. Серия Физико-математические науки. 2009. 151. №2 С. 147-151.

10. Башов М. А., Селезнев С. Н. О длине функций k-значной логики в классе полиномиальных нормальных форм по модулю k // Дискретная математика. – 2014. – Т. 26, вып. 3. – С. 3-9.

11. Селезнева С. Н. Сложность систем функций алгебры логики и систем функций трехзначной логики в классах поляризованных полиномиальных форм // Дискретная математика. – 2015. – Т. 27, вып. 1. – С. 111 – 122.

12. Балюк А. С., Янушковский Г. В. Верхние оценки функций над конечными полями в некоторых классах кронекеровых форм // Известия Иркутского государственного университета. Серия: Математика. – 2015. – Т.14. – С. 3-17.

13. *SageMath, the Sage Mathematics Software System (Version 7.2)*, The Sage Developers, 2016, `http://www.sagemath.org`.