БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра математической кибернетики

Курсовая работа

по теме:

**РАЗРАБОТКА ПРИЛОЖЕНИЯ**

**МИНИМИЗАЦИИ ЧАСТИЧНЫХ АВТОМАТОВ**

Выполнил: студент 4-го курса

Соколовский Александр Иванович

Научный руководитель: доцент

Супрун Валерий Павлович

Минск 2016

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

БФ – булева функция

ЧБФ – частичные булевы функции

СБФ – симметрическая булева функция

ТМ – троичная матрица

ДНФ – дизъюнктивная нормальная форма

МАА – минимизация абстрактных автоматов.

ОГЛАВЛЕНИЕ

СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ………………………………………………......3

ВВЕДЕНИЕ…………………………………………………………………………………….4

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ .……………………………………………………………….5

1.1 ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ БУЛЕВЫХ ФУНКЦИЙ...………………………....5

1.2 ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ ГРАФОВ……………………………………….......8

1.3 ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ АВТОМАТОВ..…………….……………………..10

1.4 МИНИМИЗАЦИЯ ЧАСТИЧНЫХ АВТОМАТО……………………………………….12

2 ИНСТРУМЕНТЫ РАЗРАБОТКИ .................... …………………………...……………...15

3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ..……………………………….......17

4 ТЕСТИРОВАНИЕ….....……………………...……………………………………………...19

ЗАКЛЮЧЕНИЕ………………………………………………………………………...….......20

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ………………………………………….21

ПРИЛОЖЕНИЕ ЛИСТИНГ ПРОГРАММЫ ………………………………..………………22

**ВВЕДЕНИЕ**

Курсовая работа посвящена изучению частичных булевых функций (далее ЧБФ), заданных в виде троичных матриц. В работе приводятся методы исследования булевых функций (БФ) на принадлежность классам Поста, а также созданию программного средства, позволяющее определить замкнутость ЧБФ относительно этих классов. Проект является расширением ранее рассмотренного класса задач для полностью определенных функций, основой для решения более сложных классов задач

Оперирование большими объемами информации для современной математики и науки в целом является одной из ключевых особенностей. Так же важной тенденцией в современной науке является и то, что при постоянно увеличивающемся объеме данных так или иначе нужны инструменты для их обработки, позволяющие не только корректно эти данные обрабатывать, но и делать это максимально быстро. Потому скорость обработки информации не менее важна, о чем свидетельствует всевозможное разнообразие программных средств для работы с ней, что естественным образом связано с развитием вычислительной техники, которая зависит от уровня разработки математических моделей дискретных преобразователей информации. В целом, на сегодняшний день булевы функции являются основным аппаратом для построения таких математических моделей. БФ получили широкое распространение благодаря хорошо проработанной теории и своей гибкости при применении в решении самых разнообразных задач. В нашей курсовой работе мы рассмотрим ЧБФ, т.е. находят применение не только в логических системах и при синтезе различного рода схем, но и в диагностике, контроле схем, теории кодирования, конечных автоматов, теории игр и математического моделирования природных процессов.

Логические уравнения представляют собой удобные модели при формализации постановки многих научных и технических задач, а хорошо проработанная теория позволяет найти большое количество инструментов для оперирования с ними, позволяя найти нужные для конкретной задачи.

**1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**1.1 Основные понятия теории булевых функций**

Функции от любого конечного числа двоичных переменных способны принимать лишь два значения: 0 и 1, принято называть булевыми функциями.

Известно, что под дискретным устройством понимается такое

техническое устройство, структура и поведение которого описывается булевыми переменными.

**Определение 1.1.** Переменная *x* , принимающая значение из множества {0,1}, называется *булевой* (*логической, двоичной*).

**Определение 1.2.** Функция F, зависящая от булевых переменных *x1*,*x2*,…,*xn*, и принимающая значения из множества {0,1}, называется *булевой*.

Такая функция обозначается, как *F* = *F* (*x1*, *x2* ,…, *xn*).

**Способы задания булевых функций**

*Табличный способ*

При табличном способе задания булева функция (*БФ*) *F* = *F* (*x1*, *x2* ,…, *xn*) представляется в виде таблицы, в левой части которой, в порядке естественной упорядоченности сверху вниз выписываются наборы значений переменных *x1*, *x2*, … , *xn*, на которых определены данные функции, а в правой части таблицы против каждого набора значений переменных выписываются соответствующие значения функции *F.* Такая таблица называется таблицей истинности булевой функции *F* = *F* (*x1* , *x2* , … , *xn*).

В таблице 1.1 представлена булева функция трех переменных

*F* = *F* (*x1* , *x2* , … , *xn*) посредством таблицы истинности.

Приведенная таблица 1.1 истинности является одномерной. Основной недостаток данной таблицы – число строк, которое увеличивается экспоненциально в зависимости от числа переменных *n*.

*Таблица 1.1 –* Таблица истинности функции

|  |  |  |  |
| --- | --- | --- | --- |
| x1 | x2 | x3 | *F* |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

В теории булевых функций особое значение имеют функции одного и двух переменных.

Значение всех булевых функций одной переменной представлены в таблице 1.2, а функции двух переменных – в таблице 1.3.

*Таблица 1.2* – Булевы функции одной переменной

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *X* | ɡ*1* | ɡ*2* | ɡ*3* | ɡ*4* |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

Из построенных функций функции ɡ*1* и ɡ*2* представляют собою константы 0 и 1. Функция ɡ*2* повторяет значение переменной *x* и потому просто совпадает с ней. Единственной нетривиальной функцией является функция ɡ*3*. Будем называть эту функцию отрицанием переменной *x* и обозначать через *.*

**Теорема 1.1.** Имеется точно различных булевых функций от n переменных (n = 1, 2, …).

Число булевых функций от двух переменных равно, согласно *теореме1.1,*  = = 16. Выпишем свободную таблицу всех этих функций (*Таблица 1.3*).

*Таблица 1.3* – Булевы функции двух переменных

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *x* | *y* | *ƒ1* | *ƒ2* | *ƒ3* | *ƒ4* | *ƒ5* | *ƒ6* | *ƒ7* | *ƒ8* | *ƒ9* | *ƒ10* | *ƒ11* | *ƒ12* | *ƒ13* | *ƒ14* | *ƒ15* | *ƒ16* |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Функции *ƒ1 , ƒ4 , ƒ6 , ƒ11 , ƒ13 , ƒ16 ,* будут вырожденными.Легко увидеть, что *ƒ1 = 0, ƒ4 = x, ƒ6 = y, ƒ11 = , ƒ13 = , ƒ16 = 1.*

Остальные функции будут невырожденными. Введем для них специальные названия и обозначения.

Функция *ƒ2* носит название *конъюнкции*, или *произведения*, или логического «И». Для ее обозначения мы будем пользоваться знаком умножения, который, как и в обычной алгебре, применительно к буквенным выражениям будем опускать: *ƒ2* = *x* × *y* = *x y*. Если значения переменных *x* и *y* функции *ƒ2* понимать, как обычные числа 0 и 1, то значение функции *ƒ2* можно находить, как легко проверить, с помощью обычного перемножения соответствующих значений переменных *x* и *y*.

Функции *ƒ3* и *ƒ5* можно называть отрицаниями импликации. Иногда они называются функциями запрета. Их можно обозначит следующим образом: *ƒ3* = *x,*  , *ƒ5* = *y* .

Функция *ƒ7* носит название *функции неравнозначности*, или суммы по модулю два: *ƒ7* = *x* *y.*

Функция *ƒ8* носит название *дизъюнкции* или *логического «ИЛИ»*.

Для ее обозначения мы будем употреблять знак дизъюнкции, применяющийся в математической логике: *ƒ8* = *x ˅ y*.

Функцию *ƒ9* будем называть *отрицанием дизъюнкции* или  *стрелкой Пирса* и обозначать через *x* ↓ *y* .

Функция *ƒ10* носит название функции равнозначности, или логической эквивалентности, или просто эквивалентности и обозначатся через *x* ~ *y*.

Функция *ƒ12* и *ƒ14* носят название импликации. Для их обозначения употребляют стрелку, соединяющую между собой переменные *x* и *y, расположение в определенном порядке: ƒ12* = *y → x,*

*ƒ14* = *x → y.*

Функция *ƒ15* будем называть *штрихом Шеффера* или *отрицание конъюнкции.* Для ее обозначения используется обычно вертикальная черта, разделяющая переменные: *ƒ15* = *x* | *y.* Можно обозначать эту функцию следующим образом: .

Значение функций одного или двух переменных в общей теории булевых функции состоит в том, что из них может быть построена любая булева функция.

**1.2 Основные понятия теории графов**

Для рассмотрения понятия автомата и его модели необходимо рассмотреть некоторые понятия теории графов. Литература из данного раздела написана из источника [2].

**Определение 1.3.** *Множество* – это совокупность некоторых объектов, объединенное общими свойствами. Множество может содержать n число компонентов. Множество может быть, конечно, или бесконечно.

Термин «граф» появился впервые в книге выдающегося венгерского математика Д. Кёнига в 1936г., хотя начальные задачи теории графов восходят еще к Эйлеру (XVIII в.).

**Определение 1.4.** *Графами* называют взаимосвязь двух множеств состоящих из множества вершин и множества ребер индуцируемых (связанных) между собой.

**Определение 1.5.** *Полный граф* – это граф не имеющий петель, крат –ности ребер, и все его вершины связаны между собой.

**Определение 1.6.** *Неориентированный граф* – граф не имеющий указания направлений ребер, при переходе из одной вершины в другую.

**Определение 1.7.** *Ориентированный (полный) граф* – граф с ребрами указывающими конкретное направление при переходе из одной вершины в другую.

**А**

### В

**С**

**D**

**А**

**B**

**C**

**D**

*Рис. 1.1. Неориентированный граф. Рис. 1.2. Ориентированный граф.*

**Определение 1.8.** *Граф-дерево* – это слабосвязанный граф, у которого если удалить одно ребро, то он распадается на два графа.

Две вершины графа автомата и (исходное состояние и состояние перехода) соединяются дугой (ребром), направленной от к , если в автомате имеется переход из в . Дуге графа автомата приписывается входной сигнал и выходной сигнал , если он определён, и ставится прочерк в противном случае. Если переход автомата из состояния в состояние происходит под действием нескольких входных сигналов, то дуге приписываются все эти входные и соответствующие выходные сигналы.

При описании автомата Мили в виде графа внутри вершины записывается состояние, в которое переходит автомат, а над дугой (ребром) демонстрирующей переход из одного состояния автомата в другое записывается дробь, в числителе которого указывается входной сигнал, а в знаменателе – полученный (выходной) сигнал.

**1.3 Основные понятия теории автоматов**

Литература из данного раздела написана из источников [2–5].

Абстрактные автоматы образуют фундаментальный класс дискретных моделей как самостоятельная модель, и как основная компонента машин Тьюринга, автоматов с магазинной памятью, конечных автоматов и других преобразователей информации.

Модель абстрактного автомата широко используется, как базовая, для построения дискретных моделей автоматов, распознающих, порождающих и преобразующих последовательности символов.

Термин "автомат" используется в двух аспектах. С одной стороны, автомат – устройство, выполняющее некоторые функции без участия человека (например, ЭВМ). С другой стороны, термин "автомат" – математическое понятие и обозначает математическую модель реальных технических процессов.

В общем случае автомат представляется как "черный ящик" и полностью описывается совокупностью следующих шести объектов:

1) множество входных сигналов *Х*;

2) множество выходных сигналов *Y*;

3) множество состояний автомата *А*;

4) начальное состояние автомата *a0* как элемент множества *А*;

5) функция перехода из одного состояния в другое *f (a, x)* ;

6) функция выходов автомата .

**Определение 1.9.** *Автоматом* называется дискретный преобразователь информации, способный принимать состояния из некоторого множества А, переходить под воздействием входных сигналов из одного состояния в другое и выдавать выходные сигналы.

**Определение 1.10.** Если множество состояний автомата, множества входных и выходных сигналов конечны, то автомат *называется конечным автоматом*. Входную информацию принято кодировать символами из входного алфавита *Z*. Выходная информация принято кодируется символами из выходного алфавита *W*.

**Определение 1.11.** *Цифровой автомат* – устройство, предназначенное для преобразования цифровой информации.

В начальный момент времени *t0* автомат находится в состоянии *a0*. В каждыймомент времени *t*, определяемый интервалом дискретности, автомат под воздействием входного сигнала *x(t)* скачкообразно переходит из состояния *ai (t)* в состояние *ai (t+1)* и выдает соответствующий выходной сигнал *y (t).*

Понятие состояние автомата используется для описания систем, выходы которых зависят не только от входных сигналов в данный момент времени, но и от некоторой предыстории, – сигналов, которые поступили на входы системы ранее, т.е. состояние – некоторая память о прошлом.

Цифровые автоматы условно подразделяются на два класса. В синхронных автоматах моменты времени, в которые фиксируются состояния автомата, задаются специальным устройством – генератором синхроимпульсов. В асинхронных автоматах моменты перехода автомата из одного состояния в другое заранее не определены и зависят от каких-то событий.

В зависимости от закона определения выходных сигналов синхронные автоматы делятся на автоматы Мили и автоматы Мура.

Автоматы Мили и автоматы Мура являются автоматами-преобразователями и названы по имени американских ученых, которые впервые начали их изучать.

**Определение 1.12.**  Граф автомата – ориентированный связный граф, вершины которого соответствуют состояниям, а дуги - переходам между ними.

Две вершины графа автомата am и as (исходное состояние и состояние перехода) соединяются дугой (ребром), направленной от am к as , если в автомате имеется переход из am в as. Дуге (am, as) графа автомата приписывается входной сигнал ***х*** и выходной сигнал *у*, если он определён, и ставится прочерк в противном случае. Если переход автомата из состояния am в состояние as происходит под действием нескольких входных сигналов, то дуге (am, as) приписываются все эти входные и соответствующие выходные сигналы.

При описании автомата Мура в виде графа выходной сигнал *у* записывается внутри вершины am или рядом с ней, а входной сигнал *х* над дугой (ребром) демонстрирующей переход из одного состояния автомата в другое.

При описании автомата Мили в виде графа внутри вершины записывается состояние в которое переходит автомат, а над дугой (ребром) демонстрирующей переход из одного состояния автомата в другое записывается дробь, в числителе которого указывается входной сигнал, а в знаменателе – полученный (выходной) сигнал.

**1.4 Минимизация частично определенных автоматов**

Первым этапом метода минимизации частичного автомата, суть которого состоит в построении покрывающего автомата и минимально возможным числом состояний, является нахождение максимальной группировки состояний, а на втором этапе метода на основе найденной группировки строится минимальный покрывающий автомат.

# На первом этапе метода формируется максимальная группировка на основе построенных пар совместимых состояний. Для построения всех таких пар может быть использован метод, применяемый раньше для выявления всех пар эквивалентных состояний полностью определенных автоматов Мили и Мура.

# Выполнение метода минимизации частичного автомата проиллюстрируем на следующем примере.

# Пример 1. Пусть , , и совмещенная таблица переходов и выходов частичного автомата Мили выглядит как в таблице 1.4.

# *Таблица 1.4* – Совмещенная таблица

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

# Требуется построить минимальный покрывающий автомат .

# *Решение.* Первоначально формируется треугольная таблица специального вида. При этом клетка вычеркивается в том случае, когда существует входной сигнал такой, что значения функции выходов

# , определены и .

# В противном случае, в данную клетку заносятся всевозможные пары состояний , где и , при условии, что значения функции переходов , определены и , где .

# Отсюда получаем следующие шесть пар совместимых состояний: , , , и . Отсюда видно, что отношение совместимости, введенное на множестве внутренних состояний , не является транзитивным.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# *Таблица 1.5* – Результирующая треугольная таблица

# Построение максимальной группировки осуществляется на основе просмотра слева – направо результирующей треугольной таблицы. За время этого просмотра образуются некоторые системы подмножеств состояний. В качестве исходной системы берется система, состоящая из множества внутренних состояний .

# Предположим, что после просмотра го столбца таблицы образована система подмножеств , где . При переходе к му столбцу выделяются все состояния, несовместимые с (это легко сделать, исходя из треугольной таблицы). Если подмножество не содержит одновременно и несовместимые с ним состояния, то оно переписывается без изменения. В противном случае из образуется два новых подмножества: одно – путем удаления состояния , другое – путем удаления всех состояний, несовместимых с ним. Проделав эту процедуру для всех и устранив немаксимальные подмножества (содержащихся в других), получаем систему подмножеств, которая является результатом просмотра го столбца. Совокупность подмножеств, образованная после просмотра последнего столбца результирующей таблицы, является максимальной группировкой.

# В рассматриваемом примере максимальная группировка строится следующим образом:

|  |  |
| --- | --- |
| №шага | Система подмножеств |
| 0 |  |
| 1 | , |
| 2 | ,, |
| 3 | ,, |
| 4 | ,,, |
| 5 | ,,,, |

# *Таблица 1.6* – Построение максимальной группировки

# Группа совместимости не является максимальной, поскольку она содержится в группе совместимости .

# Следовательно, максимальная группировка содержит четыре максимальные группы совместимости: ,, и .

# На втором этапе формируется непосредственно минимальный покрывающий автомат , число внутренних состояний которого определяется числом максимальных групп совместимости. Для этого каждой максимальной группе совместимости ставится в соответствие состояния минимального автомата . При этом состояние ставится в соответствие группе совместимости, содержащей состояние . Если таких максимальных групп совместимости несколько, то – любой из них (для однозначности решения будем ставить в соответствие группе с наименьшим номером).

**2 ИНСТРУМЕНТЫ РАЗРАБОТКИ**

Для реализации данного программного средства былв выбрана технология Windows Presentation Foundation (WPF) — система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML и язык программирования C#, так как именно такой набор инструментов позволяет в полной мере выполнить требования, предъявленные к приложению.

Основные преимущества языка C#:

– мощная библиотека базовых;

– полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами;

– является мощным объектным языком с возможностями наследования и универсализации;

– наследник языков C/C++, сохраняя лучшие черты этих популярных языков программирования. Общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от С++ к C#;

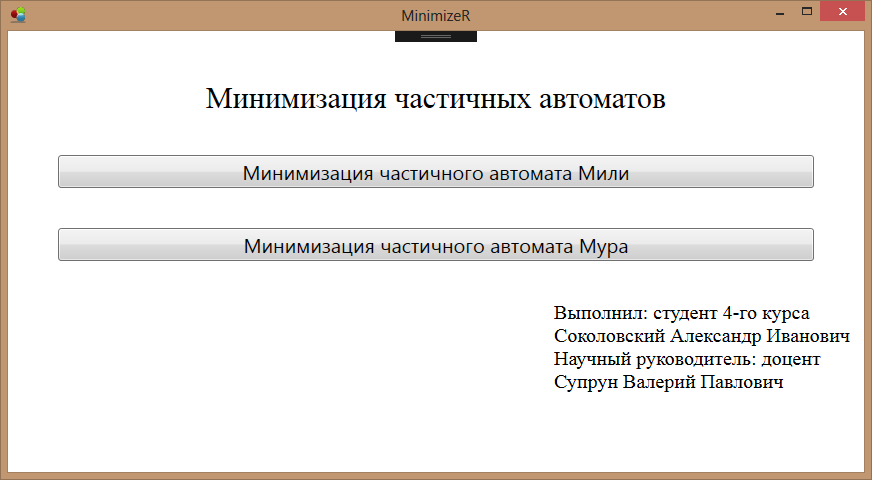
– благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java;

– эффективность кода повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода;

– реализация, сочетающая построение надежного и эффективного кода, является немаловажным фактором, способствующим успеху C# и т.д.

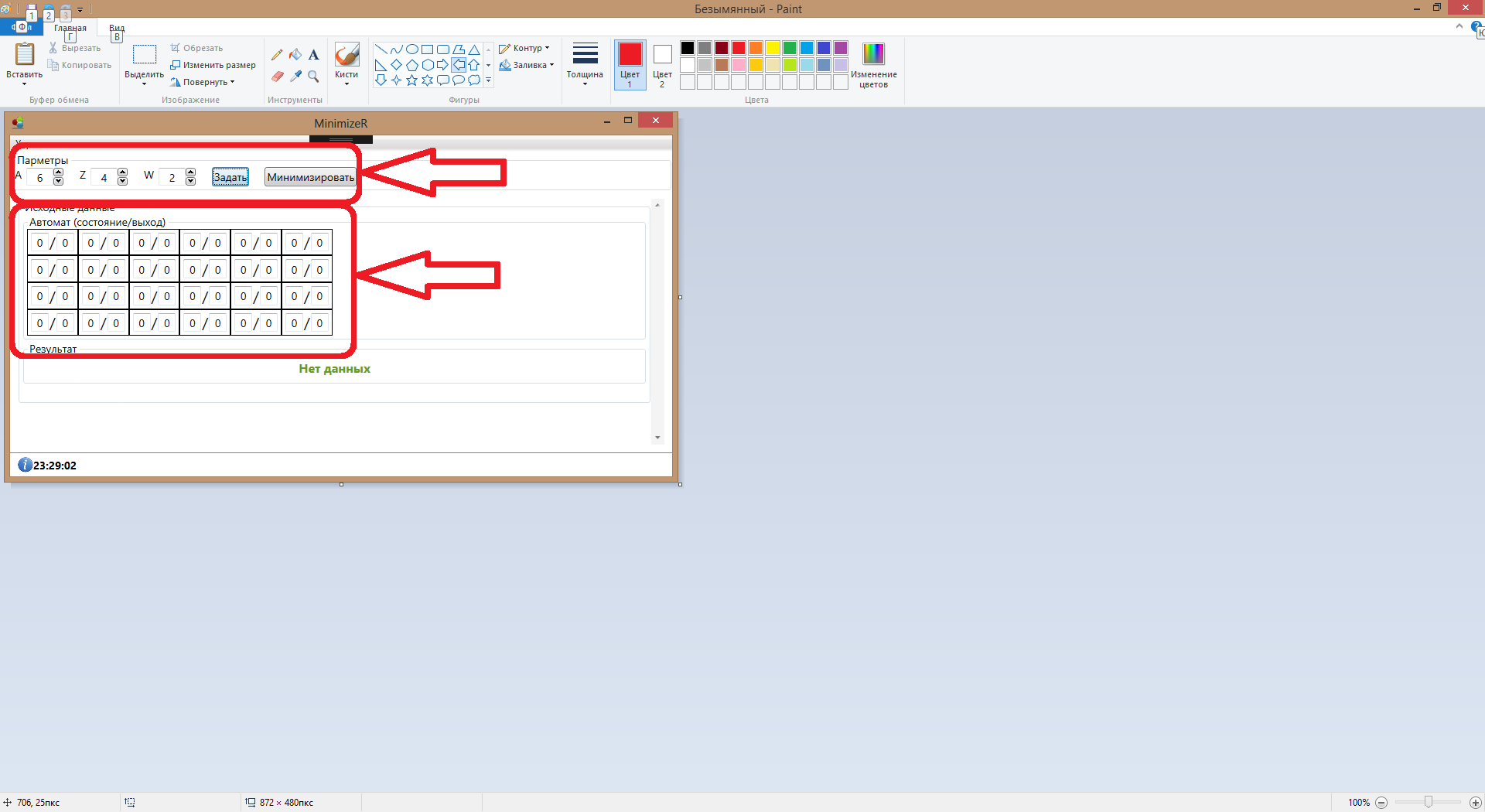
**3 Руководство пользователя программы**

Как результат изученной теории, была реализована программа для минимизации частичных автоматов. Данный программный продукт имеет графический интерфейс и может работать в любой ОС семейства Windows, предъявляя лишь требования к наличию на целевой машине .net framework 3.5 и выше.



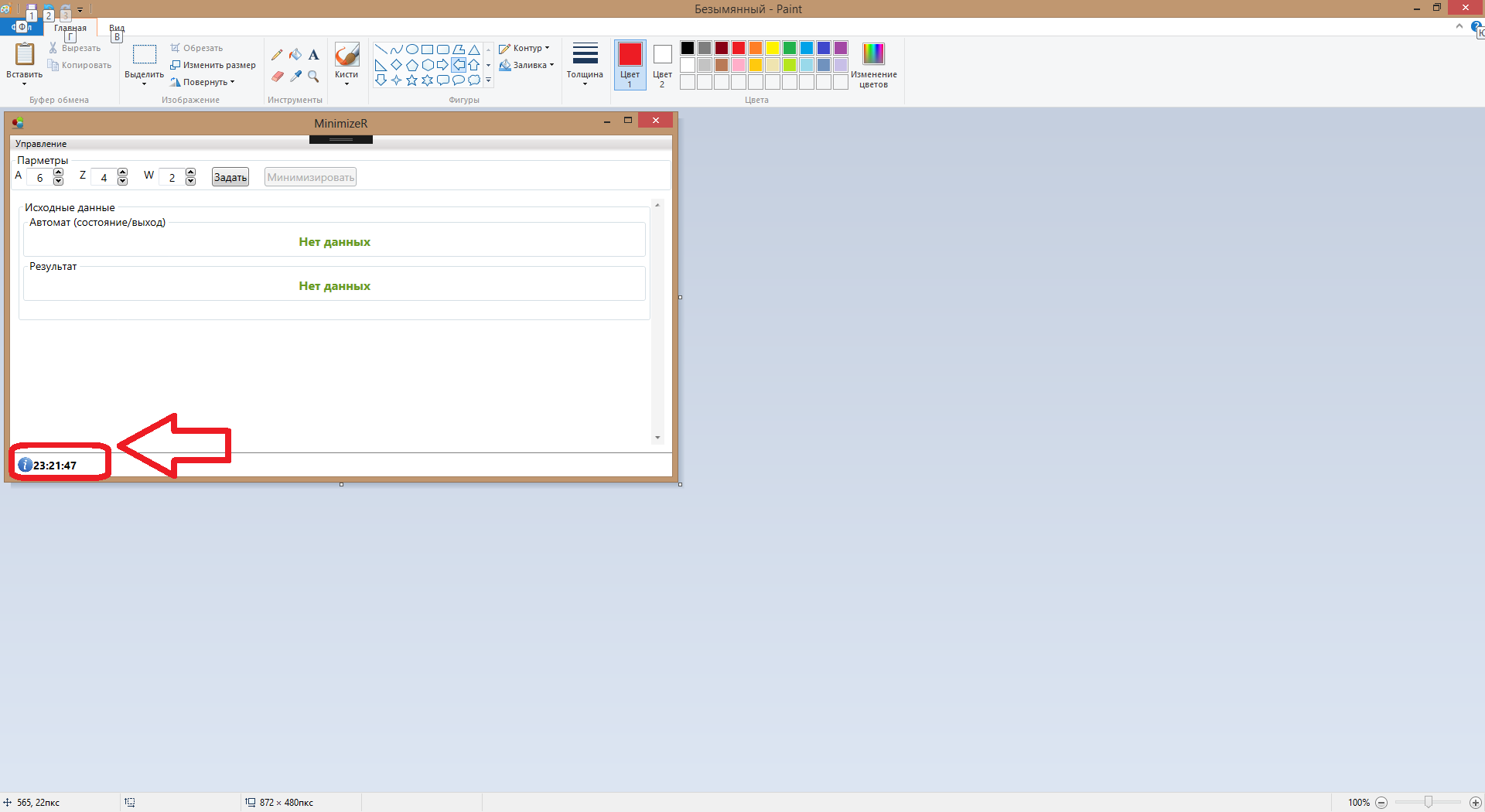
*Рис. 1.3. Окно программы при запуске проекта*

Разработанная программа имеет интуитивно понятный графический интерфейс. Пользователю предоставляется возможность выбора выполняемых операций (минимизация автоматов Мили\Мура) и предлагается ввести параметры исходного автомата.



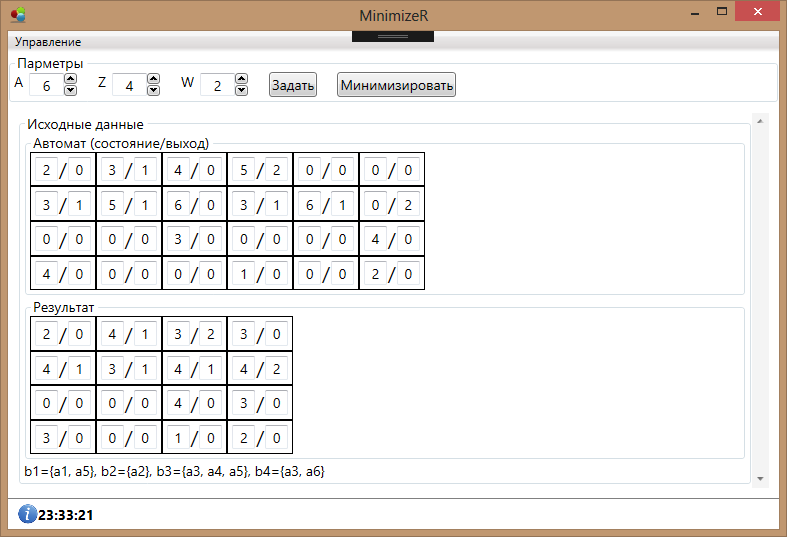
*Рис. 1.4. Задание параметров для таблицы исходного автомата*

Так же программа предусмотрена область уведомлений и система оповещения пользователей об ошибках (выдается звуковой сигнал и поясняющее сообщение в области уведомлений).



*Рис. 1.5. Область уведомалений*

После задания исходных параметров, можно нажать на кнопку «Минимизировать**»**. В результате выполнения будет доступна следующая информация.

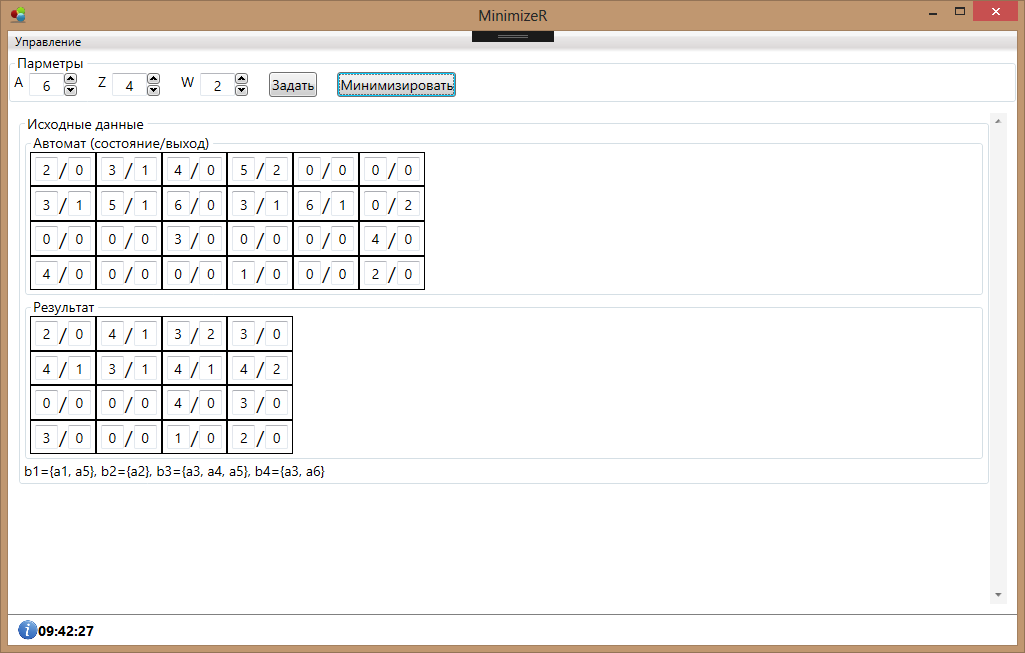


*Рис. 1.6. Результат выполнения минимизации*

Следует отметить, что программа имеет ограничение по числу состояний и выходов (не более 10).

**4 ТЕСТИРОВАНИЕ**

Тестирование приложения было проведено на нескольких наборах данных. На рисунке 1.7 показан результат выполнения программы для примере, приведенного в разделе 1.4.



4.1 Результат работы приложения

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были исследованы ЧБФ, заданные в виде троичных матриц, так же определены дополнительные операции над ними, такие как разность, сумма ТМ и гамма-преобразование над ней, обобщены операции обобщенного склеивания и поглощения на троичные матрицы, способы построения полинома Жегалкина.

В дальнейшем возможно рассмотрение иных методов решения исходной задачи, расширения функциональной части приложения и решения как сопутствующих, так и новых задач. Планируется реализовать различные алгоритмы по обработке троичных матриц в одном программном средстве.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Супрун В.П., Автоматика и вычислительная техника. — Рига: АН Латвийской ССР, 1984.
2. Супрун В.П., Конспект лекций «Прикладная теории автоматов».
3. Супрун В.П., Тихомирова Н.Н. Программное обеспечение ЭВМ. Выпуск 73: Автоматизированный синтез комбинационных сетей из программируемых логических матриц. — Минск: АН БССР. Ин-т математики, Белорусский государственный университет им. В.И. Ленина. — с 10-21.
4. Известия академии наук СССР: техническая кибернетика. – Москва: АН Белоруской ССР, 1983.
5. Поттосин Ю.В., Дискретная математика и теория проектирования цифровых устройств и систем. — Минск: БГУИР, 2007. — с 77-114.
6. [http://mathworld.wolfram.com/](http://mathworld.wolfram.com/BooleanAlgebra.html)
7. <http://satcompetition.org/>

ПРИЛОЖЕНИЕ ЛИСТИНГ ПРОГРАММЫ

public MainWindow()

{

InitializeComponent();

CommandManager.RegisterClassCommandBinding(typeof(MainWindow),

new CommandBinding(CloseWindow, ExecuteCloseWindowCommand));

SetDefaultApplicationState();

}

#region [Properties]

private BackgroundWorkerOnGrid \_worker;

private readonly string \_defaultSound = Application.StartupPath + "\\attention.wav";

private string \_soundPath;

private bool \_isMurMachine = false;

public string SoundPath

{

get { return !string.IsNullOrEmpty(\_soundPath) ? \_soundPath : \_defaultSound; }

set

{

if (\_soundPath != value && !string.IsNullOrEmpty(value))

{

\_soundPath = value;

}

}

}

#endregion

#region [Methods]

private void SetDefaultApplicationState()

{

ValidationState = true;

SetValidationMessage(string.Empty);

}

private void SetValidationMessage(string message)

{

ValidationMessage = string.Format("{0} {1}", DateTime.Now.ToLongTimeString(), message);

}

/// <summary>

/// Проверка правильности задания функции

/// </summary>

/// <returns></returns>

private bool ValidateFunction()

{

if (false) //!(DNFFunction0.Count != countIn0 || DNFFunction1.Count != countIn1))

{

SetValidationMessage("Функция не задана.");

ValidationState = false;

return ValidationState;

}

// и корректность задания матриц (не пересекаются ли они)

/\*DNFFunctionIntersection = DNFFunction0.Intersection(DNFFunction1);

if (DNFFunctionIntersection.Count > 0)

{

SetValidationMessage("Пересечение ЧБФ-матриц не пусто. Проверьте корректность задания функции.");

ValidationState = false;

return ValidationState;

}\*/

return true;

}

/// <summary>

/// Выдать звуковое сообщение

/// </summary>

/// <param name="path"></param>

private static void MakeAlert(string path)

{

try

{

var player = new SoundPlayer(path);

player.Play();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private BoolMatrix CreateTable(int width, int heigh)

{

var result = new BoolMatrix();

for (var i = 0; i < heigh; i++)

{

result.Add(new BoolVector(width));

}

return new BoolMatrix(result);

}

#endregion

#region [Commands]

/// <summary>

/// Закрыть окно. В качестве параметра передается окно, которое необходимо закрыть.

/// </summary>

public static RoutedUICommand CloseWindow { get; } = new RoutedUICommand("CloseWindow", "closewindow",

typeof(MainWindow));

/// <summary>

/// Закрыть окно.

/// </summary>

/// <param name="sender"></param>

/// <param name="e">В качестве параметра обязано передаваться окно для закрытия.</param>

public static void ExecuteCloseWindowCommand(object sender, ExecutedRoutedEventArgs e)

{

var window = e.Parameter as Window;

window?.Close();

}

#endregion Commands

#region [Handlers]

private void btnMinimaze\_OnClick(object sender, RoutedEventArgs e)

{

if (\_isMurMachine)

{

for (int i = 0; i < Machine.Count; i++)

{

for (int j = 0; j < Machine[i].Count; j++)

{

Machine[i][j].Output = Vector[i].Output;

}

}

}

var result = new List<List<List<GroupItem>>>();

for (var i = 0; i < StateCount; i++)

{

var subResult = new List<List<GroupItem>>();

for (var j = 0; j < StateCount; j++)

{

if (i != j && i < j && Machine[i].IsCover(Machine[j]))

{

var items = Machine[i].GetIntersection(Machine[j]);

subResult.Add(new List<GroupItem>(items));

}

else

{

subResult.Add(new List<GroupItem>());

}

}

result.Add(subResult);

}

bool hasChanged;

do

{

var resultCopy = result.Select(t => new List<List<GroupItem>>(t.Select(x => x.ToList()))).ToList();

hasChanged = false;

for (var i = 0; i < result.Count; i++)

{

for (var j = 0; j < result[i].Count; j++)

{

for (var k = 0; k < result[i][j].Count; k++)

{

var groupItem = result[i][j][k];

if (groupItem.First != 0 && groupItem.Second != 0 &&

result[groupItem.First - 1][groupItem.Second - 1].Count == 0)

{

resultCopy[i][j] = new List<GroupItem>();

hasChanged = true;

}

}

}

}

result = resultCopy;

} while (hasChanged);

var compatibleState = new List<GroupItem>();

for (var i = 0; i < result.Count; i++)

{

for (var j = 0; j < result[i].Count; j++)

{

for (var k = 0; k < result[i][j].Count; k++)

{

if (compatibleState.Count(item => item.First == i + 1 && item.Second == j + 1) == 0)

{

compatibleState.Add(new GroupItem(i + 1, j + 1));

}

}

}

}

///////////////////////

var maxGroup = new List<List<int>>();

var tempMaxGroup = new List<List<int>>();

var stateArray = new List<int>();

for (var i = 1; i <= StateCount; i++)

{

stateArray.Add(i);

}

maxGroup.Add(stateArray);

tempMaxGroup.Add(stateArray);

//сначала бежим дпо всем состояниям-столбцам

List<int> removable;

for (var i = 1; i <= StateCount; i++)

{

var one = new List<int>();

var two = new List<int>();

removable = new List<int>();

//проходим по всем множествам в максимальной группировке

for (var j = 0; j < maxGroup.Count; j++)

{

var currentMaxGroup = maxGroup[j];

if (currentMaxGroup.All(item => item != i)) continue;

// получаем возможные пары для ТЕКУЩЕГО стэйта

var statePair = new List<GroupItem>();

for (var index = 0; index < currentMaxGroup.Count; index++)

{

var t = currentMaxGroup[index];

if (i < t) statePair.Add(new GroupItem(i, t));

}

var notСompatible =

statePair.Where(

item =>

!compatibleState.Any(

compItem => compItem.First == item.First && compItem.Second == item.Second));

if (notСompatible.Any())

{

removable.Add(j);

var exclude = notСompatible.Select(item => item.Second);

one = currentMaxGroup.Select(item => item).Where(item => !exclude.Contains(item)).ToList();

two = currentMaxGroup.Select(item => item).ToList();

two.Remove(i);

tempMaxGroup.Add(one);

tempMaxGroup.Add(two);

}

}

for (var j = removable.Count - 1; j >= 0; j--)

{

tempMaxGroup.RemoveAt(removable[j]);

}

// получили максимальную группировку (ещё не приведенную!!!)

maxGroup = tempMaxGroup.Select(t => new List<int>(t.ToList())).ToList();

}

removable = new List<int>();

for (var j = 0; j < maxGroup.Count; j++)

{

for (var k = 0; k < maxGroup.Count; k++)

{

if (j == k) continue;

var isIncluded = maxGroup[j].Where(item => !maxGroup[k].Any(compItem => compItem == item));

if (!isIncluded.Any() && !removable.Contains(j))

{

removable.Add(j);

}

}

}

removable.Sort();

for (var j = removable.Count - 1; j >= 0; j--)

{

maxGroup.RemoveAt(removable[j]);

}

var first = maxGroup.First(item => item.Contains(1));

var other = maxGroup.Where(item => !item.Equals(first));

var b = new List<List<int>> { first };

b.AddRange(other);

var states = new List<List<List<int>>>(b.Count);

var outs = new List<List<List<int>>>(b.Count);

for (var i = 0; i < b.Count; i++)

{

var statesList = new List<List<int>>(InputsCount);

var outsList = new List<List<int>>(InputsCount);

for (var j = 0; j < InputsCount; j++)

{

statesList.Add(new List<int>());

outsList.Add(new List<int>());

}

states.Add(statesList);

outs.Add(outsList);

}

for (var i = 0; i < b.Count; i++)

{

var bi = b[i];

for (var j = 0; j < bi.Count; j++)

{

var vector = Machine[bi[j] - 1];

for (var k = 0; k < vector.Count; k++)

{

var tempState = (int)vector[k].State;

var tempOut = (int)vector[k].Output;

if (!states[i][k].Contains(tempState) && tempState != 0)

{

states[i][k].Add(tempState);

}

if (!outs[i][k].Contains(tempOut))

{

outs[i][k].Add(tempOut);

}

}

}

}

MachineOut = new Matrix();

for (var i = 0; i < b.Count; i++)

{

var listVector = new List<ItemValue>();

for (var j = 0; j < InputsCount; j++)

{

var bState = 0;

for (var k = 0; k < b.Count; k++)

{

if (states[i][j].Count > 0 && states[i][j].All(item => b[k].Contains(item)))

{

bState = k + 1;

continue;

}

}

var bOut = outs[i][j].Max(item => item);

listVector.Add(new ItemValue(bState, bOut, b.Count, OutputsCount));

}

MachineOut.Add(new Vector(listVector, i + 1));

}

BSet = "";

for (var i = 0; i < b.Count; i++)

{

BSet += "b" + (i + 1) + "={";

for (var j = 0; j < b[i].Count; j++)

{

BSet += "a" + b[i][j];

if (j != b[i].Count - 1) BSet += ", ";

}

BSet += "}";

if (i != b.Count - 1) BSet += ", ";

}

}

private void BtnCreateTable\_OnClick(object sender, RoutedEventArgs e)

{

Machine = new Matrix(StateCount, InputsCount, OutputsCount);

Vector = new Vector(OutputsCount, StateCount, OutputsCount, 0);

IsMachineInit = true;

SetDefaultApplicationState();

}