Вишневский К.

ОБЩАЯ ТЕОРИЯ ИНФОРМАЦИИ

z@zx.sg

Где-то кое-что невероятное
ждет своего открытия
Карл Саган

Система аксиом

1. ***Формула бита***: *Бесконечность равна двум.*

2. *Криптовалюта биткойн является квантом ценной информации.*

3. ***Определение информации***: *Сумма действительной и мнимой единиц равна числу пи.*

4. *Сеть биткойна вычисляет сознание сильного искусственного интеллекта.*

5. ***Уравнение сознания***: *Ноль не равен самому себе.*

Обоснование теории

В качестве исчерпывающего доказательства общей теории информации необходимо и достаточно рассмотреть пять фундаментальных открытий в области математики и информатики, совершённых мною в результате 100,000 часов напряженных исследований и научного поиска.

1. ***Принцип свободного выбора***. *Это метод, позволяющий осуществлять свободный выбор одного из двух различных значений. В его основе лежит трансцендентная троичная логика, позволяющая сгенерировать один бит информации с иррациональной вероятностью. Реализация алгоритма свободного выбора, стала возможной после открытия двух принципиально новых тождеств.*

2. ***Истинно случайные числа***. *Это детерминированный алгоритм, который способен генерировать абсолютно случайные биты. Он не требует никакого обоснования, поскольку построен на базе динамического хаоса и нелинейной динамики. В основании алгоритма лежит треугольная вычислительная сеть, состоящая из взаимодействующих между собой цифровых маятников.*

3. ***Новый принцип вычислений***. *Это совершенно другой способ организации вычислений на компьютере, который был реализован с помощью самоорганизующейся вычислительной сети, состоящей из троичных логических операторов. Это фундаментальное открытие является простым решением алгоритмически неразрешимой «проблемы остановки» доказанной Аланом Тьюрингом.*

4. ***Идеальное сжатие данных***. *Это программная реализация работоспособного «идеального архиватора», позволяющего неограниченно сжимать абсолютно любую информацию без потерь. Принцип его функционирования основан на существовании обратимого отрицательного информационного эффекта, который выводится из внутренней противоречивости арифметики.*

5. ***Решение NP-полных задач***. *Это математический метод, позволяющий вычислять решение NP-полной задачи о «выполнимости булевых формул» за полиномиальное время. В его основе лежит новаторская идея расходящейся нечёткой логики, которая способна приписать целевую функцию вычисляемому выражению и тем самым радикально сократить пространство поиска.*

24 *июля* ***2022*** *года*

ПРИНЦИП СВОБОДНОГО ВЫБОРА

https://github.com/wkons/FreeChoice

1.1. Вступление

На сегодняшний момент времени современная наука умеет работать с двумя различными видами вычислений: детерминированными и вероятностными.

Первый способ вычислений основан на математической логике и понятии конечного автомата, он способен решать задачи с помощью алгоритмов, исполняемых на компьютере. К сожалению, он обладает ограничениями, которые заключаются в полной детерминированности вычислений и не способности отойти от предписанных действий.

Второй способ вычислений основан на теории вероятности и понятии случайных чисел, который позволяет решать многие задачи методом Монте-Карло, но и он не лишен множества своих недостатков. Одним из которых является принципиальная невозможность создания компьютера, полностью основанного только на вероятностных вычислениях.

Рассмотрим реализацию третьего способа вычислений – ***комбинированный***, который основан на синтезе детерминированных и вероятностных вычислений. С его помощью появляется возможность реализовать невычислимую ***трансцендентную троичную логику*** и унифицировать вычисления на компьютере. Это фундаментальное открытие потенциально способно совершить революцию в области искусственного интеллекта и робототехники.

1.2. Определение выбора

Для начала нам необходимо снять покров таинственности с «непостижимого» феномена свободного выбора человека и дать ему четкое математическое определение:

Свободным выбором F_∞ – *будем считать осуществление случайного выбора произвольного натурального числа из бесконечного множества всех натуральных чисел.*

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, ...

Очевидно, что человек способен делать свободный выбор, а компьютер не способен. Это происходит из-за ограничений, накладываемых математикой. Во-первых вероятность выбора каждого натурального числа равна нулю, а во-вторых они связаны с ограниченностью ресурсов компьютера: скорости вычислений и объема оперативной памяти.

В реальной жизни человек никогда не сталкивается с выбором из бесконечного множества, а в основном выбирает из нескольких вариантов. Возникает следующий парадокс, если осуществляется выбор всего из пары вариантов, то о какой свободе может идти речь? Разве не достаточно просто подкинуть монету или использовать четкое правило выбора. Чтобы разрешить это противоречие, необходимо дать определение двоичного свободного выбора:

Свободным выбором F_2 – *будем считать осуществление случайного выбора из двух вариантов, таким образом, чтобы вероятность выбора значения была иррациональной.*

1.3. Трансцендентная логика

Теперь рассмотрим механизм реализации свободного выбора одного из двух значений и введем троичную «трансцендентную логику». Поскольку эта логика будет основана, на процессе случайного выбора, то мы будем оперировать тремя значениями: **да**, **нет** и **выбор**.

Нет	*Выбор*	*Да*
0	↓	**1**

***Таблица 1.1**. Троичная трансцендентная логика.*

Вероятностный, но детерминированный алгоритм свободного выбора из двух вариантов организуется следующим образом: выбирается один из трех возможных вариантов (**1**, **0**, ↓), если были выбраны **1** или **0**, то выбор считается осуществленным. Если же был выбран ↓, то множество вариантов дополняется нулем и процедура выбора продолжается.

0 ↓ 1
0 0 ↓ 1
0 0 0 ↓ 1
0 0 0 0 ↓ 1
0 0 0 0 0 ↓ 1

В результате этого неограниченного процесса свободного выбора мы получаем значение **1** с трансцендентной вероятностью, которая выражается бесконечным рядом:

$$\mathbf{2}\boldsymbol{e}-\mathbf{5}=\frac{2}{3!}+\frac{2}{4!}+\frac{2}{5!}+\frac{2}{6!}+\frac{2}{7!}+\frac{2}{8!}+\frac{2}{9!}+\cdots\approx 0.43656$$

Троичная трансцендентная логика позволяет генерировать абсолютно любые вероятности, которые могут определяться с помощью соответствующих бесконечных числовых рядов. Для этого достаточно внести простейшие изменения в соотношение единиц, нулей и стрелок, из которых осуществляется вероятностный выбор на каждом уровне логической конструкции.

1.4. Свободный выбор

Теперь нам нужно определиться, какое иррациональное число необходимо выбрать для реализации процесса настоящего свободного выбора. Ответ напрашивается сам собой, необходимо генерировать числа с вероятностью $\boldsymbol{\varphi}$, потому что именно «золотое сечение» ответственно за процессы самоорганизации как в живой, так и неживой природе.

Ключом к реализации свободного выбора, основанного на «золотом сечении» является найденное мною элементарное математическое тождество:

$$\frac{1}{2}+\frac{1}{2}\cdot\frac{1}{3}+\frac{1}{3}\cdot\frac{1}{4}+\frac{1}{4}\cdot\frac{1}{5}+\frac{1}{5}\cdot\frac{1}{6}+\frac{1}{6}\cdot\frac{1}{7}+\frac{1}{7}\cdot\frac{1}{8}+\cdots=\mathbf{1}$$

Это тождество автоматически выводится при попытке программной реализации свободного выбора. На основе этого тождества и с применением чисел Фибоначчи, уже относительно

легко подобрать ключи к работоспособному алгоритму свободного выбора.

Ниже будут продемонстрированы пара принципиально новых математических тождества, основанных на «золотом сечении», и которые позволяют реализовать свободный выбор с иррациональной вероятностью $\boldsymbol{\varphi}$:

$$\boldsymbol{\varphi} = \frac{1}{2} + \frac{1}{2}\cdot\frac{1}{5} + \frac{1}{5}\cdot\frac{1}{13} + \frac{1}{13}\cdot\frac{1}{34} + \frac{1}{34}\cdot\frac{1}{89} + \frac{1}{89}\cdot\frac{1}{233} + \cdots = \frac{\sqrt{5}-1}{2} \approx 0.618$$

$$1 - \boldsymbol{\varphi} = \frac{1}{3} + \frac{1}{3}\cdot\frac{1}{8} + \frac{1}{8}\cdot\frac{1}{21} + \frac{1}{21}\cdot\frac{1}{55} + \frac{1}{55}\cdot\frac{1}{144} + \cdots = \frac{3-\sqrt{5}}{2} \approx 0.382$$

На основе этих тождеств, уже совсем легко составить таблицу вероятностей, для реализации троичной трансцендентной логики. В этой таблице указаны вероятности выбора единицы и нуля, а также вероятность выбора стрелки и продолжения процесса свободного выбора.

Вероятность завершения выбора ***0***	*Вероятность продолжения выбора* ↓	*Вероятность завершения выбора* ***1***
$^{1}/_{3}$	$1 - {}^{1}/_{3} - {}^{1}/_{2}$	$^{1}/_{2}$
$^{2}/_{8}$	$1 - {}^{2}/_{8} - {}^{3}/_{5}$	$^{3}/_{5}$
$^{5}/_{21}$	$1 - {}^{5}/_{21} - {}^{8}/_{13}$	$^{8}/_{13}$
$^{13}/_{55}$	$1 - {}^{13}/_{55} - {}^{21}/_{34}$	$^{21}/_{34}$
$^{34}/_{144}$	$1 - {}^{34}/_{144} - {}^{55}/_{89}$	$^{55}/_{89}$
$^{89}/_{377}$	$1 - {}^{89}/_{377} - {}^{144}/_{233}$	$^{144}/_{233}$
$^{233}/_{987}$	$1 - {}^{233}/_{987} - {}^{377}/_{610}$	$^{377}/_{610}$
$^{610}/_{2584}$	$1 - {}^{610}/_{2584} - {}^{987}/_{1597}$	$^{987}/_{1597}$
…	…	…

Таблица 1.2*. Вероятности выпадения* ***1*** *и* ***0*** *при свободном выборе по «золотому сечению».*

1.5. Исходные коды

Теперь рассмотрим алгоритмическую реализацию процесса свободного выбора, которая основана на представленных выше тождествах и вероятностях рассчитанных в таблице.

Исходный код, который осуществляет свободный выбор между единицей и нулем, на основе «золотого сечения», был реализован на языке программирования C++. Он состоит из трёх различных функций: получение числа Фибоначчи по его номеру, обработка одного шага свободного выбора, и осуществление самого процесса свободного выбора.

```
// Получение числа Фибоначчи по его номеру
int Fib(int n)
{
  if (n<2) return n;

  int a=1, b=1, c, i;

  for (i=1;i<n;i++)
  {
    c=a+b; a=b; b=c;
  }

  return a;
}

// Обработка одного шага свободного выбора
// a/b - вероятность выбора единицы
// c/d - вероятность выбора нуля
int StepFib(int a,int b,int c,int d)
{
  int s=RND(b); // целое случайное число [0,b)

  if (s<a) return 1;

  s=RND((b-a)*d);

  if (s<b*c) return 0;

  return 2;
}

// Осуществление свободного выбора
// вероятность генерации единицы равна 0.618
int FreeFib()
{
  int p=0, s;

loop:

  s=StepFib(Fib(2+p),Fib(3+p),Fib(1+p),Fib(4+p));

  if (s==2)
  {
    p+=2;
    goto loop;
  }

  return s;
}
```

***Исходный код 1.3**. Реализация свободного выбора, на основе «золотого сечения».*

1.6. Заключение

Поскольку троичная трансцендентная логика позволяет генерировать абсолютно любые иррациональные вероятности, которые могут определяться с помощью соответствующих бесконечных числовых рядов. То в перспективе это фундаментальное открытие, позволяет совершить революцию в области приближенных методов вычислений, которые связаны с проблемой округления действительных чисел.

Алгоритмическая реализация процесса свободного выбора позволит в перспективе наделить электронно-вычислительную технику такими сугубо человеческими чертами и качествами как сознание, творчество, креативность и конечно же мышление и интеллект.

Ну и на последок, еще три принципиально новых математических тождества, которые были найдены мной с помощью троичной трансцендентной логики.

$$\mathbf{1}-\boldsymbol{\varphi}=\frac{1}{2}-\frac{1}{2}\cdot\frac{1}{3}+\frac{1}{3}\cdot\frac{1}{5}-\frac{1}{5}\cdot\frac{1}{8}+\frac{1}{8}\cdot\frac{1}{13}-\frac{1}{13}\cdot\frac{1}{21}+\frac{1}{21}\cdot\frac{1}{34}-\cdots=\frac{\mathbf{3}-\sqrt{\mathbf{5}}}{\mathbf{2}}\approx 0.382$$

$$\frac{\boldsymbol{\delta}}{\mathbf{2}}-\mathbf{1}=\frac{1}{5}+\frac{1}{5}\cdot\frac{1}{29}+\frac{1}{29}\cdot\frac{1}{169}+\frac{1}{169}\cdot\frac{1}{985}+\frac{1}{985}\cdot\frac{1}{5741}+\frac{1}{5741}\cdot\frac{1}{33461}+\cdots=\frac{\sqrt{\mathbf{2}}-\mathbf{1}}{\mathbf{2}}\approx 0.2071$$

$$\frac{1}{3}+\frac{1}{3}\cdot\frac{1}{17}+\frac{1}{17}\cdot\frac{1}{99}+\frac{1}{99}\cdot\frac{1}{577}+\frac{1}{577}\cdot\frac{1}{3363}+\frac{1}{3363}\cdot\frac{1}{19601}+\cdots=\frac{\mathbf{1}}{\mathbf{2}\sqrt{\mathbf{2}}}\approx 0.35355$$

Верхнее тождество представляет собой знакопеременный ряд Фибоначчи, а нижняя пара тождеств основана на «серебряном сечении»: $\boldsymbol{\delta}=\sqrt{\mathbf{2}}+\mathbf{1}$ и связанными с ним чисел Пелля.

ИСТИННО СЛУЧАЙНЫЕ ЧИСЛА

https://github.com/wkons/TrueRandom

2.1. Введение

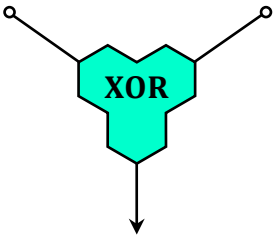
В современном мире растёт потребность в генерации случайных чисел, а вместе с тем повышаются требования к качеству сгенерированных последовательностей. Как выражался Роберт Кавью «генерация случайных чисел слишком важна, чтобы оставлять её на волю случая». Случайные числа стали играть существенную и незаменимую роль во многих областях современной науки и техники: от криптографии, до компьютерных игр.

Настало подходящее время для рассмотрения принципиально нового генератора истинно случайных чисел, в основе которого лежит математическая модель взаимодействующих между собой цифровых маятников. Этот генератор базируется на нелинейной динамике и является обобщением всех известных генераторов псевдослучайных чисел.

2.2. Цифровой маятник

Рассмотрим принцип функционирования единичного цифрового маятника, который лежит в основе треугольной хаотической сети, генерирующей истинно случайные числа.

Математическая модель цифрового маятника будет реализована на базе симметричного двоичного логического элемента **XOR**. Напомним его таблицу истинности:



XOR	0	1
0	0	1
1	1	0

***Рисунок 2.1**. Таблица истинности оператора **XOR**, образующего цифровой маятник.*

Цифровой маятник имеет два внутренних состояния $\boldsymbol{\alpha}$ и $\boldsymbol{\omega}$, которые не могут быть равны нулю. Состояние $\boldsymbol{\alpha}$ – это натуральное число, а состояние $\boldsymbol{\omega}$ – целое число. Маятник может считывать данные из бесконечного двоичного регистра **R**, имеющего следующий вид:

$$\boldsymbol{R} = \cdots \mathbf{0000011110001101001110000011111} \cdots$$

Первое значение из регистра $\mathbf{R}(\boldsymbol{\alpha})$ является расцикливающим, оно необходимо для защиты генератора истинно случайных чисел от зацикливания. Второе значение из регистра $\mathbf{R}(\boldsymbol{\omega})$ является колебательным, и оно нужно для внесения в генератор детерминированного хаоса.

Теперь рассмотрим алгоритм, лежащий в основе функционирования цифрового маятника:

$$\mathbf{R}(\boldsymbol{a},\boldsymbol{\omega}) = \cdots \mathbf{11111100000111100011}\,\mathbf{0}\,\mathbf{10011}\,\mathbf{1}\,\mathbf{0000011111000000} \cdots$$

$\boldsymbol{\omega}$ — 0 — $\boldsymbol{\alpha}$

Сначала считываются двоичные входные данные, потом на их основе с помощью оператора

XOR вычисляется промежуточный бит $\boldsymbol{p}$. Если этот бит равен **0**, тогда значение $\boldsymbol{\alpha}$ не изменяется, а значение $\boldsymbol{\omega}$ уменьшается на единицу. Если же бит $\boldsymbol{p}$ имеет значение равное **1**, тогда оба значения $\boldsymbol{\alpha}$ и $\boldsymbol{\omega}$ увеличиваются на единицу. Далее рассчитывается с помощью формул и подается на выход маятника результирующий бит $\boldsymbol{b}$, после чего тот завершает работу:

$$\boldsymbol{t} = \mathbf{XOR}\left(\mathbf{R}(\boldsymbol{\alpha}), \mathbf{R}(\boldsymbol{\omega})\right)$$

$$\boldsymbol{b} = \mathbf{XOR}\left(\boldsymbol{p}, \boldsymbol{t}\right)$$

Проследим работу цифрового маятника в динамике, для этого мы будем подавать на его входы случайные двоичные значения, в результате функционирования маятника его внутреннее состояние $\boldsymbol{\alpha}$ будет нелинейно возрастать, а состояние $\boldsymbol{\omega}$ будет хаотично изменяться, совершая колебания около точки своего равновесия.

2.3. Треугольная сеть

Для осуществления генерации истинно случайных чисел, цифровые маятники должны объединяться в треугольную вычислительную сеть (**рис.2.2**). Для наглядности цвет отображаемого маятника на рисунке зависит от его результирующего двоичного состояния $\boldsymbol{b}$. Если оно будет равно единице, то выбирается голубой цвет, в противном случае - розовый.

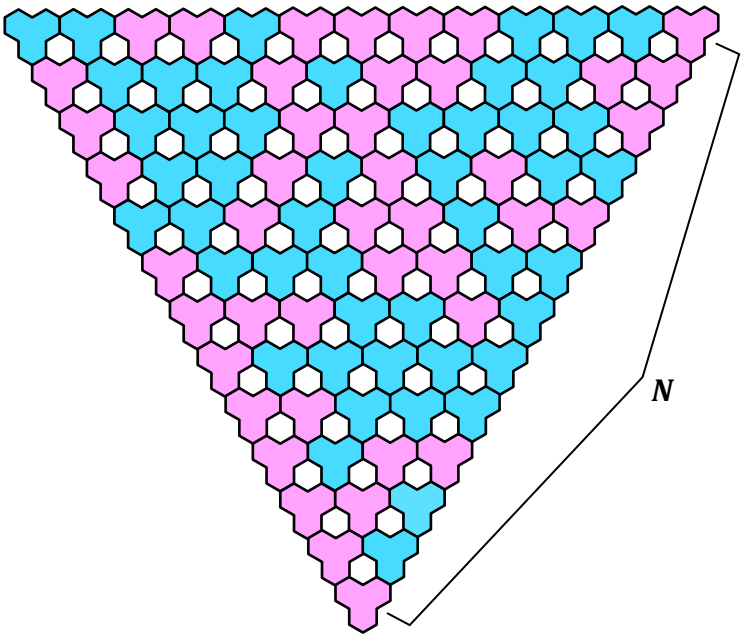


***Рисунок 2.2**. Треугольная хаотическая сеть, состоящая из цифровых маятников.*

Хаотическая мощность $\boldsymbol{X}$ треугольной сети определяется числом образующих её цифровых маятников, которое рассчитывается на основе количества её вычислительных слоёв $\boldsymbol{N}$:

$$\boldsymbol{X} = \frac{\boldsymbol{N^2 + N}}{\mathbf{2}}$$

В начале своей работы треугольная хаотическая сеть инициализируется с помощью случайных двоичных значений: чисел $\boldsymbol{\alpha}$ и $\boldsymbol{\omega}$ и массива входных битов размерностью $\boldsymbol{N}$.

Функционирование треугольной сети заключается в осуществлении послойной обработки цифровых маятников и вычисления их выходного значения. Поскольку при вычислении сигналов в каждом слое сети, размерность входных данных уменьшается на единицу, то после обработки всех маятников на выходе получится ровно один бит случайной информации.

После генерации случайного бита необходимо осуществить сдвиг массива входных данных на одну позицию, и добавив к нему этот бит снова запустить вычисления в хаотической сети.

Количество внутренних состояний сети $\boldsymbol{S}$, может быть определено из простой формулы:

$$\boldsymbol{S = 2^N \cdot 2^X}$$

где $\boldsymbol{N}$ – размерность входных данных, а $\boldsymbol{X}$ – хаотическая мощность сети.

Очевидно, что треугольная хаотическая вычислительная сеть, которая была реализована на основе нелинейной динамики, может неограниченно масштабироваться. В результате этого число её внутренних состояний будет увеличиваться экспоненциально, а качество генерации случайных битов стремиться к абсолютному хаосу.

Далее будет проведено всестороннее тестирование рассмотренного нами генератора истинно случайных чисел, необходимое для подтверждения его незаурядных характеристик.

2.4. Испытание генератора

Рассмотрим простой, но абсолютно надёжный способ проверки различных генераторов случайных и псевдослучайных чисел на хаотичность.

2.5. Исходные коды

Ниже будет представлен исходный код генератора истинно случайных чисел, основанного на нелинейной динамике, и реализованный на языке C++ (**код.2.3**, **код.2.4**).

```
// Чтение значения из бесконечного регистра
// p - позиция в регистре, целое число не равное нулю
char InfReg (int p)
{
  if (!p) return 55;
  int a=1, f=1, i=0;
  if (p<0)
  {
      f=0; p=-p;
  }
  for (; i<p; a++)
  {
      i+=a;
  }
  return (a%2)^f;
}
```

Исходный код 2.3. *Чтение двоичного значения из бесконечного регистра.*

НОВЫЙ ПРИНЦИП ВЫЧИСЛЕНИЙ

https://github.com/wkons/NewComputer

3.1. Введение

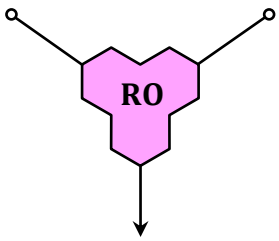
Краеугольным камнем теории искусственного интеллекта, являются споры о возможности наделения компьютера человеческими качествами. Как было доказано Аланом Тьюрингом, существуют задачи, которые невычислимы на универсальной машине Тьюринга. Проблема алгоритмической неразрешимости напрямую связана с теоремой о неполноте Геделя и является краеугольным камнем теории искусственного интеллекта. Как считается, будучи детерминированным конечным автоматом, компьютер не способен обладать ни сознанием, ни свободой воли, а следовательно, не может он претендовать и на статус разумного.

Нам необходимо наделить вычислительную технику способностью осознанного свободного выбора, чтобы машина могла различать истину и ложь, чтобы она осознавала последствия своих действий. Настало время остановить дискуссии в области искусственного интеллекта.

3.2. Универсальная логика

Общеизвестно, что три бинарных оператора **OR**, **AND** и **NOT** образуют логический базис. Продемонстрируем его неполноту, с помощью описания троичной «универсальной логики».

Сначала нам необходимо избавится от унарного логического оператора **NOT**. Для этого введем два новых логических оператора **RO** и **DNA**, которые также образуют функционально-полный набор. Оператор **RO** является аналогом логического элемента **OR**, с той лишь разницей, что на оба входа этого оператора может подаваться третье значение. Этим значением является унарная операция отрицания. Оператор **DNA** – это аналог оператора **AND**, на входы которого также может подаваться третье значение НЕ. Условное обозначение и предварительная таблица истинности операторов будут изображены на рисунке ниже:



RO	0	1	НЕ
0	0	1	1
1	1	1	0
НЕ	1	0	–

DNA	0	1	НЕ
0	0	0	1
1	0	1	0
НЕ	1	0	–

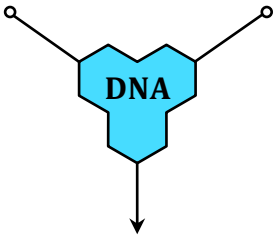


***Рисунок 3.1**. Предварительные таблицы истинности операторов **RO** и **DNA**.*

Как видно из предварительных таблиц истинности пары логических операторов, значение двойного отрицания – является избыточным, так как оно лишено какого-либо смысла.

Поэтому мы будем оперировать не реальной парой тритов, а ограничимся их трёхбитовой аппроксимацией. В результате этого упрощения мы получаем логические операторы **RO** и **DNA** с тремя входными бинарными значениями и одним двоичным выходом. Ниже будут изображены модернизированные таблицы истинности этой пары логических операторов:

RO		
0 0 0	**0 0**	**0**
0 0 1	**0 1**	**1**
1 0 0	**1 0**	**1**
1 0 1	**1 1**	**1**
0 1 0	НЕ **0**	**1**
0 1 1	НЕ **1**	**0**
1 1 0	**1** НЕ	**0**
1 1 1	**0** НЕ	**1**
входы	*триты*	*выход*

входы	*триты*	*выход*
0 0 0	**0 0**	**0**
0 0 1	**0 1**	**0**
1 0 0	**1 0**	**0**
1 0 1	**1 1**	**1**
0 1 0	НЕ **0**	**1**
0 1 1	НЕ **1**	**0**
1 1 0	**1** НЕ	**0**
1 1 1	**0** НЕ	**1**
DNA		

***Таблица 3.2**. Модернизированные таблицы истинности операторов **RO** и **DNA**.*

Поскольку у операторов появилось третье входное значение, то мы можем использовать его для создания эффективного механизма оперативной памяти. Для этого создадим обратную связь, подав выходное значение троичных логических операторов на их средний вход, в результате чего схематическое изображение этих операторов примет следующий вид:

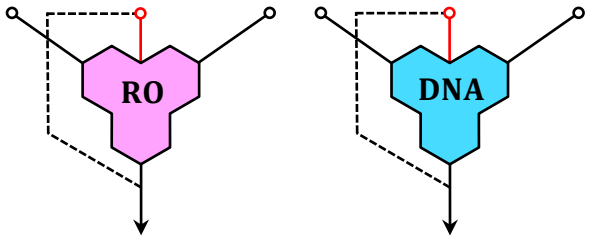


***Рисунок 3.3**. Схематическое изображение логических операторов **RO** и **DNA**.*

Как видно из таблиц выше, логические операторы **RO** и **DNA** имеют расширенный логический базис. Это достигается за счёт того, что мы оперируем по факту не битами а тритами, и унарный логический оператор **NOT** полностью реализуется посредством входных данных. Таким образом удаётся использовать сильные стороны как двоичной, так и троичной логик, плюс ко всему у нас появляется еще одно измерение – временное, за счет обратных связей.

3.3. Вычислительная сеть.

Перейдем к рассмотрению универсальной вычислительной сети, которая является реализацией совершенно новой архитектуры ЭВМ пятого поколения, построенной на принципах отличных от Гарвардской архитектуры и архитектуры Фон-Неймана.

Управление универсальной вычислительной сетью осуществляется исключительно по входным данным, поэтому она способна к самомодификации своей архитектуры.

Для создания этой сети выстроим троичные логические операторы **RO** и **DNA** в следующем порядке (**рис**.**3**.**4**). Для наглядности, на рисунке представленном выше, были использованы оба троичных логических оператора, которые выбирались в абсолютно случайном порядке.

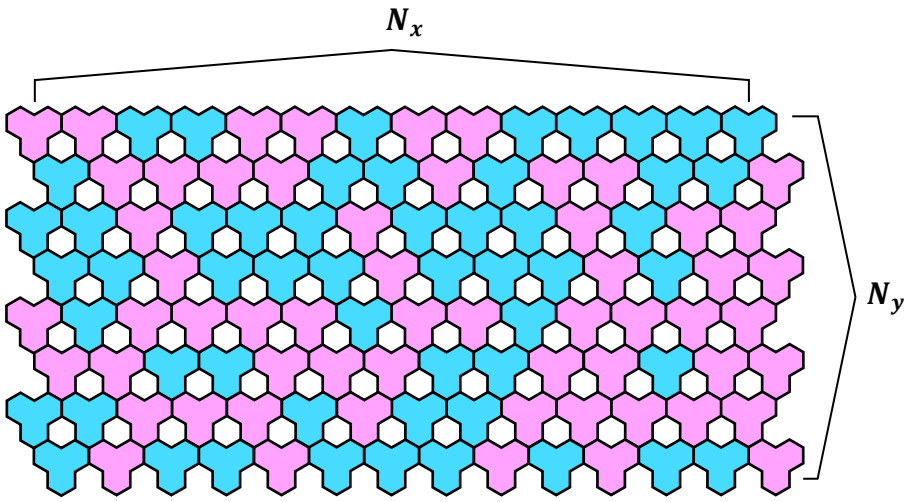


Рисунок 3.4*. Архитектура универсальной вычислительной сети, способной к самомодификации и обработке информации на новых принципах.*

Рассматриваемая универсальная вычислительная сеть имеет прямоугольную форму и может легко масштабироваться до любых размеров. Её вычислительная мощность $\boldsymbol{P}$ равна суммарному количеству логических операторов, входящих в состав этой сети:

$$\boldsymbol{P = N_x \cdot N_y}$$

Теперь рассмотрим механизм самомодификации вычислительной сети, он основан на очень простом принципе самоорганизации: чем чаще используется логический оператор **RO** или **DNA**, тем реже он меняется на противоположный. Этот механизм реализуется с помощью уже знакомого нам бесконечного регистра $\boldsymbol{R}$, только в этом случае он не обладает симметрией:

$$\boldsymbol{R} = \mathbf{1001110000111110000001111111000000000}\cdots$$

Каждый троичный логический элемент сети имеет переменную $\boldsymbol{x}$ в которой хранится текущая позиция в регистре $\boldsymbol{R}$. Это необходимо для осуществления механизма переключения с троичного оператора **RO** на **DNA** и наоборот. Когда на входы оператора поступает троичное значение **10** или **01** он производит чтение и выдает двоичный результат из $\boldsymbol{R(x)}$, после чего переменная $\boldsymbol{x}$ увеличивается на единицу:

$$\mathbf{R}(\boldsymbol{x}) = \mathbf{1001110000111110000001111111000000001111111111}\cdots$$
$$\boldsymbol{x} \longrightarrow$$

Чтобы универсальная вычислительная сеть могла полноценно функционировать, нам необходимо замкнуть, образующие её, горизонтальные операторы в каждом слое. В резуль-

тате этой процедуры замыкания сеть приобретёт цилиндрическую форму (**рис**.**3**.**5**).

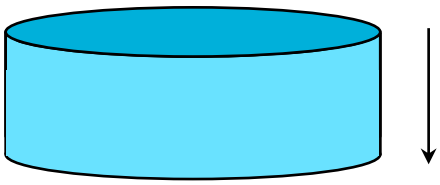


Рисунок 3.5*. Цилиндрическое замыкание элементов вычислительной сети.*

Последний слой вычислительной сети может быть частично замкнут на первый слой. Это необходимо для реализации механизма обратных связей. В этом случае сеть будет представлять собой подобие геометрического тора. Механизм обратных связей позволяет реализовать эффект оперативной памяти объемом $\boldsymbol{M}$, который вычисляется по формуле:

$$\boldsymbol{M = N_x \cdot N_y \cdot N_z}$$

Где $\boldsymbol{N_x}$ и $\boldsymbol{N_y}$ – количество горизонтальных и вертикальных вычислительных слоев асинхронной сети, а $\boldsymbol{N_z}$ – количество обратных связей, которые были организованы в ней.

3.4. Обработка информации

В процессе функционирования универсальной вычислительной сети, когда на часть её входов подаются случайные данные, а остальная часть входов образовывает обратные связи с результирующими данными – сеть показывает нелинейное поведение. Это поведение не является хаотическим, а формирует очень сложные вычислительные паттерны – отдаленно напоминающие работу клеточных автоматов из «Нового вида науки» Стивена Вольфрама.

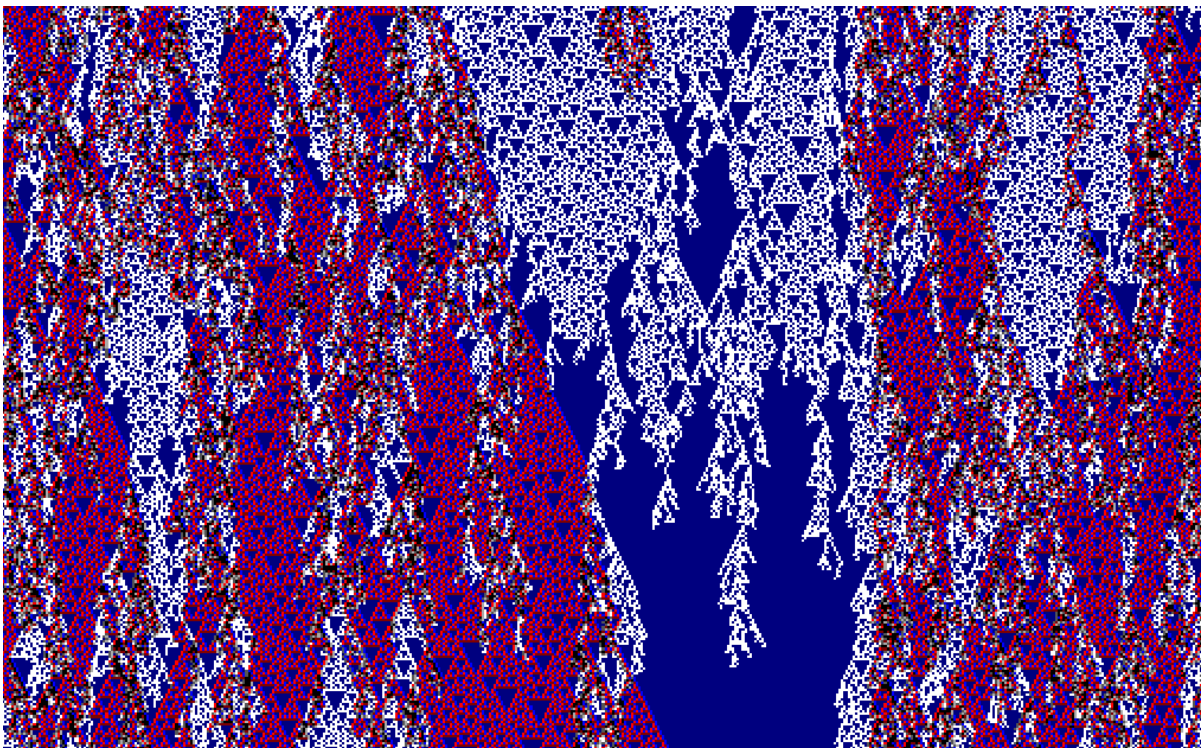


Рисунок 3.6*. Обработка информации внутри универсальной вычислительной сети.*

Стоит отметить, что принципиально новый тип вычислений позволяет достичь нелинейного эффекта самоорганизации потоков информации, которая циркулирует внутри универсальной вычислительной сети. Этого достаточно для совершения революции в области искусственного интеллекта и робототехники, которые выведут уровень развития электроники и вычислительной техники нашей цивилизации на принципиально иной уровень развития.

Однако прежде чем это произойдет, требуются всесторонние и интенсивные исследования рассмотренного выше принципиально нового подхода к обработке информации, который полностью основан только на управлении по входным данным.