ТЕХНОЛОГИЧЕСКОЕ ОБРАЗОВАНИЕ

***Очков Валерий Федорович***

# БУЛЕВЫ КУБИКИ

Есть *переменные*, которые могут принимать только два значения: 1 – 0, Да – Нет, True – False. Эти переменные выступают в качестве аргументов *функции*, возвращающей также только два значения [1]. Поговорим об этих переменных и функциях, но не в традиционном стиле классической математики, а отталкиваясь от проблем, возникающих при работе в средах тех или иных языков программирования, да и вообще, при использовании *цифровой* вычислительной техники, в основе которой лежит двоичный «атом» – элементарный элемент памяти, находящийся в одном из *двух* состояний (заряжено – разряжено, намагничено – размагничено и т.д.). Из «атомов» (биты) составляются «молекулы» (байты), которые, в свою очередь, формируют новые «соединения» – переменные, массивы переменных – все то, чем оперируют программисты.

**ФУНКЦИИ ОДНОГО АРГУМЕНТА**

Таких функций *четыре* ($f_1 - f_4$ – см. таблицу 1), но на практике работают только с одной – с $f_1$, которую называют *отрицанием* (*инверсией*).

**ФУНКЦИИ ДВУХ АРГУМЕНТОВ**

Таких функций уже шестнадцать – см. таблицу 2.

Таблица 2 делится на две половинки – на «именную» ($f_1 - f_8$) и безымянную

| $a$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ |
|---|---|---|---|---|
| 0 | **1** | 0 | **1** | 0 |
| **1** | 0 | **1** | **1** | 0 |
| Обозначение | $\neg a$<br>Not($a$)<br>!$a$<br>$\overline{a}$ | $a$ | **1** | 0 |

**Таблица 1.** *Двоичные функции одного двоичного аргумента*

| $a$ $b$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ | $f_{16}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 0 | 0 | **1** | 0 | **1** | **1** | **1** | **1** | 0 | 0 | **1** | **1** | 0 | 0 | **1** | 0 |
| 0 **1** | 0 | **1** | 0 | **1** | 0 | **1** | 0 | **1** | 0 | **1** | **1** | 0 | 0 | **1** | **1** | 0 |
| **1** 0 | 0 | **1** | 0 | **1** | **1** | 0 | 0 | **1** | **1** | 0 | 0 | **1** | **1** | 0 | **1** | 0 |
| **1 1** | **1** | **1** | **1** | 0 | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | **1** | **1** | **1** | 0 |
| Обозначение | $\wedge$<br>*<br>×<br>·<br>И<br>And<br>&<br>&&<br>min | $\vee$<br>+<br>Или<br>Or<br>\|\|<br>max | $\leftrightarrow$<br>$\equiv$<br>$\Leftrightarrow$<br>=<br>Eqv<br>== | $\oplus$<br>$\neq$<br>$\not<>$<br>Xor<br>!= | $\rightarrow$<br>$\supseteq$<br>$\Rightarrow$<br>Imp<br>$\geq$ | $\rightarrow$<br>$\supseteq$<br>$\Rightarrow$<br>Imp<br>$\leq$ | $\downarrow$<br>¬And | \|<br>¬Or | > | < | $\neg a$ | $\neg b$ | $a$ | $b$ | **1** | 0 |

**Таблица 2.** *Двоичные функции двух двоичных аргументов*

($f_9 - f_{16}$). Вот имена первых восьми функций:

- $f_1$ – конъюнкция (логическое умножение)
- $f_2$ – дизъюнкция (логическое сложение)
- $f_3$ – равнозначность (эквивалентность, тождественность)
- $f_4$ – неравнозначность (неэквивалентность, разделительная дизъюнкция, сумма по модулю 2)
- $f_5$ и $f_6$ – импликация ($f_5$ – импликация от $a$ к $b$; $f_6$ – импликация от $b$ к $a$, логическое следование)
- $f_7$ – функция (стрелка) Пирса (функция Вебба, функция Даггера, антидизъюнкция)
- $f_8$ – функция (штрих) Шеффера (антиконъюнкция)

Остальные восемь функций таблицы 2 ($f_9 - f_{16}$, как, впрочем, и три последние функции таблицы 1) не имеют ни имен, ни практического применения. Это либо константы ($f_{15}$ и $f_{16}$), либо функции только одного аргумента ($f_{11} - f_{14}$). Имя, да и то условно, можно дать только функциям $f_9$ и $f_{10}$ – инверсия импликации.

**СЕМЬ КОММЕНТАРИЕВ К ТАБЛИЦАМ 1 И 2**

1. В таблице 1 и таблице 2 собраны имена функций и символы операторов по *всем* двадцати позициям (4+16). Для этого пришлось несколько схитрить – «притянуть» в круг двоичных функций операторы, прямо для этого не предназначенные: «>» (больше) и «<» (меньше), например. Эти операторы, хоть и возвращают двоичный результат, но предназначены для работы с вещественными, а не с двоичными операндами. Этой особенности (ее можно назвать «заглавной» особенностью статьи) мы еще коснемся в пункте 7.

2. В таблице 1 и таблице 2 автор попытался собрать *все* имена функций и символы операторов, использующихся для реализации двоичной арифметики. Список получился, конечно, неполный. Читатель может расширить его примерами из других языков программирования (Pascal, Fortran и др.) и математических программ (Maple, MatLab, Mathematica и др.).

3. Можно отметить *избыточность* значений функции в таблице 1 и таблице 2. В языках программирования программисту предоставляется некий ограниченный набор встроенных двоичных функций и операторов. Вот перечень таких функций и операторов, встроенных в популярные программные среды:

- язык BASIC: Not, And, Or, Xor и Imp
- язык C: !, &, &&, !=, || и = =
- среда Mathcad: ¬, ∧, ∨ и ⊕

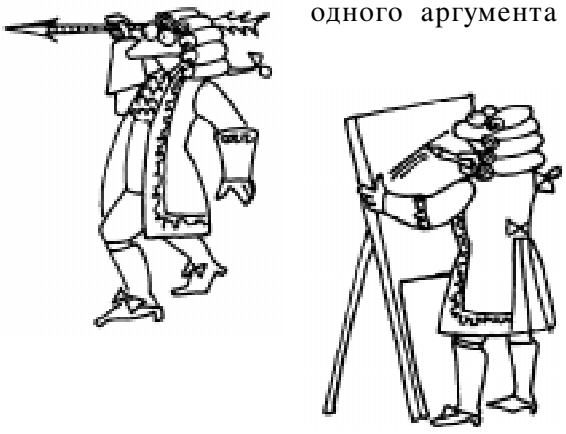
Недостающие двоичные функции (операторы) программист может ввести в программу через механизм пользовательских функций.

Но деление двоичных функций и операторов на основные (базисные) и вспомогательные появилось задолго до компьютеров и «узаконилось» в виде двоичных алгебр (в скобках отмечен их базис):

- алгебра логики (¬, &, ∨, → и ↔)
- булева алгебра (¬, & и ∨)
- алгебра Жегалкина (&, ∨ и ⊕)
- алгебра Пирса (↓)
- алгебра Шеффера (|)

Две последние двоичные алгебры примечательны тем, что в их базисе всего лишь одна двоичная функция, опираясь на которую можно построить все остальные.

На рисунке 1 показан Mathcad-документ, где с опорой на функцию Пирса (еще говорят – *стрелка* Пирса: ↓) построены другие двоичные функции: одна двоичная функция одного аргумента



*Стрелка Пирса... штрих Шеффера...*

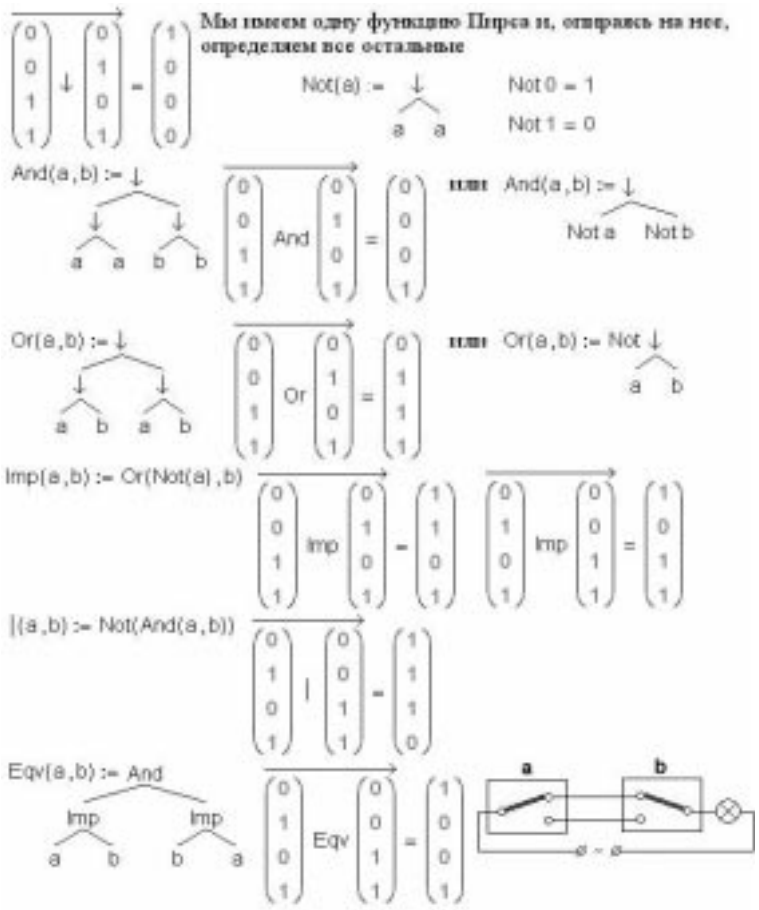



**Рисунок 1.** *Построение двоичных функций с опорой на функцию Пирса*

(отрицание, инверсия – Not) и пять двоичных функций двух аргументов: And, Or, Imp, штрих Шеффера и Eqv. Последние три функции (Imp, штрих Шеффера и Eqv) определены с использованием ранее определенных функций. Это сделано для большей компактности рисунка, но от механизма вложения пользовательских функций (Imp(*a*, *b*) := Or(Not(*a*), *b*, например) можно отказаться и оперировать «для чистоты эксперимента» только функцией (штрихом) Пирса. Двоичные функции часто иллюстрируют электрической цепью: последовательное соединение выключателей – это конъюнкция, а параллельное – дизъюнкция. На рисунке 1 показан менее тривиальный пример – электрический аналог эквиваленции: схема соединения двух выключателей, так чтобы свет независимо зажигался и тушился из двух мест.

*Двоичные функции часто иллюстрируют электрической цепью…*

На алгебру Пирса и алгебру Шеффера возлагались большие надежды в смысле построения компьютера из однотипных элементов. Потом от этой идеи отказались по ряду причин, главная из которых в том, что любой компьютер и так состоит только из однотипных элементов – из транзисторов, объединенных в интегральные микросхемы (*чипы*).

4. Можно отметить *недостаточность* набора математических инструментов, отображенных в таблице 1 и таблице 2 (Если объединить пункты 3 и 4 наших комментариев, то их можно поместить под одним заглавием «Избыточная недостаточность» [2]). Возьмем, например, самую «популярную» функцию двоичной алгебры *конъюнкцию*. Ее столбец в *таблице истинности* (а так называют таблицу 1 и таблицу 2). по идее должен быть такой:

| *a b* | *a* And *b* ($f_1$) |
|---|---|
| 0 – | 0 |
| 0 – | 0 |
| **1** 0 | 0 |
| **1 1** | **1** |

**Таблица 3.** *Уточненная конъюнкция*

Прочерк на месте нуля означает то, что если первый (*a*) аргумент равен нулю,

то *незачем* проверять, чему равен второй аргумент (*b*), и наоборот. Так и поступают, строя некоторые языки программирования – С, например. При программировании в среде языка BASIC условный переход по конъюнкции можно записать так:

If *a* And *b* Then... (1-й способ)

но лучше так:

If *a* Then If *b* Then... или

If *b* Then If *a* Then... (2-й способ)

Второй способ записи позволяет не только ускорять расчеты, но и избегать некоторых ошибок – логическое выражение *b* может иметь смысл, если на альтернативный вопрос *a* дан положительный ответ. Вот типичный пример такой «программистской» ситуации:

If $i > 0$ Then If V($i$) > V($i$-1) Then...

Можно сказать, что в языке BASIC есть две конъюнкции: And и Then If.

Если учитывать то, что в статье рассматривается не какая-то конкретная алгебра двоичных чисел (булева, Пирса, Шеффера и т.д.), а перечисляются возможные двоичные функции двоичных аргументов, то следует признать, что даже одноместных функций должно быть не четыре (см. таблицу 1), а... бесконечное множество. Запрограммированная двоичная функция может, например, возвращать единицу с вероятностью 70%, если ее аргумент равен нулю, и с вероятностью 30%, если аргумент равен 1. В остальных случаях она возвращает нуль.

5. В таблице 1 и таблице 2 мы собрали двоичные функции *одного* (таблица 1) и *двух* (таблица 2) аргументов. Но, возвращаясь к конъюнкции, можно сказать, что эта функция имеет не два, а... *полтора* аргумента – см. таблицу 3.

Такую же *нецелочисленность* (*вещественность*!) или *непостоянство* числа аргументов можно отметить и по другим двоичным функциям.

6. Можно отметить, что в таблицу 2 попали операторы, изначально предназначенные для работы не с двоичными, а с *вещественными* операндами: «>», «<», «≥», «≤», «=» и «≠». Но если принять во внимание тот факт, что множество двоичных чисел входит во множество вещественных чисел, то включение этих операторов в таблицу 2 вполне законно. В этом ряду («>», «<», «≥», «≤», «=» и «≠») также можно отметить и избыточность и недостаточность. С избыточностью все более-менее ясно («больше», например, – это инверсия от «меньше или равно» и т.д.). Недостаточность же можно наблюдать в том, например, что при работе с вещественными переменными, вместо оператора «равно», более уместно использовать оператор «примерно равно», которого нет в списках встроенных. Можно также вспомнить о существовании понятий «намного больше» или «намного меньше». Эти *операторы соотношения* также возвращают двоичные значения, но имеют фактически уже не два, а *три* аргумента (операнда): сравниваемую пару вещественных чисел и некое контекстное представление программиста о том, что такое «примерно» или «намного».

7. Если говорить не о классической двоичной алгебре, а о реальной практике программирования, то следует признать, что переменные, фигурирующие в таблице 1 и таблице 2, могут принимать не два (0 или 1), а *три* значения: 0, 1 и *неопределенно.* Эту особенность мы уже зафиксировали в таблице 3 и в таблице 4, где, вместо конкретных значений аргументов (0-1), стоит прочерк. В языках программирования есть инструменты обработки

| *a b* | *a* Or *b* ($f_2$) |
|---|---|
| 0 0 | 0 |
| 0 **1** | **1** |
| **1** – | **1** |
| **1** – | **1** |

| *a b* | *a* ($f_{11}$) | $\neg a$ ($f_{13}$) |
|---|---|---|
| 0 – | 0 | **1** |
| 0 – | 0 | **1** |
| **1** – | **1** | 0 |
| **1** – | **1** | 0 |

| *a b* | *b* ($f_{12}$) | $\neg b$ ($f_{14}$) |
|---|---|---|
| – 0 | **1** | 0 |
| – **1** | **1** | **1** |
| – 0 | 0 | 0 |
| – **1** | 0 | **1** |

| *a b* | **1** ($f_{15}$) | 0 ($f_{16}$) |
|---|---|---|
| – – | **1** | 0 |
| – – | **1** | 0 |
| – – | **1** | 0 |
| – – | **1** | 0 |

**Таблица 4.** *Двоичные функции полутора, одного и нуля аргументов*

таких «прочерков» в таблицах истинности. Если аргумент двоичной функции не определен, то расчет может либо прерываться сообщением об ошибке, либо идти по третьему пути.

Аргументы двоичных функций могут принимать не два и не три, а... бесконечное множество вещественных значений. Это множество делится на две существенно неравные части: на нуль и на ненуль (¬0, если говорить языком таблицы 1 – это числа, отличные от нуля, которые двоичными функциями воспринимаются как единицы). Бывает и так, что двоичная функция возвращает не только нули и единицы. Вот, например, как работает функция Or в одной из реализаций языка BASIC:

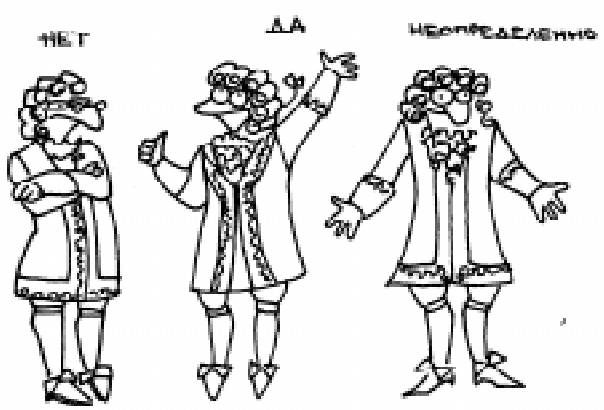
| *a* | *b* | *a* Or *b* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | **¬0** | **1** |
| **¬0** | 0 | **1** |
| **¬0** | **¬0** | **2** |

**Таблица 5.** «*Дизъюнктивная конъюнкция*»

Можно допустить и такую работу расширенного оператора Or:

| *a* | *b* | *a* Or *b* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | **¬0** | **1** |
| **¬0** | 0 | **2** |
| **¬0** | **¬0** | **3** |

**Таблица 6.** *Расширенная конъюнкция (дизъюнкция)*




*Эта... «категоричность»... вступает в противоречие с положениями теории нечетких множеств...*

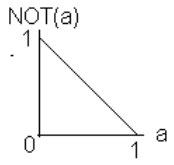
Одно дело, когда первый аргумент (операнд) не равен нулю, а другое – когда второй, и третье – когда оба одновременно.

Подытоживая разбор таблицы 1 и таблицы 2, можно сказать, что описываемые двоичные функции в реальных компьютерных реализациях могут иметь *недвоичные* аргументы и возвращать опять же *недвоичные* результаты. Но особого *недвоичного* смысла в этом нет. Просто по технологическим причинам вещественные переменные в описываемых реализациях языков программирования (BASIC, Mathcad и др.) «по совместительству» выполняют роль двоичных (булевых, логических). При этом *двоичные* (булевы, логические) функции воспринимают свои *вещественные аргументы* «*двоично*»: нуль есть нуль («Нет», «False»), а все остальное единица («Да», «True»).

Эта, можно сказать, «категоричность» описываемых встроенных функций вступает в противоречие с положениями теории *нечетких множеств* (fuzzy sets) и теории *нечеткой логики* (fuzzy logic –[3, 4]). Необходимо, например, статистически обрабатывать на компьютере не только «черно-белые» (двоичные) ответы анкетируемых типа «Да (1)» – «Нет (0)», но и «цветные» (вещественные) ответы: «Да (1)», «Скорее да, чем нет (0.75, например)», «Ни да, ни нет (0.5)», «Скорее нет, чем да (0.25, например)» и «Нет (0)». Если говорить не о статистике, а об электротехнике и вернуться к электрическим цепям, которыми иллюстрируют работу двоичных функций (см. нижнюю часть рисунка 1), то можно упомянуть тот факт, что сейчас в быту получают распространение выключатели, плавно меняющие накал ламп от 100% до нуля. Еще раньше, такие устройства стали применять в театрах и кинозалах. Медики уверяют, что плавный переход от света к темноте через полумрак не портит зрение.

Можно привести еще множество примеров, толкающих к тому, что аргументами функций, перечисленных в таблице 1 и таблице 2, могут и должны быть

не только двоичные, но и вещественные числа, плавно меняющиеся от нуля до единицы. И возвращать функции, перечисленные в таблице 1 и таблице 2, должны вещественные значения, опять же плавно меняющиеся от нуля до единицы. Вот как, например, можно задать «плавную» функцию отрицания:




$\mathrm{Not}(a) := 1-a$

«Плавная» конъюнкция и «плавная» дизъюнкция получаются сами собой, если вспомнить о том, что одно из обозначений конъюнкции – это min (см. столбец $f_1$ в таблице 2), а одно из обозначений дизъюнкции – это max (см. столбец $f_2$ в таблице 2).

$\mathrm{AND}(a, b) := \min(a, b)$ $\mathrm{OR}(a, b) := \min(a, b)$

Несложно задать и другие «плавные» двоичные функции:

$\mathrm{EQV}(a,b) := 1-|a-b|$

$\mathrm{XOR}(a, b) := 1-\mathrm{EQV}(a,b)$

Для иллюстрации функций двух переменных требуются уже не линии, а поверхности. На рисунке 2 показаны, если так можно выразиться, заглавные «булевы кубики» – поверхности «плавных» двоичных функций (AND, OR, EQV, XOR), которые при двоичных аргументах полностью повторяют работу своих традиционных «четких» аналогов (And, Or, Eqv, Xor), но при вещественных аргументах возвращают также вещественные значения, плавно меняющиеся от 0 до 1.

Если вращать кубики, показанные на рисунке 2, то можно увидеть все 16 функций из таблицы 2:

- Вращаем кубик AND вокруг вертикальной оси – получаем функции $f_7$, $f_9$ и $f_{10}$ (три единицы внизу, а одна наверху)
- Вращаем кубик OR вокруг вертикальной оси – получаем функции $f_5$, $f_6$ и $f_8$ (три единицы наверху, а одна внизу)
- Вращаем кубик функции $f_{14}$ вокруг вертикальной оси – получаем функции $f_{11}$, $f_{12}$ и $f_{13}$ (две единицы внизу, а две наверху)
- Переворачиваем вверх дном кубик функции $f_{16}$ (четыре единицы внизу) – получаем функцию $f_{15}$ (четыре единицы наверху).

### ФУНКЦИИ МНОГИХ АРГУМЕНТОВ

На рисунке 3 показано формирование в среде Mathcad «плавной» функции трех аргументов, возвращающей решение жюри присяжных, которые могут выдавать уже не «черно-белые» ответы (виновен – невиновен), а... «цветные»: виновен на 30%, невиновен на 70%, например. В электрическом аналоге машинки для голосования выключатели заменены на реостаты.

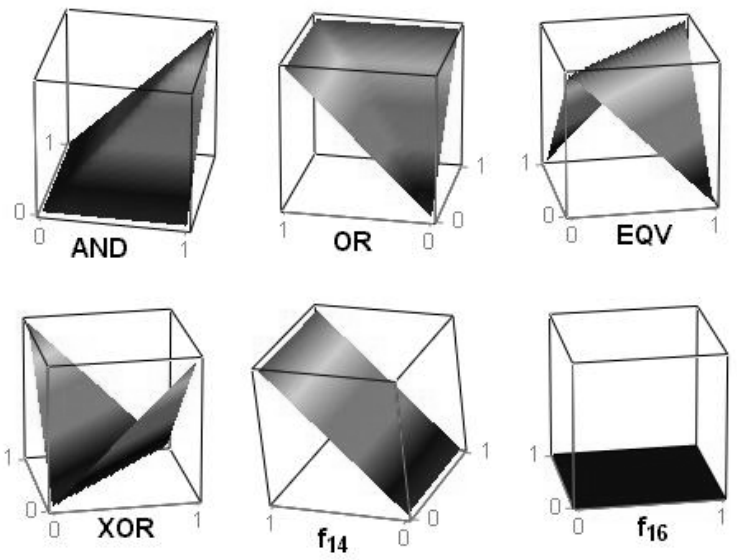
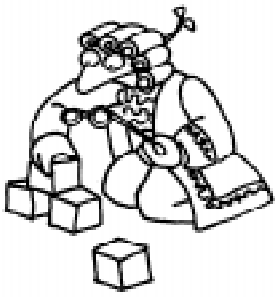



**Рисунок 2.** *Булевы кубики*



Функция Решение, показанная на рисунке 3, при двоичных аргументах возвращает двоичный ответ, а при вещественных – вещественный, естественно. На рисунке 3 показан соответствующий «булев кубик» при $a$=0.3 – мы видим гибрид конъюнкции с дизъюнкцией: мнение одного члена жюри переводит вердикт из области OR в область AND.

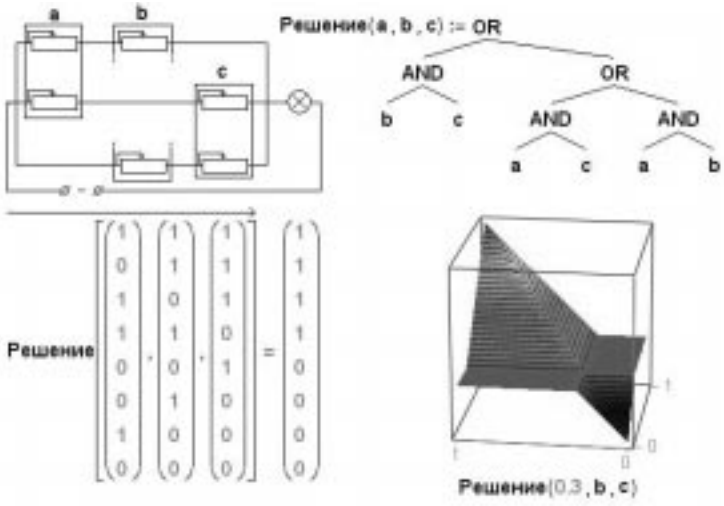



**Рисунок 3.** *Машинка для нечеткого (мягкого, рейтингового) голосования*

**Литература.**

1. Есипов А.С. Логические основы построения и работы компьютеров. Компьютерные инструменты в образовании. № 1, 2000.
2. Очков В.Ф. Принцип неопределенности программирования. КомпьютерПресс, № 7, 1996 (http://twt.mpei.ac.ru/ochkov/IZBYT.htm).
3. Очков В.Ф. Mathcad и нечеткие множества. КомпьютерПресс, № 1, 1998 (http://twt.mpei.ac.ru/ochkov/F_sets.htm)
4. Очков В.Ф. Mathcad и нечеткая логика. КомпьютерПресс, № 8, 1998 (http://twt.mpei.ac.ru/ochkov/F_log.htm)

## НАШИ АВТОРЫ

***Очков Валерий Федорович, кандидат техн. наук, доцент Московского энергетического института (Технического университета).***