Министерство образования и науки Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра программирования

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р техн.наук, профессор
_____ А.Ю. Матросова
«_5_» _июня_ 2018 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

ПОИСК УСТАНОВОЧНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ ДЛЯ ПОСЛЕДОВАТЕЛЬНОСТНОЙ СХЕМЫ С ИСПОЛЬЗОВАНИЕМ SAT-РЕШАТЕЛЕЙ

по основной образовательной программе подготовки магистров
по направлению подготовки 01.04.02 Прикладная математика и информатика
и направленности (профилю) подготовки
«Математическое и программное обеспечение вычислительных машин и компьютерных сетей»

Пашкова Наталья Николаевна

Научный руководитель ВКР
доцент, канд.техн.наук
_____ С.А. Останин
«_4_» _июня_ 2018 г.

Автор работы
студент группы №_1162м_
Пашкова Н.Н. Н.Н. Пашкова

Томск – 2018

Министерство образования и науки Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра программирования

УТВЕРЖДАЮ
Руководитель ООП
д-р техн.наук, профессор
_____ А.Ю. Матросова
«5» февраля 2018 г.

ЗАДАНИЕ

по подготовке ВКР магистра

студенту: Пашковой Наталье Николаевне группы № 1162м

1. Тема ВКР «Поиск установочной последовательности для последовательностной схемы с использованием SAT-решателей»

2. Срок сдачи студентом выполненной ВКР:
а) на кафедре 4 июня 2018
б) в ГЭК 7 июня 2018

3. Исходные данные к работе
Задачи. Поиск кратчайшей установочной последовательности для последовательностной схемы посредством сведения к задаче выполнимости КНФ разрешения.
Цели. Изучение существующих методов построения установочных последовательностей и алгоритмов упрощения КНФ. Разработка приложения для поиска установочной последовательности, относительно оригинального метода.
Язык программирования C#.

4. Краткое содержание работы:
Обзор существующих методов построения установочных последовательностей;
Исследование алгоритмов упрощения КНФ;
Реализация приложения, для поиска установочной последовательности.

5. Указать предприятие, организацию, по заданию которого выполняется работа:
Кафедра программирования НИ ТГУ

6. Перечень графического материала (с точным указанием обязательных чертежей)
Иллюстрации к тексту – 18

7. Дата выдачи задания «5» февраля 2018 г.

Руководитель ВКР (Научный руководитель ВКР)
Доцент, канд. техн. наук _____ С.А. Останин
должность, место работы *подпись* *инициалы, фамилия*

Задание приняла к исполнению 5 февраля 2018 Пашкова Н.Н.
дата, подпись студента

РЕФЕРАТ

Выпускная квалификационная работа состоит из 47 страниц, 18 рисунков и 7 таблиц.

Цель работы – изучение существующих методов построения установочных последовательностей и алгоритмов упрощения КНФ; разработка приложения для поиска установочной последовательности, относительно оригинального метода.

Задачи – поиск кратчайшей установочной последовательности для последовательностной схемы посредством сведения к задаче выполнимости КНФ разрешения.

В процессе работы был использован язык программирования C#. Для реализации приложения использовалась среда разработки Microsoft Visual Studio 2017.

Разработанное приложение используется для поиска установочной последовательности.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5
1 Постановка задачи и определение целей 6
2 Обзор существующих методов построения установочных последовательностей 9
2.1 Поиск установочной последовательности 9
2.2 Метод регулярных безусловных установочных экспериментов 10
2.3 Метод построения установочной последовательности для конечных автоматов с помощью прямого различающего дерева 11
2.4 Задача выполнимости 12
2.4.1 Задача выполнимости булевых формул 12
2.4.2 Метод поиска кратчайшей установочной последовательности посредством сведения к выполнимости КНФ 13
2.4.3 Оригинальный метод поиска кратчайшей установочной последовательности 16
2.4.4 Результаты SAT Competition 2017 19
3 Алгоритмы упрощения КНФ 21
3.1 Методы, ориентированные на произвольные КНФ 21
3.1.1 Метод резолюции 21
3.1.2 Алгоритм минимизации функции component-wise quadratic (CQ) 22
3.1.3 Horn минимизация путем итеративного разложения 24
3.2 Методы, ориентированные на специфику КНФ 27
3.2.1 Идея метода упрощения КНФ разрешения 27
3.2.2 Упрощение КНФ разрешения оригинального метода 28
3.3 Выводы 28
4 Описание приложения 30
4.1 Обзор использованных технологий 30
4.1.1 Ubuntu 30
4.1.2 C Sharp 30
4.1.3 Visual Studio 2017 31
4.1.4 SAT-решатели 32
4.2 Диаграмма классов 33
4.3 Berkeley Logic Interchange Format (BLIF) 35
4.4 Построение КНФ разрешения 36
4.5 Преобразование КНФ разрешения к сертифицированному формату DIMACS CNF 37
4.6 Упрощение КНФ разрешения для блоков комбинационной логики комбинационного эквивалента 38
4.7 Проверка на корректность установочной последовательности 42
4.8 Выводы 42
ЗАКЛЮЧЕНИЕ 45
ЛИТЕРАТУРА 47

ВВЕДЕНИЕ

Сложность проектирования сверхбольших интегральных схем (СБИС) значительно возросла за последние годы. В связи с этим усложнилась задача обеспечения правильного функционирования СБИС. Большая часть времени теперь тратится на разработку тестов, которые обеспечивают быстрое обнаружение неисправностей, возникших в процессе производства схемы.

Тестирование комбинационных схем выполняется проще чем тестирование последовательных схем с памятью, так как поведение таких схем зависит от набора состояний элементов памяти, перед подачей на входы тестовых воздействий.

Методы тестирования разделяются на встроенные и внешние. Наиболее перспективным представителем встроенного метода является Built-In Self-Test (BIST) или встроенное самотестирование. Однако при таком подходе к тестированию не все неисправности из рассматриваемого класса могут быть обнаружены, кроме того, для реализации этого подхода требуется дополнительная аппаратура. В ситуации, при которой требуется большое покрытие неисправностей тестами, используются методы внешнего тестирования. Метод внешнего тестирования основан на генерации наборов тестовой последовательности, обнаруживающей заданную неисправность. Рассматриваются синхронные логические схемы, начальное состояние которых известно. Множество тестовых наборов для заданной неисправности строится по комбинационной составляющей синхронной схемы. Обычно находятся все тестовые наборы. Тестовые наборы представляют собой полные состояния, содержащие входную составляющую и внутреннее состояние схемы. Последовательность входных векторов (установочная последовательность), переводящая схему из начального состояния в некоторое внутреннее состояние из заданного множества, представленного тестовыми наборами, является искомой тестовой последовательностью. Именно эта последовательность позволяет обнаружить заданную неисправность. В работе [1] поиск установочной последовательности сведен к операциям над ROBDD-графами, извлекаемыми из комбинационной составляющей синхронной схемы. В данной работе предлагается альтернативный подход, заключающийся в сведении к булевой выполнимости конъюнктивной нормальной формы (КНФ), строящейся для комбинационного эквивалента, состоящего из нескольких комбинационных составляющих.

1 Постановка задачи и определение целей

Последовательностная схема представляется в виде двухблочной структуры:

- блок комбинационной логики;
- блок триггеров.

Рассматривается случай, когда схема является синхронной, элементы памяти – D-триггеры или триггеры задержки. D-триггер запоминает состояние входа и выдает его на выход. Набор состояний, в которых находится элемент памяти, называется состоянием схемы. Комбинационные элементы могут быть представлены в виде простых вентилей типа И, ИЛИ, И-НЕ, ИЛИ-НЕ, или элементы из некоторой библиотеки, реализующие более сложные булевы функции.

Блок комбинационной логики – это комбинационная схема, выходы которой являются входами блока триггеров. Выходы триггеров – входы комбинационного блока схемы в следующем такте работы схемы. Структура последовательностной схемы представлена на рисунке 1.

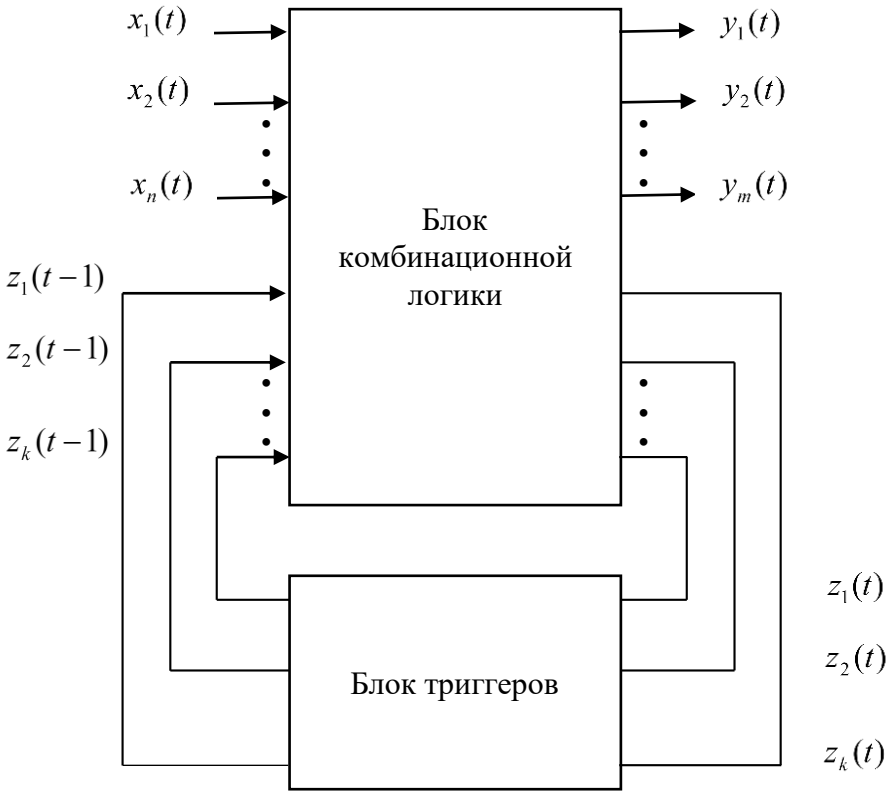


Рисунок 1 – Структура последовательностной схемы с памятью

Комбинационный логический блок имеет два типа входных полюсов:

- внешние – значения определяются сигналами извне;

- внутренние – значения определяются сигналами с выходных полюсов триггеров и задают состояние элементов памяти.

Комбинационный логический блок имеет два типа выходных полюсов:

- внешние – значения передаются во внешнее окружение последовательностной схемы;
- внутренние – значения передаются на блок памяти и определяют функции возбуждения триггеров.

Последовательностная схема состоит из $n+k$ входных переменных $x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_k$ и k выходных переменных $y_1, y_2, \ldots, y_k$, которые определяют k функций возбуждения триггеров. Входных полюсов n, выходных полюсов триггеров k. Причем значения выходных переменных комбинационной схемы являются значениями входных переменных $z_1, z_2, \ldots, z_k$ на следующем такте работы.

Рассмотрим синхронную схему, которая состоит из: n – входных полюсов и k – D-триггеров. Обозначим через $x^i = \left(x_1^i, x_2^i, \ldots, x_n^i\right)$ булев вектор значений входных сигналов, подаваемых на входы схемы в момент времени i.

Последовательность входных воздействий $X = (x^1, x^2, \ldots, x^p)$ длины p называется установочной последовательностью, если состояние, которое определяется состоянием всех триггеров и в которое переходит схема после подачи на ее входы этой последовательности, определяется однозначно вне зависимости от того, в каком состоянии схема была перед началом эксперимента.

Установочная последовательность наименьшей длины называется кратчайшей. При подаче установочной последовательности на входы схемы все элементы памяти устанавливаются в некоторые известные состояния.

Задача заключается в поиске кратчайшей установочной последовательности для последовательностной схемы посредством сведения к задаче выполнимости КНФ разрешения, эффективное решение которой может быть достигнуто с помощью некоторых Satisfiability-решателей (SAT-решатели). Функцией разрешения элемента, который реализует функцию $y = f(z_1, z_2, \ldots, z_n)$, называется функция $\varphi(y, f) = y \sim f(z_1, z_2, \ldots, z_n)$. КНФ разрешения элемента – это представление функции $\varphi(y, f)$ в форме КНФ. Перед проверкой на выполнимость исходную КНФ разрешения необходимо сократить.

Цель данной работы заключается в изучении существующих методов построения установочных последовательностей и алгоритмов сокращения КНФ, а также реализация приложения, которое будет позволять выполнять операции:

- преобразования файла формата «*.blif» в нужную форму КНФ разрешения для каждого блока комбинационной логики комбинационного эквивалента, представленную в виде таблицы;
- упрощения КНФ разрешения;
- преобразования КНФ разрешения к сертифицированному формату DIMACS CNF (или «*.cnf»);
- проверка на выполнимость с помощью SAT решателей (Riss7 или candy [2]);
- нахождение кратчайшей установочной последовательности с помощью SAT решателя;
- проверка на корректность установочной последовательности.

2 Обзор существующих методов построения установочных последовательностей

2.1 Поиск установочной последовательности

Определен конечный автомат $M = (I, O, S, \delta, \lambda)$, где I – множество входных символов, O – множество выходных символов, S – множество состояний, $\delta : S \times I \to S$ – функция перехода состояния, $\lambda : S \times I \to O$ – выходная функция [3].

Установочная последовательность:

- начальные состояния неизвестны;
- применяется входная последовательность $x \in I$;
- возможно только одно конечное состояние $|\delta(S, x)| = 1$.

Идея вычисления установочной последовательности:

- итеративная конкатенация строк;
- на каждом шаге уменьшается $|\delta(S, x)|$;
- каждая строка должна объединяться для двух состояний в $|\delta(S, x)|$:
  a. слияние последовательности $y \in I^*$ для двух состояний $s, t \in S$ приводит их к одному и тому же конечному состоянию: $|\delta(s, y)| = |\delta(t, y)|$;
  b. $|\delta(S, xy)| < |\delta(S, x)|$;
  c. слияние последовательности существует для всех состояний тогда и только тогда, когда существует установочная последовательность.

Алгоритм вычисления установочной последовательности:

1. на вход подается автомат Мили, при этом $x \leftarrow \varepsilon$, где $x \in I^*$, ε – входная строка;
2. *while* $|\delta(S, x)| > 1$:
   a. возьмем два разных состояния $s, t \in \delta(S, x)$;
   b. пусть y является последовательностью слияния s, t. Если такая последовательность не существует, *return failure*;
   c. $x \leftarrow xy$;
3. *return x*.

Оценка сложности алгоритма:

- время: $O(n^3 + n^2 + |I|)$, где n – число состояний, $|I|$ – число входных символов;
- пространство: $O(n^2 + n \cdot |I|)$ (не считая пространства, необходимого для выхода);
- длина последовательности: $< (n^3 - n)/6$.

2.2 Метод регулярных безусловных установочных экспериментов

Установочный эксперимент, построенный с помощью метода, который не гарантирует минимальной длины, называется регулярным установочным экспериментом [4].

Заданы автомат M с n состояниями и его множество допустимых начальных состояний *A(M)*. Пусть существует ветвь, связанная с *A*-группой *G*, состоящей из σ-множеств $g_1, g_2, \ldots, g_u$, в которой, по крайней мере, одно σ-множество, скажем g_h , не является однородным. Если g_h содержит r состояний, то оно должно содержать, по крайней мере, два состояния, скажем σ_i, σ_j , которые являются $(n - r + 1)$-различными. Следовательно, подпуть, начинающийся с существующей ветви и описывающийся последовательностью $\varepsilon(\sigma_i, \sigma_j)$, должен вести к *A*-группе G', которая состоит, по меньшей мере, из $(u+1)$-го σ-множества. Следовательно, если *G* является неоднородной, то всегда может быть найдет подпуть длины $(n - r + 1)$ или меньшей, который ведет от *G* к *A*-группе, решение которой превышает решение *G*. Описываемая таким подпутем подпоследовательность называется регулярной подпоследовательностью *G*.

Необходимо найти установочную последовательность для M и *A(M)*, которая является любой последовательностью, которая, будучи приложенной к $M|\sigma_i$ и $M|\sigma_j$, дает две различные выходные последовательность, если она переводит σ_i, σ_j в два различных состояния.

Лемма 1. Входная последовательность, описываемая установочным путем в установочном дереве, построенном для M и *A(M)*, есть установочная последовательность для M и *A(M)*.

Алгоритм:

1) пусть *A(M)* является G_0. Полагаем $k = 0$;

2)

a. если G_k не является однородным, то определим регулярную подпоследовательность, скажем ε_k, для G_k. Пусть ε_k – преемником G_k является G_{k+1}. $k = k + 1$ и возвращаемся к 2);

b. Если G_k однородно, то $\varepsilon_0, \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{k-1}$ есть установочная последовательность для *M* и *M(A)*.

2.3 Метод построения установочной последовательности для конечных автоматов с помощью прямого различающего дерева

Множество σ – это множество, содержащее все состояния исправного автомата *A* и неисправного автомата $A_н$.

Назовем *A*-группу гомогенной, если в ней состояния исправного и неисправного автоматов *A* и $A_н$ находятся отдельно в разных σ-множествах.

Прямое различающее дерево (ПРД) – это дерево приемников состояний, в котором вершина α *k*-го уровня становится оконечной, если выполняется одно из условий:

1. для *A* – группы A_α, которая связана с вершиной α, существует *A*-группа A_β, которая связана с вершиной β-го уровня, предшествующего *k*-му, такая, что для каждого σ – множества из A_α существует включающее его σ – множества из A_β;

2. существует вершина *k*-го уровня, связанная с гомогенной *A*-группой.

Пример фрагмента ПРД (установочная последовательность – «1111») представлен на рисунке 2.

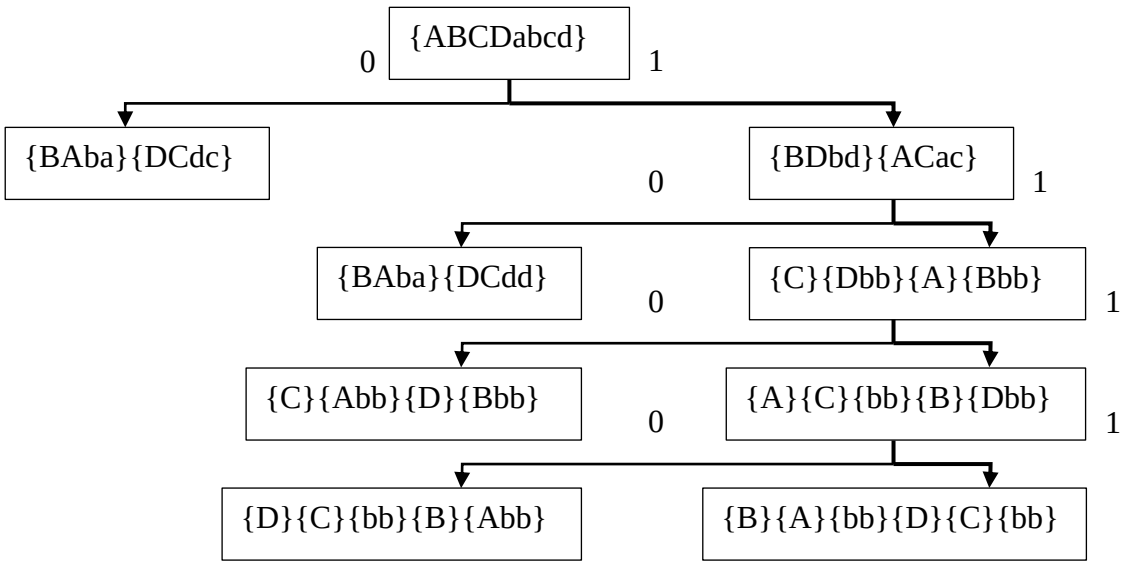


Рисунок 2 – Фрагмент прямого различающего дерева

2.4 Задача выполнимости

2.4.1 Задача выполнимости булевых формул

Задача выполнимости булевой формулы, является одной из важнейших задач в области информационных технологий, потому что многие практические задачи сводятся к задаче о выполнимости (SAT).

Теорема 1 (теорема Кука, 1971). Задача выполнимости булевой формулы – есть NP-полная задача.

NP-полные задачи могут быть решены алгоритмами за экспоненциальное время. Не существует известных алгоритмов, решающие NP-полные задачи за полиномиальное время, и никому еще не удалось доказать, что таких алгоритмов нет.

Задача выполнимости решается для дизъюнктивной нормальной формы (ДНФ) за полиномиальное время, в то время, как задача определения тавтологии близка к NP-полной. Для КНФ все наоборот. Таким образом, выполнимость легче определить для ДНФ, а тавтологию для КНФ, но это не имеет значения, так как переход между КНФ и ДНФ является экспоненциальным.

Частные случаи задачи SAT:

- задача выполнимости булевых формул в КНФ (SATCNF) – аналогичная задача, с наложенным на формулу условием: она должна быть записана в конъюнктивной нормальной форме. Задача SATCNF также NP-полна;

- задача выполнимости булевых формул в *k*-КНФ (*k*-SAT) – задача выполнимости при условии, что формула записана в *k*-КНФ. Задача является NP-полной при $k \geq 3$;
- задача выполнимости булевых формул в *2*-КНФ имеет полиномиальное решение, то есть принадлежит классу P.

2.4.2 Метод поиска кратчайшей установочной последовательности посредством сведения к выполнимости КНФ

Сведение задачи поиска установочной последовательности к задаче проверки выполнимости КНФ разрешения комбинационного блока, реализующего функции возбуждения триггеров [5].

Множество равенств типа $x_i = \sigma_i$, где $\sigma_i \in \{0,1\}$, $i = \overline{1,n}$ задает присваивание значений компонентам вектора $x = (x_1, x_2, \ldots, x_n)$. Присваивание значений компонентам вектора *x* может быть полным, если задаются значения всех x_i , иначе частичным. Необходимо найти такое присваивание (может быть частичным) значений переменным КНФ, которое обращает КНФ в единицу. Если такое присваивание существует, то КНФ выполнима.

Комбинационная схема представляется в виде КНФ разрешения схемы, задающая все возможные комбинации сигналов на всех её $n+l$ полюсах, где n – число входных полюсов схемы, l – число элементов. При этом КНФ разрешения всех элементов объединяется операцией конъюнкции в одну КНФ разрешения схемы.

Примеры КНФ разрешения простых элементов:

- $y = \bar{z} \rightarrow (z \vee y) \wedge (\bar{z} \vee \bar{y})$;
- $y = z_1 \wedge z_2 \wedge \ldots \wedge z_n \rightarrow (z_1 \vee \bar{y}) \wedge (z_2 \vee \bar{y}) \wedge \ldots \wedge (z_n \vee \bar{y}) \wedge (\bar{z}_1 \vee \bar{z}_2 \vee \ldots \vee \bar{z}_n \vee y)$;
- $y = z_1 \vee z_2 \vee \ldots \vee z_n \rightarrow (\bar{z}_1 \vee y) \wedge (\bar{z}_2 \vee y) \wedge \ldots \wedge (\bar{z}_n \vee y) \wedge (z_1 \vee z_2 \vee \ldots \vee z_n \vee \bar{y})$;
- $y = \overline{z_1 \wedge z_2 \wedge \ldots \wedge z_n} \rightarrow (z_1 \vee y) \wedge (z_2 \vee y) \wedge \ldots \wedge (z_n \vee y) \wedge (\bar{z}_1 \vee \bar{z}_2 \vee \ldots \vee \bar{z}_n \vee \bar{y})$;
- и другие.

Метод сокращения вычислительных затрат при поиске кратчайшей установочной последовательности путем сведения к задаче проверки выполнимости КНФ. Этот метод описывается блок-схемой, которая представлена на рисунке 3.

Процесс поиска кратчайшей установочной последовательности начинается с поиска установочной последовательности входных воздействий, которые приводят к установке триггеров в известное состояние за один такт работы схемы. При построении КНФ разрешения C^i для комбинационной схемы аргументы $(x_1^i, x_2^i, \ldots, x_n^i)$ соответствуют входным полюсам последовательностной схемы, а аргументы $(z_1^i, z_2^i, \ldots, z_k^i)$ соответствуют k линиям обратной связи последовательностной схемы. Изначально, при $i = 1$ входные переменные $(z_1^i, z_2^i, \ldots, z_k^i)$ не определены, то есть $z_1^i = z_2^i = \ldots = z_k^i =$"–" . Если КНФ разрешения $C^i(i = 1)$ выполнима, тогда будет найден набор значений переменных $X^i = (x_1^i, x_2^i, \ldots, x_n^i)$, который представляет кратчайшую установочную последовательность длины i. Так же результатом подачи на входы значений из X^i найден набор значений переменных $Y^i = (y_1^i, y_2^i, \ldots, y_n^i)$, который задает значения выходных функций комбинационного блока и определяет состояние последовательностной схемы в следующий такт её работы.

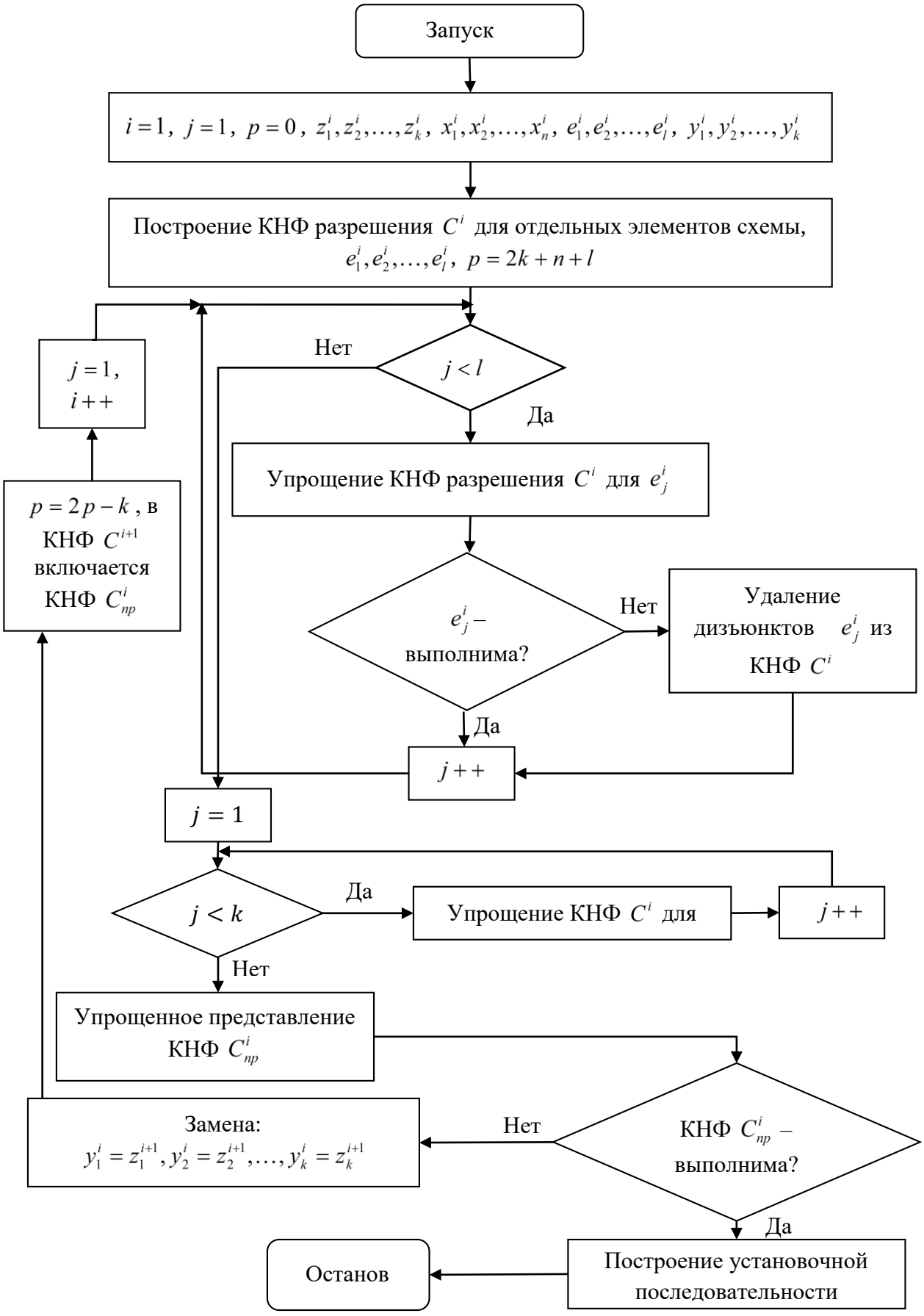


Рисунок 3 – Блок-схема задачи проверки выполнимости КНФ

Результаты, которые соответствуют выходным значениям отдельных элементов комбинационной схемы, обозначаются в виде элементов $e_1^i, e_2^i, \ldots, e_l^i$.

Если КНФ разрешения C^i невыполнима, то длина искомой установочной последовательности увеличивается на единицу и формируется КНФ разрешения C^{i+1} для поиска установочной последовательности X^{i+1}, Y^{i+1}, при этом C^{i+1} формируется для двухблочной комбинационной схемы. Структура блоков соответствует исходной последовательностной схеме. При этом переменные $(z_1^{i+1}, z_2^{i+1}, \ldots, z_k^{i+1})$, которые соответствуют линиям обратной связи, их значения совпадают на следующем такте со значениями входных переменных блока i: $y_1^i = z_1^{i+1}, y_2^i = z_2^{i+1}, \ldots, y_k^i = z_k^{i+1}$. Процесс увеличения числа блоков комбинационной схемы продолжается до тех пор, пока на некотором шаге не будет найдена установочная последовательность, которая и будет кратчайшей, или число итераций не достигнет наперед заданного предельного числа.

Пример увеличения числа блоков комбинационной логической схемы представлен на рисунке 4.

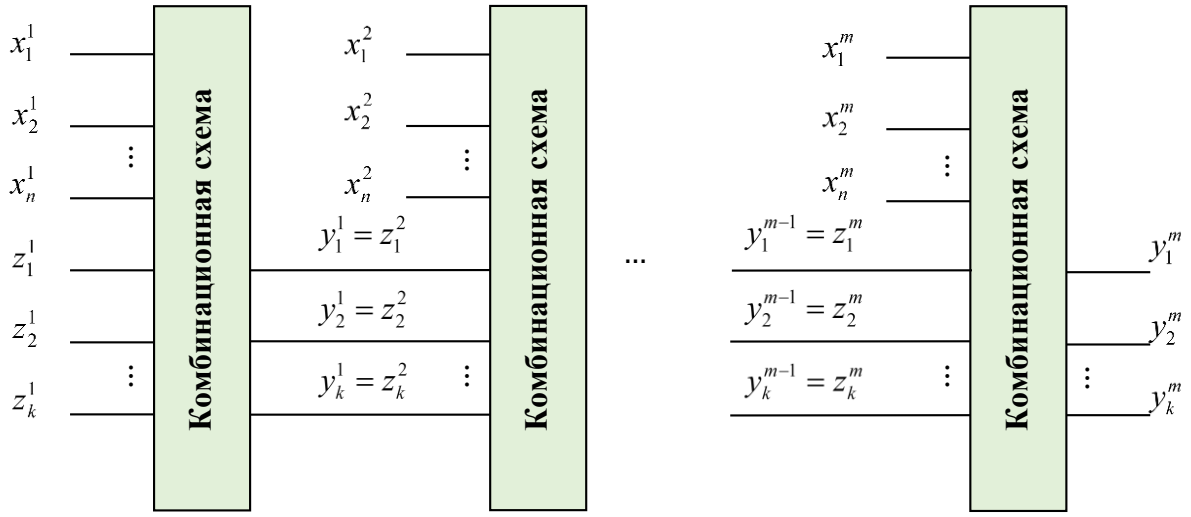


Рисунок 4 – Пример увеличение числа блоков комбинационной логической схемы

2.4.3 Оригинальный метод поиска кратчайшей установочной последовательности

В работе предлагается метод ориентированный на использование специфики КНФ, строящейся для нахождения последовательности, переводящей автомат из начального внутреннего состояния в одно из внутренних состояний множества. Множество представлено интервалом (конъюнкцией K) в пространстве внутренних переменных.

Комбинационная составляющая строится из двух выходных и одно выходных вентилей. Большинство из порождаемых ими дизъюнктов содержит по две литеры.

Операции метода:

1. сначала фиксируются константами внутренние переменные первого блока в соответствие с заданным начальным состоянием синхронного автомата;

2. далее в дизъюнктах последнего блока вычеркиваем переменные, инверсные присутствующим в конъюнкции K.

Операции 1, 2 приводят к образованию однолитерных дизъюнктов. Возникает возможность применения следующих правил упрощения:

- пусть A - одно литерный дизъюнкт, тогда $A \wedge (A \vee X) = A$;
- $A \wedge (\overline{A} \vee X) = A \wedge X$;
- $A \wedge A = A$.

Проиллюстрируем использование предложенного нами метода упрощения КНФ с последующим построением установочной последовательности на схеме (рисунок 5). Пусть начальное состояние представляется булевым вектором «111» в пространстве внутренних переменных. Конъюнкция $K = z_1 z_2 z_3$ в том же пространстве переменных.

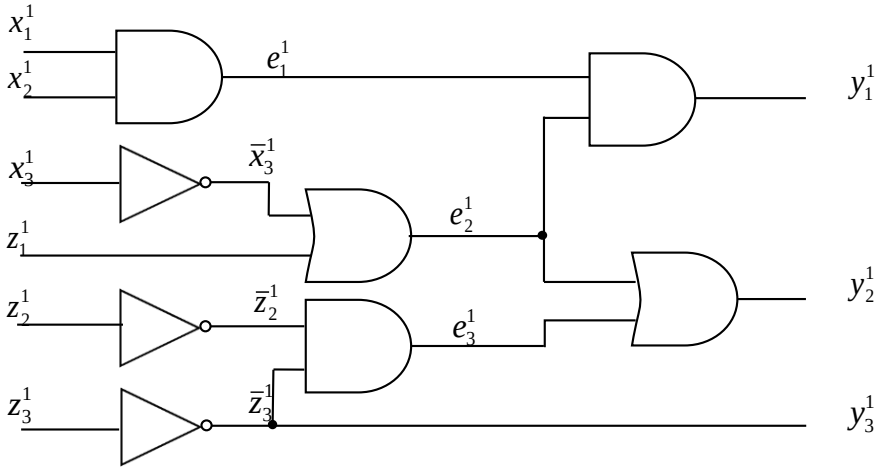


Рисунок 5 – Пример комбинационной составляющей

КНФ разрешения C^1 для первого блока принимает следующий вид:

$(x_1^1 \vee \overline{e}_1^1) \wedge (x_2^1 \vee \overline{e}_1^1) \wedge (\overline{x}_1^1 \vee \overline{x}_2^1 \vee e_1^1) \wedge (x_3^1 \vee e_2^1) \wedge (\overline{z}_1^1 \vee e_2^1) \wedge (\overline{x}_3^1 \vee z_1^1 \vee \overline{e}_2^1) \wedge (\overline{z}_2^1 \vee \overline{e}_3^1) \wedge$
$(\overline{z}_3^1 \vee \overline{e}_3^1) \wedge (z_2^1 \vee z_3^1 \vee e_3^1) \wedge (e_1^1 \vee \overline{y}_1^1) \wedge (e_2^1 \vee \overline{y}_1^1) \wedge (\overline{e}_1^1 \vee \overline{e}_2^1 \vee y_1^1) \wedge (\overline{e}_2^1 \vee y_2^1) \wedge (\overline{e}_3^1 \vee y_2^1) \wedge$
$(e_2^1 \vee e_3^1 \vee \overline{y}_2^1) \wedge \overline{z}_3^1.$

КНФ разрешения для первого блока после применения правил упрощения имеет следующий вид: $C^1 = x_1^1 \wedge x_2^1 \wedge e_1^1 \wedge e_2^1 \wedge \overline{e}_3^1 \wedge y_1^1 \wedge y_2^1 \wedge 0.$

Полученная КНФ разрешения C^1 невыполнима, то есть установочной последовательности длины 1 для схемы, представленной на рисунке 5, не существует. Переходим к построению КНФ разрешения C^2 для следующего блока, пусть этот блок является последним. Пример комбинационной составляющей изображен на рисунке 6.

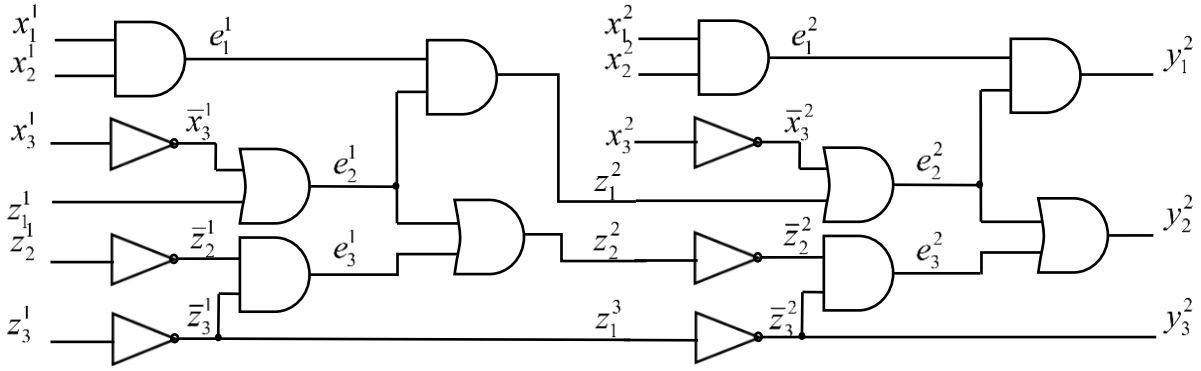


Рисунок 6 – Пример комбинационно составляющей

Заметим, что в схеме, являющейся комбинационной составляющей схемы с памятью, выходы y_1^1, y_2^1, y_3^1 отождествляются с входами z_1^2, z_2^2, z_3^2, то есть $z_1^2 = y_1^1$, $z_2^2 = y_2^1$, $z_3^2 = y_3^1$. КНФ разрешения C^2 принимает следующий вид:

$$C^2 = x_1^1 \wedge x_2^1 \wedge e_1^1 \wedge e_2^1 \wedge \bar{e}_3^1 \wedge z_1^2 \wedge z_2^2 \wedge (x_1^2 \vee \bar{e}_1^2) \wedge (x_2^2 \vee \bar{e}_1^2) \wedge (\bar{x}_1^2 \vee \bar{x}_2^2 \vee e_1^2) \wedge (x_3^2 \vee e_2^2) \wedge$$
$$(\bar{z}_1^2 \vee e_2^2) \wedge (\bar{x}_3^2 \vee z_1^2 \vee \bar{e}_2^2) \wedge (\bar{z}_2^2 \vee \bar{e}_3^2) \wedge (\bar{z}_3^2 \vee \bar{e}_3^2) \wedge (z_2^2 \vee z_3^2 \vee e_3^2) \wedge (e_1^2 \vee \bar{y}_1^2) \wedge (e_2^2 \vee \bar{y}_1^2) \wedge$$
$$(\bar{e}_1^2 \vee \bar{e}_2^2 \vee y_1^2) \wedge (\bar{e}_2^2 \vee y_2^2) \wedge (\bar{e}_3^2 \vee y_2^2) \wedge (e_2^2 \vee e_3^2 \vee \bar{y}_2^2) \wedge \bar{z}_3^2.$$

КНФ разрешения C^2 после упрощения:

$$C^2 = x_1^1 \wedge x_2^1 \wedge e_1^1 \wedge e_2^1 \wedge \bar{e}_3^1 \wedge z_1^2 \wedge z_2^2 \wedge \bar{z}_3^2 \wedge x_1^2 \wedge x_2^2 \wedge e_1^2 \wedge e_2^2 \wedge \bar{e}_3^2 \wedge y_1^2 \wedge y_2^2.$$

КНФ разрешения C^2 оказывается выполнимой, один из выполняющих наборов значений её переменных имеет следующий вид:

z_1^1	z_2^1	z_3^1	x_1^1	x_2^1	x_3^1	e_1^1	e_2^1	e_3^1	z_1^2	z_2^2	z_3^2	x_1^2	x_2^2	x_3^2	e_1^2	e_2^2	e_3^2	y_1^2	y_2^2	y_3^2
1	1	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0

Он найден с помощью SAT решателя Riss7 и представляет установочную последовательность длины 2: $X^1(110), X^2(110), Y^2(110)$.

2.4.4 Результаты SAT Competition 2017

Ежегодное международное SAT соревнование является конкурентным событием для решателей SAT проблемы [6]. Соревнование организовано в качестве мероприятия для International Conference on Theory and Applications of Satisfiability Testing и является традицией SAT соревнований. Соревнование состоит из следующих треков: main track, parallel track, incremental track, agile track, random SAT track, no-limits track.

Main Track – это трек последовательных SAT решателей. Мы будем использовать 300-600 контрольных проблем, время выполнения составляет 5000 секунд. Результаты представлены в таблице 1.

Таблица 1 – Результаты Main Track

Топ-3	Оценка	Суммарно решено	SAT	UNSAT	Наименование решателя
1	1610934.19	208	102	106	Maple_LCM_Dist
	1640696.51	206	100	106	Maple_LCM
	1654244.83	204	98	108	MapleLRB_LCMoccRestart
	1676517.64	198	94	104	MapleLRB_LCM
2	1780711.47	188	93	95	MapleCOMSPS_LRB_VSIDS_2_drup
	1805445.41	185	96	89	MapleCOMSPS_LRB_VSIDS_drup
3	1798300.37	188	89	99	COMiniSatPS_Pulsar_drup

Random SAT Track – SAT решатели для случайных генерируемых экземпляров. Результаты представлены в таблице 2.

Таблица 2 – Результаты Random SAT Track

Топ-3	Оценка	Суммарно решено	SAT	UNSAT	Наименование решателя
1	1794277.02	124	124	0	yalsat
2	1894840.68	113	113	0	tch_glucose3
3	2075884.23	97	97	0	Score2SAT

Agile Track – это трек для простых быстрых SAT решателей с низкими расходами, которые подходят для решения большого количества не слишком сложных SAT экземпляров. Результаты представлены в таблице 3.

Таблица 3 – Результаты Agile Track

Топ-3	Оценка	Суммарно решено	SAT	UNSAT	Наименование решателя
1	240983.14	3287	937	2350	cadical-agile
	248613.04	3129	704	2425	cadical-noproof
2	257997.93	3162	825	2337	glu_vc

3	263772.21	3100	770	2330	glucose-4.1

Parallel Track – трек для параллельных SAT решателей, предназначенный для компьютеров с несколькими процессорами или ядрами. Результаты представлены в таблице 4.

Таблица 4 – Результаты Parallel Track

Топ-3	Оценка	Суммарно решено	SAT	UNSAT	Наименование решателя
1	1229297.26	237	113	124	syrup24
	1334230.98	227	112	115	Syrup48
2	1266163.50	239	110	129	plingeling
3	1368420.42	223	114	109	Painless MapleCOMSPS

Incremental Track – трек для инкрементных SAT-решателей. Данный трек основан на интерфейсе Re-entrant Incremental Satisfiability Application Program Interface (IPARIS). Пользователи, которые хотят использовать инкрементные SAT-решатели в своих приложениях, выигрывают от такого интерфейса. Интерфейс был разработан с учетом следующих свойств:

- простота в реализации;
- простота использовании, так что каждый может легко создавать приложения на основе SAT;
- универсальность и мощность, так что он может использоваться для широкого спектра промышленных и исследовательских приложений.

Результаты трека представлены в таблице 5.

Таблица 5 – Результаты Incremental Track

Benchmark App	AbcdSAT	Glucose	Riss
Essential Variables	2	1	3
Longest Path Search	1	2	3
Partial MaxSAT	1	2	3
Pigeon Hole Principle	3	1	1
Automated Planning	1	2	3
ijtihad (QBF solver)	2	3	1
CBMC (C verification)	3	2	1
SATPin (axiom pinpointing)	2	3	1
average rank	1.875	2.000	2.000

Было принято решение использовать SAT-решатели из Incremental Track благодаря своей универсальности.

3 Алгоритмы упрощения КНФ

Проблема состоит в том, что исходные КНФ комбинационных схем имеют большие размерности для практических задач. Желательно упростить исходную КНФ, если это возможно. В дальнейшем будем рассматривать комбинационные эквиваленты синхронной схемы фиксированной длины с представлением их в виде КНФ.

КНФ комбинационного эквивалента является формулой, сложность которой растет с увеличением длины комбинационного эквивалента. Разным блокам сопоставляются аналогичные последовательности дизъюнктов, отличающиеся друг от друга символами переменных.

Известные методы упрощения КНФ делятся на методы, ориентированные на произвольные КНФ, и методы, учитывающие специфику КНФ.

3.1 Методы, ориентированные на произвольные КНФ

3.1.1 Метод резолюции

Проблема состоит в том, что исходные КНФ имеют большие размерности даже при небольшой размерности исходной задачи. Преобразование исходной КНФ методом резолюции позволяет получить КНФ с меньшим количеством дизъюнктов и литералов, эквивалентную исходной [7].

«Резольвента» – дизъюнкция конъюнктов, отличающихся знаком по единственной переменной. Все возможные резольвенты добавляются к КНФ и используются для вычисления других резольвент.

Дублирующие конъюнкты и тавтологии удаляются, и используется сокращенная процедура с глубиной рекурсии 1. Вычислительная сложность процедуры $O(n \cdot \log n)$. Метод резолюции в применении к КНФ позволяет уменьшить исходное число конъюнктов до 50%.

Метод резолюции основан на правилах исчисления высказываний и включает в себя ряд методик:

1) Простая резолюция и разрешение уникальных переменных. Этот метод основан на следующих правилах вывода:

   a. $x \wedge (x \vee Y) \equiv x$;

b. $TRUE \vee x \equiv TRUE$;

c. $FALSE \vee x \equiv x$;

d. $x \vee x \equiv TRUE$;

e. $x \wedge x \equiv o$ (пустой дизъюнкт);

2) Резолюция по соседним дизъюнктам. Два дизъюнкта являются соседними, если они различаются знаком по единственному литералу. Резольвентой по соседним дизъюнктам называется дизъюнкт, являющийся максимальным общим подмножеством соседних дизъюнктов. Если в КНФ присутствует конъюнкция соседних дизъюнктов, то конъюнкция заменяется резольвентой по соседним дизъюнктам;

3) Бинарная резолюция. Два дизъюнкта разрешимы относительно бинарной резолюции если они совпадают хотя бы по одной переменной, которая входит в один дизъюнкт с отрицанием, а в другой – без.

3.1.2 Алгоритм минимизации функции component-wise quadratic (CQ)

На вход алгоритма подается КНФ C, представляющая функцию h из класса CQ.

На выходе КНФ $C_{\min}$, представляющая h с минимальным числом конъюнкций и с минимальным числом литералов.

Алгоритм [8]:

1) построение импликационного графа G_C , пример графа представлен на рисунке 7. Построение транзитивного замыкания G_h;

2) построение графа $D_C = (V_C, E_C)$, где $V_C = V(D_C) = C$ – множество вершин, $E_C = E(D_C)$ – множество ребер. Для C_1, $C_2 \in C$ существует $(C_1, C_2) \in E_C$ если и только если выполняются два условия:

- $Head(C_1) \in Cone_{G_C}(Head(C_2))$, где $C = \bigvee_{x \in S} \bar{x} \vee y = S \vee y$ и каждая пропозициональная переменная $x \in S$ называется подцелью C и пропозициональная переменная y называется $Head(C) = y$, $Sub(g(C)) = S$. $Cone_D(x)$ – наименьшее (по включению) начальное множество D содержащее x;
- $Sub(g(C_1)) \subseteq FC_C(Sub(g(C_2)))$. Пусть подмножество пропозициональных переменных – Q , а множество простых *Horn* конъюнкций H . Обозначим

множество $M = Q$. До тех пор, пока $\exists \quad C \in \mathrm{H} : Sub(g(C)) \subseteq M$ и $Head(C) \notin M$ вычисляем $M = M \cup \{Head(C)\}$. Результатом является $FC_{\mathrm{H}}(Q) = M$;

Поиск сильных компонентов графа D_C и поиск некоторого топологического порядка этих компонент: $T = (K_1, \ldots, K_t)$. Если граф не сильно связан, то его множество вершин можно разложить единственным образом в максимально сильно связанные подмножества, которые называются сильными компонентами.

3) основной цикл: обработать сильные компоненты графа D_C в топологическом порядке *T*, который был найден на предыдущем шаге, и для каждой компоненты *K* выполнить следующие действия:

- Действие $\mathcal{K}_0$: если компонента *K* является $\mathcal{K}_0$ типа, тогда ничего не делать;
- Действие $\mathcal{K}_1$: если компонента K является $\mathcal{K}_1$ типа, тогда выполняем следующие пункты:
  - (a) переключить все множества подцелей $Q(K)$ всех конъюнкций в *K* некоторого множества $A(K)$, получив множество конъюнкций K' и новое представление C';
  - (b) построение подграфа $Q'(K')$ из $Q(K')$ чьи дуги генерируются только по конъюнкциям из $Cone_{D_{C'}}(K')$ и отметить все дуги генерируемые конъюнкциями из K' как свободные;
  - (c) найти уменьшение $Q'(K')$ с добавлением ограничения, это уменьшение может содержать все фиксированные дуги. Свободные дуги могут быть удалены и заменены другими дугами;
  - (d) Для каждой удаленной дуги, удалим из C' соответствующую конъюнкцию. Для каждой добавленной дуги, добавить C' соответствующую квадратичную конъюнкцию и удлинить конъюнкцию, добавив множество подцелей $A(K)$. Обновить граф $D_{C'}$, соответственно;

4) Выход: КНФ $C_{\min}$.

Теорема 2 (о временной сложности алгоритма). Пусть *h* – функция *Horn n* переменных из класса CQ. Пусть *C* – простая КНФ, представляющая *h*, подающаяся на вход

алгоритма. Тогда алгоритм выполняется за $O(n^2 + ml)$ время, где m – это число конъюнкций, а l – число литералов в C.

3.1.3 Horn минимизация путем итеративного разложения

Пусть C – Horn КНФ, G_C – импликационный граф, C_i – сильный компонент.

Построение графа $G_C = (N, A_C)$, где $N = \{x \mid x$ пропозициональная переменная в $C\}$, $A_C = \{(x, y) \mid \exists$ конъюнкция $T \in \hat{C} : T \geq \bar{x} \vee y\}$.

Пусть $C_i = P(C_i) \cap S(C_i)$, где:

- $P(C_i) = \{x \mid x \in C_j$ для некоторых $C_i \preccurlyeq C_j\} = \bigcup_{j: C_j \preccurlyeq C_i} C_j$;
- $S(C_i) = \{x \mid x \in C_j$ для некоторых $C_i \preccurlyeq C_j\} = \bigcup_{j: C_i \preccurlyeq C_j} C_j$.

Алгоритм декомпозиции [9]:

1. **For** каждая конъюнкция $T = A \vee x \in \hat{C}$ **do** (предположить, что $x \in C_i$):

   1.1. **If** $|A \cap C_i| \geq 2$ **then** пометить все компоненты в $P(C_i)$ как заблокированные;

   1.2. **If** $|A \cap C_i| = 1$ (говорят $A \cap C_i = \{y\}$) **then for** каждая $z \in A\{y\}$ **do** пометить все компоненты в $P(C_j)$ как заблокированные (при условии $z \in C_j$);

2. **For** каждая конъюнкция $T = A \vee x \in \hat{C}$ **do** (предположить, что $x \in C_i$):

   2.1. **If** C_i заблокирована **then** положить T в очередь B;

   2.2. **If** C_i переключаемая **then**

      2.2.1. **If** $|A \cap C_i| = 1$ **then** положить T в очередь $N_1(i)$;

      2.2.2. **If** $|A \cap C_i| = 0$ **then** положить T в очередь $N_0(i)$;

3. **For** $i = \overline{s,1}$ **do if** C_i переключаемая **then**

   3.1. **For** каждая конъюнкция $T = A \vee x$ в очереди $N_1(i)$ **do** переключить x и пропозициональная переменная в $A \cap C_i$;

   3.2. **For** каждая конъюнкция $T = A \vee x$ в очереди $N_0(i)$ **do** переключить x и устранить T из C;

4. Выход: все исключенные отрицательные конъюнкции из всех очередей N_0 и все «не подключенные» простые Horn КНФ, состоящие из конъюнкций в очереди *B* и всех очередей N_1.

Пример 1. Horn КНФ представлена формулой (1):

$$C = (\bar{x}_1 \vee \bar{x}_5 \vee x_6) \wedge (\bar{x}_1 \vee \bar{x}_6 \vee x_7) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee x_4) \wedge (\bar{x}_4 \vee \bar{x}_7 \vee x_8) \wedge (\bar{x}_7 \vee x_6) \quad (1)$$

На рисунке 7 изображен импликационный граф G_C для заданной КНФ *C*.

В данном примере получено пять сильных компонент, два последних при этом являются переключаемыми. Таким образом переменные в последней сильной компоненте $C_5 = \{x_8\}$ могут быть переключены первыми, то есть заменяем x_8 на $\bar{x}_8$. Получаем КНФ $C' = (\bar{x}_1 \vee \bar{x}_5 \vee x_6) \wedge (\bar{x}_1 \vee \bar{x}_6 \vee x_7) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee x_4) \wedge (\bar{x}_4 \vee \bar{x}_7 \vee \bar{x}_8) \wedge (\bar{x}_7 \vee x_6)$, в которой содержатся негативная конъюнкция. Импликационный граф $G_{C'}$ для КНФ C' изображен на рисунке 8. Таким образом получим сокращенную КНФ $C'_{сокр}$, которая представлена в формуле (2).

$$C'_{сокр} = (\bar{x}_1 \vee \bar{x}_5 \vee x_6) \wedge (\bar{x}_1 \vee \bar{x}_6 \vee x_7) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee x_4) \wedge (\bar{x}_7 \vee x_6) \quad (2)$$

Далее можно переключить переменные в компоненте $C_4 = \{x_6, x_7\}$. Будет получена формула (3) КНФ C''.

$$C'' = (\bar{x}_1 \vee \bar{x}_5 \vee x_6) \wedge (\bar{x}_1 \vee x_6 \vee \bar{x}_7) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee x_4) \wedge (x_7 \vee \bar{x}_6) \quad (3)$$

Аналогично можно получить сокращенную КНФ $C''_{сокр}$, которая представлена формулой (4):

$$C''_{сокр} = (\bar{x}_1 \vee \bar{x}_6 \vee x_7) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee x_4) \wedge (x_7 \vee \bar{x}_6) \quad (4)$$

Граф $G_{C''}$ соответствующий КНФ C'' изображен на рисунке 9.

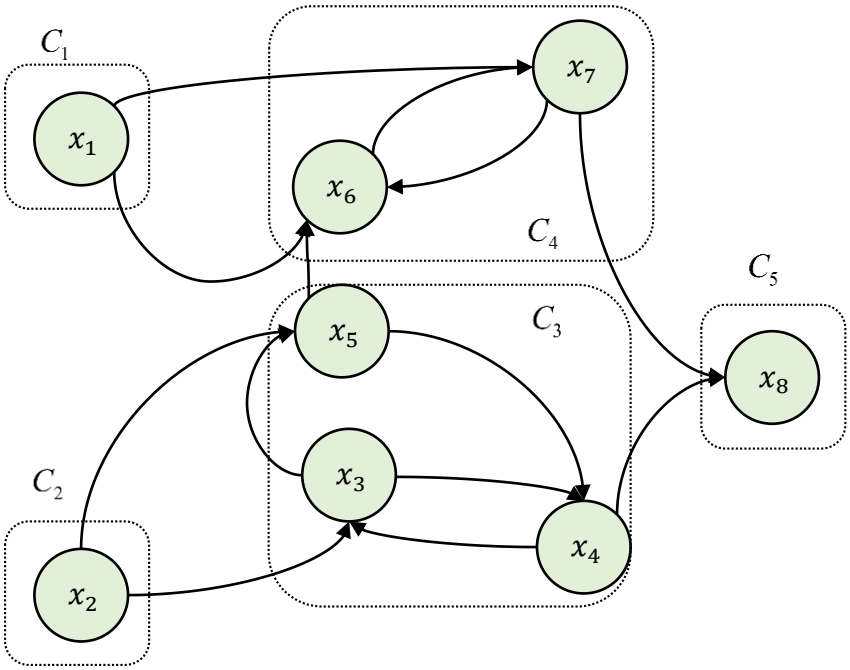


Рисунок 7 – Пример импликационного графа G_C

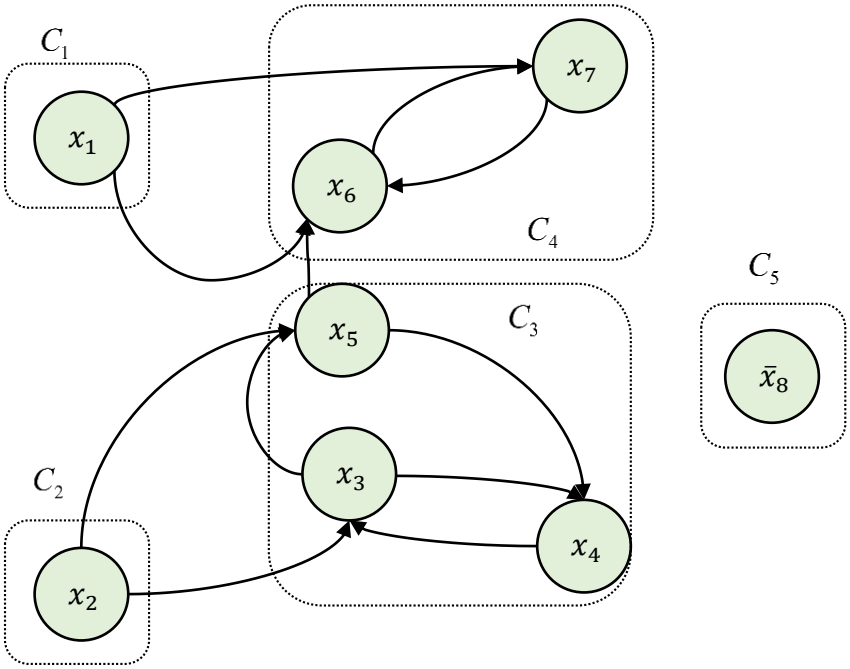


Рисунок 8 – Пример импликационного графа G_C

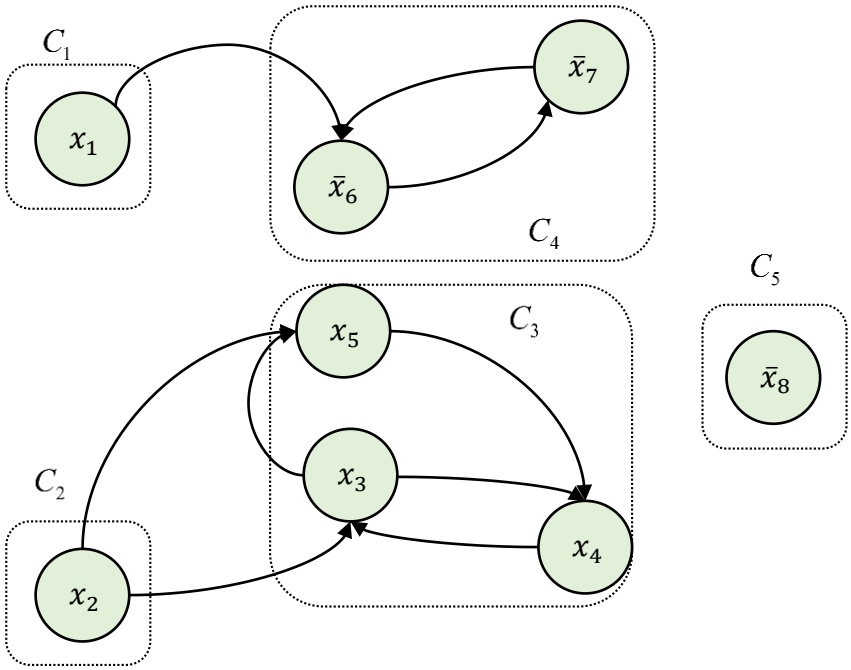


Рисунок 9 – Пример импликационного графа $G_{C''}$

Так как переключаемых компонент больше нет, алгоритм декомпозиции заканчивает свою работу. Результирующей минимальной КНФ является КНФ $C''_{сокр}$.

3.2 Методы, ориентированные на специфику КНФ

3.2.1 Идея метода упрощения КНФ разрешения

Метод поиска кратчайшей установочной последовательности посредством сведения к выполнимости КНФ [3] основан на последовательном увеличении числа однотипных блоков комбинационной схемы, до тех пор, пока на некотором шаге не будет получена выполнимая КНФ разрешения, по которой определяется кратчайшая установочная последовательность, или не будет исчерпан лимит времени.

Идея метода упрощения КНФ разрешения в данной работе заключается в следующем:

а) в первом блоке замена определенных значений переменных $z_1^1, z_2^1, \ldots, z_k^1$ на неопределенные;

b) проверка на выполнимость КНФ разрешения отдельных элементов комбинационной схемы. Если данная КНФ разрешения невыполнима, тогда исключаются все вхождения этого элемента в дизъюнктах КНФ разрешения комбинационной схемы;

c) применение элементарных правил упрощения.

3.2.2 Упрощение КНФ разрешения оригинального метода

Операции 1, 2 приводят к образованию однолитерных дизъюнктов. В следствии чего возникает возможность применения следующих правил упрощения КНФ резолюции:

- пусть A - одно литерный дизъюнкт, тогда $A \wedge (A \vee X) = A$;
- $A \wedge (\overline{A} \vee X) = A \wedge X$;
- $A \wedge A = A$.

3.3 Выводы

При анализе методов упрощения КНФ было выявлено, что известные методы делятся на два типа:

1. методы, ориентированные на произвольные КНФ;
2. методы, учитывающие специфику КНФ.

Так же оценка сложности является хорошим способом прогнозирования времени работы алгоритма при увеличении объема данных. Никакие тесты производительности не дадут такой информации, т.к. они зависят от особенностей конкретного компьютера и обработки конкретных данных. В ходе изучения представленных выше алгоритмов упрощения КНФ, для некоторых из них была приведена временная сложность.

Временная сложность методов упрощения КНФ, ориентированных на произвольные КНФ:

1. метод резолюции: $O(n \cdot \log(n))$;
2. алгоритм минимизации функции CQ: $O(n^2 + ml)$, где m – число конъюнкций, а l – число литералов в простой конъюнкции;
3. Horn минимизация путем итеративного разложения: $O(n)$.

Примерный график временной сложности данных методов изображен на рисунке 10.

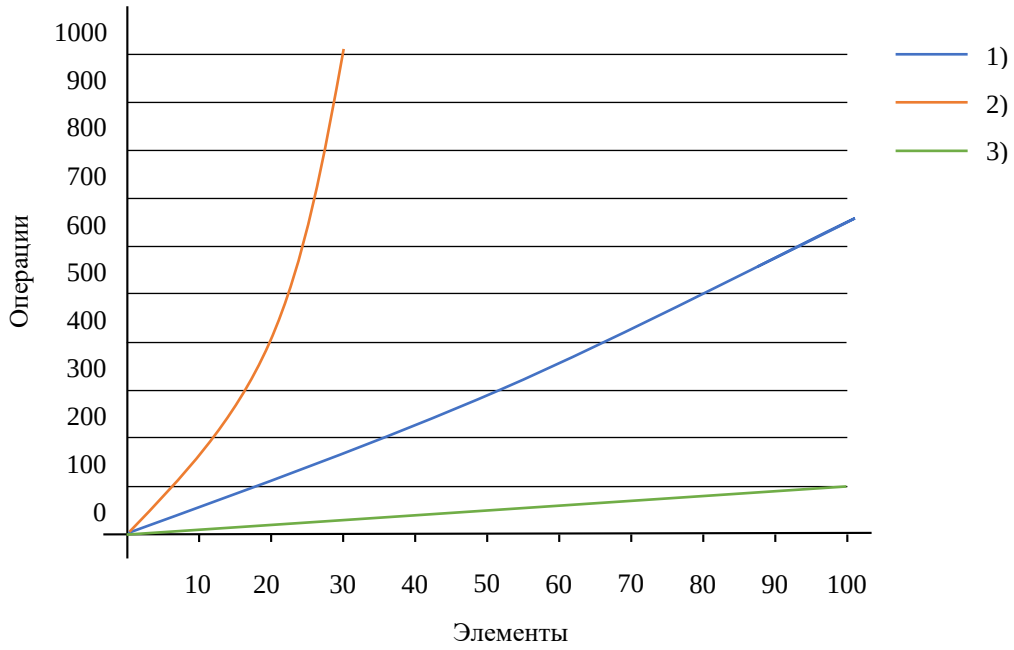


Рисунок 10 – Примерный график временной сложности методов упрощения КНФ, ориентированных на произвольные КНФ

При анализе временных сложностей алгоритмов было установлено, что метод 2) можно использовать только для небольшого объема данных. Методы 1) и 3) работают с хорошей скоростью и для большого объема данных. Тем не менее методы, которые являются ориентированными на произвольные КНФ, для решения текущей задачи не подходят. А методы, учитывающие специфику КНФ, позволяют найти кратчайшую установочную последовательность посредством сведения к задаче проверки выполнимости КНФ разрешения комбинационного блока. Был предложен к реализации оригинальный метод упрощения КНФ разрешения, удовлетворяющий требованиям.

4 Описание приложения

4.1 Обзор использованных технологий

4.1.1 Ubuntu

Ubuntu – операционная система, основанная на Debial GNU/Linux [10]. Разработчиком является компания Canonical. Первый выпуск был в октябре 2004 года. Первоначальное имя проекта было «No-Name-Yet» («пока еще нет имени»).

Ubuntu ориентирована на удобство и простоту использования, которая включает в себя широко распространенные утилиты «sudo» («substitute user and do» или «подменить пользователя и выполнить») и позволяет пользователям выполнять администраторские задачи, не запуская потенциально опасную сессию суперпользователя (специальный аккаунт в UNIX – подобных системах с идентификатором, владелец которого имеет право на выполнение всех без исключений операций).

Была использована интегрированная операционная система Ubuntu 16.04 LTS (дата выпуска версии 21 апреля 2016 года) в Windows 10, которая использовалась при сборке проектов SAT-решателей, с помощью системы cmake («cmake -DCMAKE_BUILD_TYPE=Release ..»). И используется для запуска SAT-решателей Riss7 и candy («./riss-core input.cnf output.txt»).

4.1.2 C Sharp

C Sharp (C#) – объектно-ориентированный язык программирования [11]. Был разработан группой инженеров под руководством Андерса Хейлсберга в 1998-2001 годах. Название «Си шарп» произошло от музыкальной нотации, где диез означает повышение соответствующего ноте звука на полутон. Этот язык программирования используется при разработке различных приложений, предназначенных для выполнения в среде .NET Framework. Синтаксис C Sharp наиболее близок к C++ и Java.

C Sharp поддерживает:

- полиморфизм;
- перегрузку операторов;
- делегаты;
- атрибуты;

- события;
- свойства;
- обобщенные типы и методы;
- итераторы;
- анонимные функции с поддержкой замыкания;
- LINQ;
- исключения;
- комментарии в формате XML.

Но C# не поддерживает множественное наследование классов, в отличии от C++.

Версии:

- версия 1.0 анонсирована в июне 2000 года, окончательный выход в январе 2002 года;
- версия 2.0 опубликована в октябре 2003 года компанией Microsoft, окончательный выход в ноябре 2005;
- версия 3.0 представлена августе 2007 года;
- версия 4.0 представлена в апреле 2010 года;
- версия 5.0 опубликована в июне 2013;
- версия 6.0 представлена в июле 2015;
- версия 7.0 анонсирована в марте 2017.

4.1.3 Visual Studio 2017

Microsoft Visual Studio (MVS) – среда разработки программного обеспечения, выпущена в начале марта 2017 года [12]. Позволяет воспользоваться всем набором функций для разработки кроссплатформенных решений. Этот выпуск Visual Studio доступен для отдельных разработчиков, для разработки проектов с открытым исходным кодом, академических исследований, образования и небольших групп специалистов. Среда позволяет создавать приложения для Windows, Android, iOS, веб-приложения и облачные службы. Так же можно выполнять сборку приложений для любой платформы.

Компоненты Visual Studio:

- Visual Basic .NET;
- Visual C++;
- Visual C#;

- Visual F++;
- и другие.

В таблице 6 представлена взаимосвязь версий C# со средами разработок программного обеспечения Visual Studio.

Таблица 6 – Версии C# с соответствующими им Visual Studio

Версия C#	Visual Studio
C# 1.0	Visual Studio 2002
C# 2.0	Visual Studio 2005
C# 3.0	Visual Studio 2008
C# 4.0	Visual Studio 2010
C# 5.0	Visual Studio 2012
C# 6.0	Visual Studio 2015
C# 7.0	Visual Studio 2017

4.1.4 SAT-решатели

SAT-решатель (SAT-солвер) – это решатель огромных булевых формул в форме КНФ. Он выдает ответ, есть ли набор входных значений, который удовлетворяет КНФ-выражению, и какие это значения должны быть.

В 1960 году Дэвис и Патнэм начали применять классические дедуктивные методы для решения SAT. В 1962 году Мартином Дэвисом, Хилари Патнэмом, Джорджем Логеманом и Дональдом Лавлендом был опубликован алгоритм DPLL. Алгоритм DPLL – это полный алгоритм поиска с возвратом для определения выполнимости булевых формул, записанных в виде КНФ, то есть для решения задачи КНФ-SAT. Данный алгоритм служил основой для многих SAT-решателей на протяжении десятилетий.

В 1996 голу Жуаном Маркесом-Сильвой и Каремом Сакаллой был разработан алгоритм CDCL (conflict-driven clause learning). Алгоритм основан на алгоритме DPLL и является эффективным решателем задач выполнимости булевых формул.

Начиная с 2001 года начали появляться Locality Based Search SAT-решатели, эффективно выбирающие подпространства для полного поиска на основе локальной информации, такие как BerkMin (Berkley-Minsk) и многие другие.

4.2 Диаграмма классов

Разработанная система состоит из базы данных файлов в формате «*.blif» и пяти классов:

1. «**NamesStr**» – соответствует именам переменных («names») и булевым значениям («namesBool») одного блока из файла «*.blif» («.names a b», «0 1»);
2. «**Blif**» – обрабатывает и преобразовывает данные, представленные в виде файла формата «*.blif». Свойства класса:
   - «input_z» – список переменных $(z_0^i, z_1^i, \ldots, z_k^i)$, которые соответствуют k линиям обратной связи последовательностной схемы;
   - «input_x» – список переменных $(x_1^i, x_2^i, \ldots, x_n^i)$, которые соответствуют входным полюсам;
   - «input_e» – список результирующих переменных, которые соответствуют выходным значениям отдельных элементов комбинационной схемы и обозначаются в виде элементов $(e_1^i, e_2^i, \ldots, e_l^i)$;
   - «outputs_» – список переменных $(y_1^i, y_2^i, \ldots, y_k^i)$, который задает значения выходных функций комбинационного блока;
   - «namesStr» – список, состоящий из имен переменных и их булевых значений;
3. «**IndexValue**» – текущий класс содержит данные об индексе («index_») и значении («value_») элементов из строк, соответствующих строкам в таблице КНФ разрешении;
4. «**Cnf_resolution**» – данный класс отвечает за формирование КНФ разрешения в виде таблицы, её упрощения, и проверки на выполнимость на каждом шаге алгоритма. Свойства класса:
   - «listNamesIndex» – список имен переменных;
   - «cnf_resolution» – КНФ разрешения, которая подвергается последующему упрощению;
   - «cnf_resolution_before» – копия КНФ разрешения «cnf_resolution» до момента упрощения;
   - «cnf_K» – конъюнкция, которая используется при упрощении для последнего шага алгоритма;
5. «**FormingFileCNF**» – используется для формирования файла в формате «*.cnf» для дальнейшей его обработки с помощью SAT решателей. Свойства класса:

- «cnf» – КНФ разрешения в формате «*.cnf», используется для подачи в SAT решатель;
- «cnfSolver» – установочная последовательность;
- «SAT» – результат выполнения SAT решателя («SATISFIABLE» или «UNSATISFIABLE»).

На рисунке 11 изображена полная диаграмма классов системы.

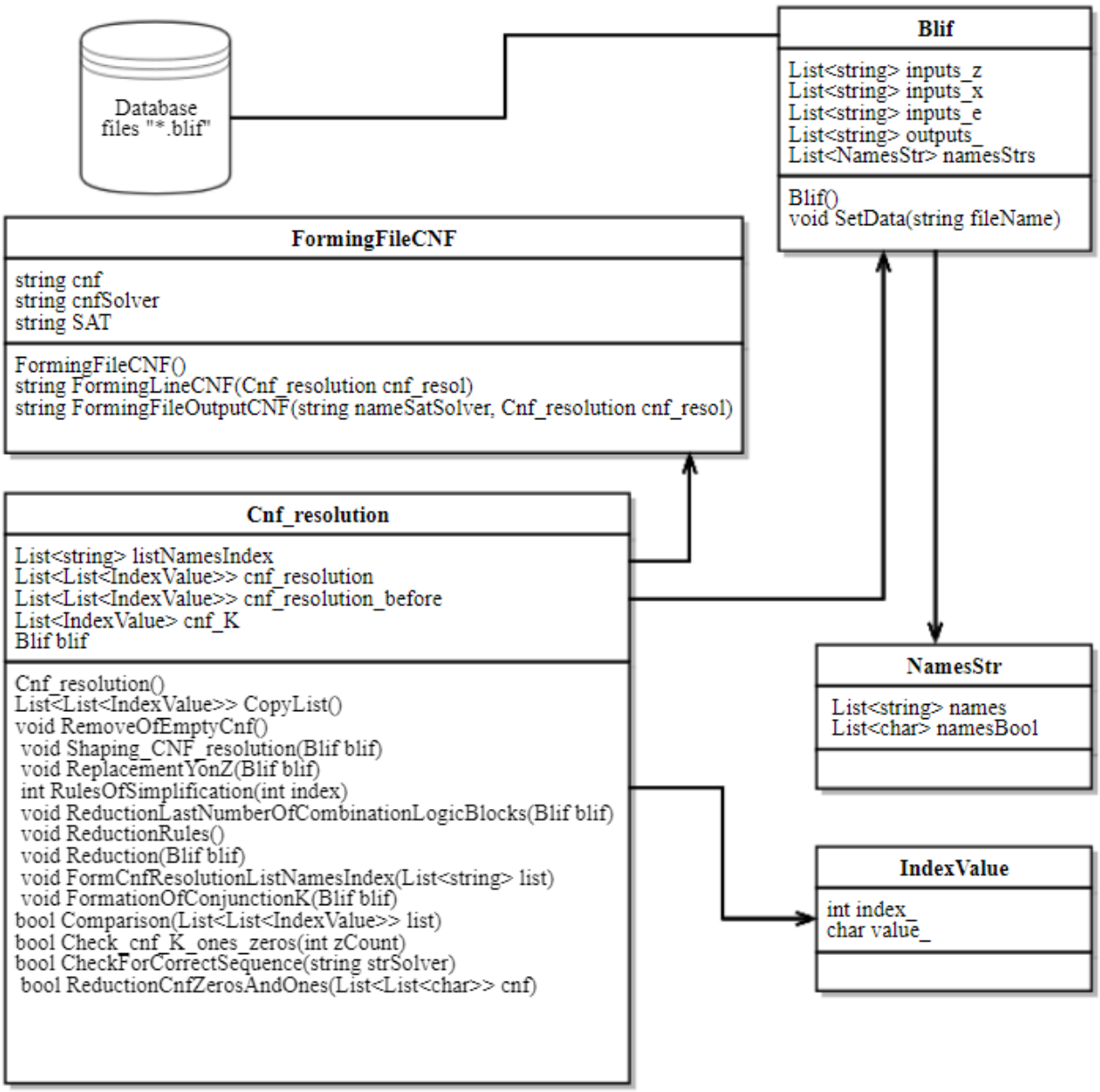


Рисунок 11 – Диаграмма классов приложения

4.3 Berkeley Logic Interchange Format (BLIF)

С помощью программной системы ABC, для синтеза и проверки двоичных последовательных логических схем, файлы формата «*.blif» были преобразованы в комбинационные составляющиеся из двух выходных и одно выходных вентилей. Преобразование производилось с помощью команды «strash». Большинство из порождающих вентилями дизъюнктов содержат по две литеры.

Пример файла «*.blif» до преобразования и после преобразования, с помощью программной системы ABC, представлен на рисунке 12.

```
.model 1-adder
.inputs cin a0 b0
.outputs cout s0
.names cin a0 b0 s0
001 1
010 1
100 1
111 1
.names cin a0 b0 cout
11- 1
1-1 1
-11 1
.end
```

```
# Benchmark "1-adder" written by ABC
.model 1-adder
.inputs cin a0 b0
.outputs cout s0
.names a0 b0 n5
11 1
.names a0 b0 n6
00 1
.names cin n6 n7
10 1
.names n5 n7 n8
00 1
.names n5 n6 n9
00 1
.names cin n9 n10
10 1
.names a0 b0 n11
01 1
.names a0 b0 n12
10 1
.names n11 n12 n13
00 1
.names cin n13 n14
00 1
.names n10 n14 n15
00 1
.names n8 cout
0 1
.names n15 s0
0 1
.end
```

Рисунок 12 – Примеры исходного и преобразованного файла в формате «*.blif»

4.4 Построение КНФ разрешения

Рассматривается многовыходная комбинационная составляющая синхронной схемы. Логические элементы схемы (вентили) реализуют элементарные логические функции: НЕ, И, ИЛИ, И-НЕ, ИЛИ-НЕ и другие.

КНФ разрешения для элементарных функций алгебры логики:

- $y = \bar{z} \to (z \vee y) \wedge (\bar{z} \vee \bar{y})$;
- $y = z_1 \wedge z_2 \wedge \ldots \wedge z_n \to (z_1 \vee \bar{y}) \wedge (z_2 \vee \bar{y}) \wedge \ldots \wedge (z_n \vee \bar{y}) \wedge (\bar{z}_1 \vee \bar{z}_2 \vee \ldots \vee \bar{z}_n \vee y)$;
- $y = z_1 \vee z_2 \vee \ldots \vee z_n \to (\bar{z}_1 \vee y) \wedge (\bar{z}_2 \vee y) \wedge \ldots \wedge (\bar{z}_n \vee y) \wedge (z_1 \vee z_2 \vee \ldots \vee z_n \vee \bar{y})$.

Примечание: КНФ разрешения инвертора $y = \bar{z}$ можно опустить с помощью замены всех вхождений переменной y на $\bar{z}$ в КНФ разрешения схемы.

Приложение позволяет выбрать необходимый файл из списка файлов и преобразовать его в нужный формат. КНФ разрешения преобразуется с помощью формул элементарных функций алгебры логики и представляется в виде таблицы. Пример таблицы представлен на рисунке 13.

C

z0	z1	x0	x1	x2	e0	e1	e2	e3	e4	e5	y0	y1
1	-	-	-	-	0	-	-	-	-	-	-	-
-	-	1	-	-	0	-	-	-	-	-	-	-
0	-	0	-	-	1	-	-	-	-	-	-	-
-	-	1	-	-	-	0	-	-	-	-	-	-
-	-	-	1	-	-	0	-	-	-	-	-	-
-	-	0	0	-	-	1	-	-	-	-	-	-
-	1	-	-	-	-	-	0	-	-	-	-	-
-	-	-	-	-	-	0	0	-	-	-	-	-
-	0	-	-	-	-	1	1	-	-	-	-	-
-	-	-	-	-	0	-	-	0	-	-	-	-
-	-	-	-	-	-	-	0	0	-	-	-	-
-	-	-	-	-	1	-	1	1	-	-	-	-
-	-	-	-	1	-	-	-	-	0	-	-	-
-	-	-	-	-	-	0	-	-	0	-	-	-
-	-	-	-	0	-	1	-	-	1	-	-	-
-	-	-	-	-	-	-	0	-	-	0	-	-
-	-	-	-	-	-	-	-	-	0	0	-	-
-	-	-	-	-	-	-	1	-	1	1	-	-
-	-	-	-	-	-	-	-	0	-	-	-	-
-	-	-	-	-	-	-	-	-	-	0	-	-

Рисунок 13 – Пример таблицы КНФ разрешения

Строкам таблицы соответствуют дизъюнкции между литерами. Например первая и вторая строки представлены в виде: $z_0 \vee \bar{e}_0$, $x_0 \vee \bar{e}_0$. КНФ разрешения комбинационной составляющей схемы представляет собой логическое перемножение КНФ разрешения всех её элементов (строк).

В процессе упрощения, описанного в разделе ниже, так же происходит построение КНФ разрешения для каждого блока комбинационной логики комбинационного эквивалента. Результат построения КНФ разрешения для нескольких блоков представлен на рисунке 14.

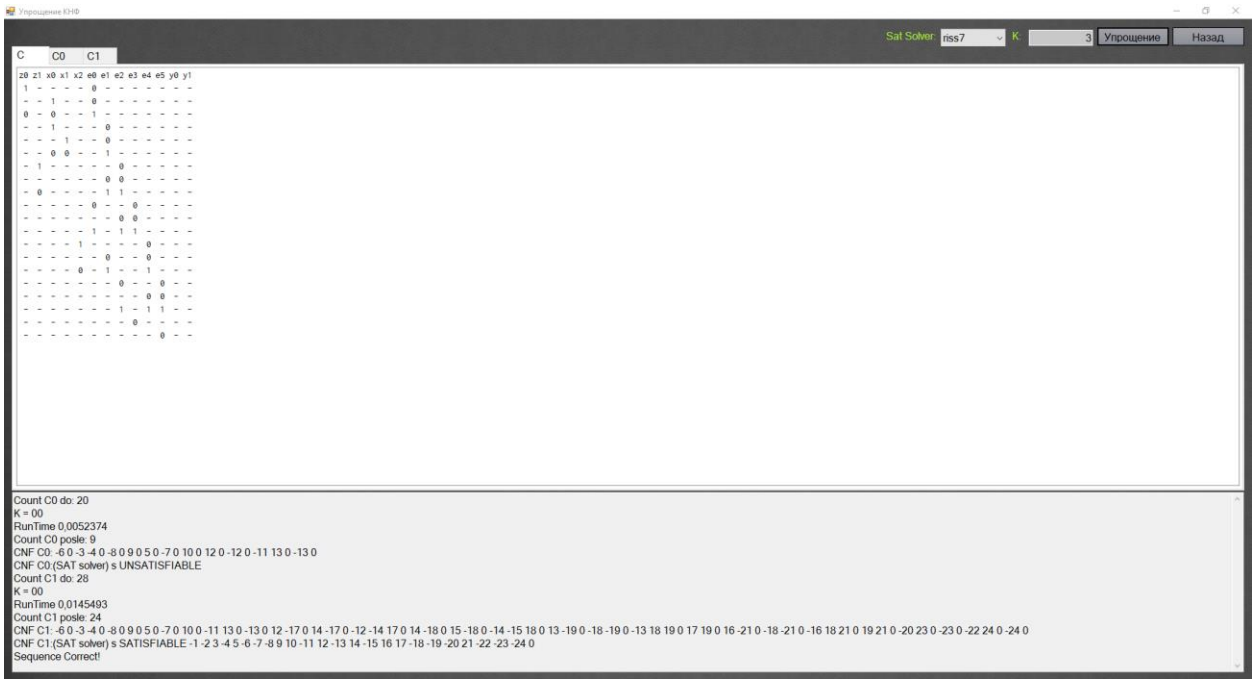


Рисунок 14 – Пример построения КНФ разрешения для нескольких блоков комбинационной логики

На рисунке выше блок «C» представляет собой исходную КНФ разрешения, блок «C0» –КНФ разрешения после упрощения, «C1» – упрощенная КНФ разрешения для следующего блока.

4.5 Преобразование КНФ разрешения к сертифицированному формату DIMACS CNF

Формат DIMACS CNF широко используется в качестве стандартного формата для представления булевых формул в КНФ. Файл данного формата начинается с задания количества литер и количества строк («p cnf 4 8»). Каждая из следующих строк задает

дизъюнкцию: положительный литерал обозначается соответствующим числом, а отрицательный литерал обозначается соответствующим отрицательным числом. Последнее число в строке должно быть равно нулю. Пример файла в формате DIMACS CNF представлен на рисунке 15.

```
p cnf 4 8
 1 2 -3 0
-1 -2 3 0
 2 3 -4 0
-2 -3 4 0
 1 3 4 0
-1 -3 -4 0
-1 2 4 0
 1 -2 -4 0
```

Рисунок 15 – Пример файла в формате DIMACS CNF

Примечание: первую строку можно игнорировать.

Преобразование КНФ разрешения к описанному выше формату необходимо для дальнейшей проверки на выполнимость с помощью SAT решателей.

4.6 Упрощение КНФ разрешения для блоков комбинационной логики комбинационного эквивалента

Перед проверкой на выполнимость исходную КНФ разрешения необходимо сократить. Упрощения КНФ разрешения происходят циклическим образом, то есть до тех пор, пока не будет достигнута выполнимость КНФ разрешения или не закончится заранее заданное количество блоков комбинационной логики комбинационного эквивалента.

Все упрощения происходят относительно следующих операций метода:

1. сначала фиксируются константами внутренние переменные первого блока в соответствие с заданным начальным состоянием синхронного автомата и формируется конъюнкция K;
2. далее в дизъюнктах последнего блока вычеркиваем переменные, инверсные присутствующим в конъюнкции K.

Операции 1, 2 приводят к образованию однолитерных дизъюнктов. Возникает возможность применения следующих правил упрощения:

- пусть A - одно литерный дизъюнкт, тогда $A \wedge (A \vee X) = A$;
- $A \wedge (\overline{A} \vee X) = A \wedge X$;

- $A \wedge A = A$.

Формирование конъюнкции K происходит после подсчета количества «1» и «0» для дизъюнкции с двумя литерами и переменных $(z_0^i, z_1^i, \ldots, z_k^i)$, которые соответствуют k линиям обратной связи последовательностной схемы. То есть конъюнкция $K = \bar{z}_0 \bar{z}_1 \ldots \bar{z}_k$, если количество «1» больше чем «0», иначе $K = z_0 z_1 \ldots z_k$. Построение таким образом конъюнкции K необходимо для эффективности упрощения КНФ разрешения.

После упрощения, полученная КНФ разрешения проверяется на выполнимость с помощью одного из SAT решателей (Riss7 или candy), представленных в выпадающем списке. Полученная КНФ разрешения может быть не выполнимой («UNSATISFIABLE») или выполнимой («SATISFIABLE»).

В процессе выполнения программы могут возникнуть несколько случаев:

1. КНФ разрешения выполнима, тогда заканчивается выполнение программы и на экран выводятся результаты, пример представлен на рисунке 16;

2. КНФ разрешения не выполнима, тогда происходит построение КНФ разрешения для следующего блока комбинационной логики. Построение начинается с отождествления в упрощенной КНФ разрешении, полученной на предыдущем шаге, выходов $y_0^i, y_1^i, y_2^i, \ldots, y_k^i$ с входами $z_0^{i+1}, z_1^{i+1}, z_2^{i+1}, \ldots, z_k^{i+1}$, то есть $y_0^i = z_0^{i+1}$, $y_1^i = z_1^{i+1}$, $y_2^i = z_2^i$, …, $y_k^i = z_k^{i+1}$. Далее в полученную КНФ разрешения добавляется КНФ разрешения нового блока комбинационной логики. И начинается очередная операция упрощения с проверкой на выполнимость. Пример этого случая изображен на рисунке 17;

3. КНФ разрешения не выполнима по достижении заранее заданного количества блоков комбинационной логики комбинационного эквивалента, тогда в дизъюнктах последнего блока вычеркиваются переменные, инверсные присутствующим в конъюнкции $K = z_0 z_1 z_2 \ldots z_k$. Пример текущего случая изображен на рисунке 18.

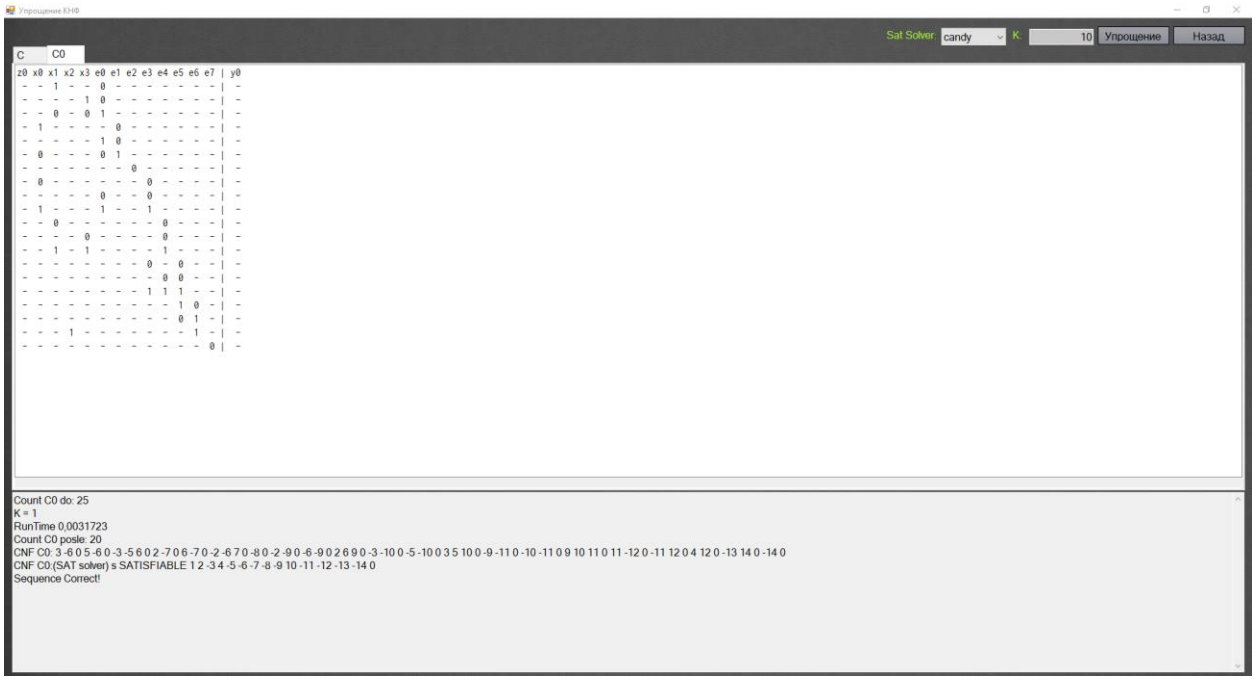


Рисунок 16 – Пример выполнимой КНФ разрешения

На рисунке 16 в нижнем блоке представляется количество строк в таблице КНФ разрешения до упрощения и после, а также представлена найденная установочная последовательность.

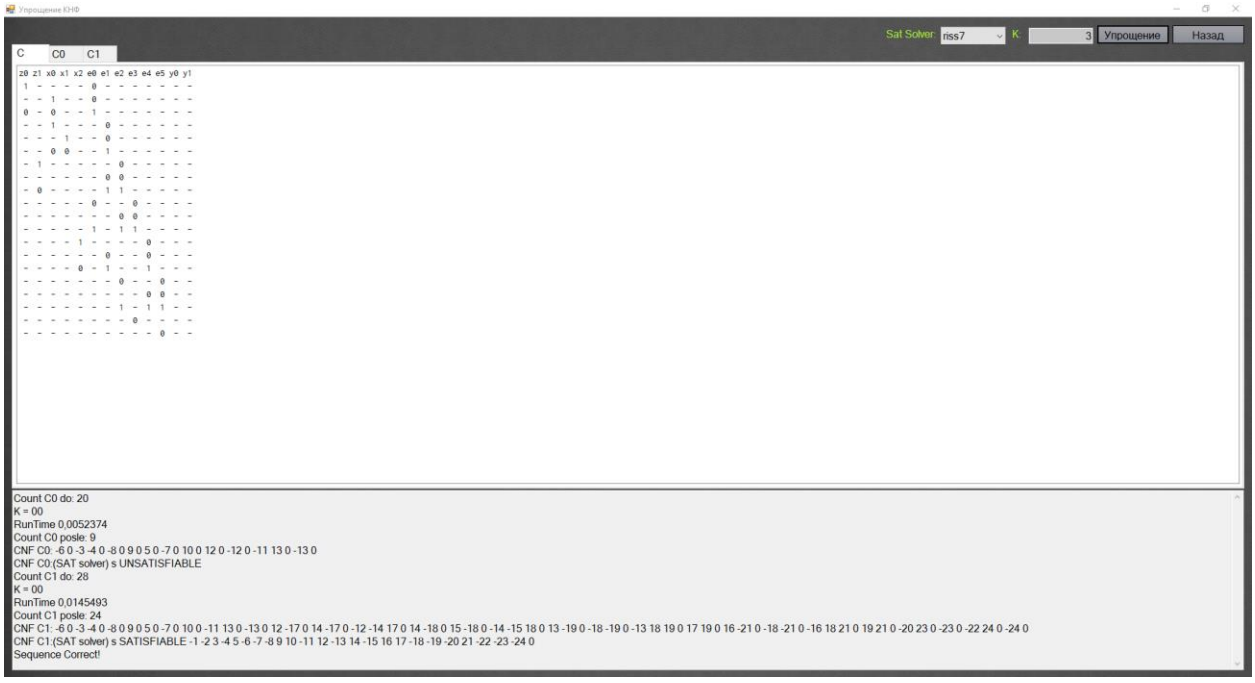


Рисунок 17 – Пример не выполнимой КНФ разрешения с построением КНФ разрешения для следующего блока комбинационной логики

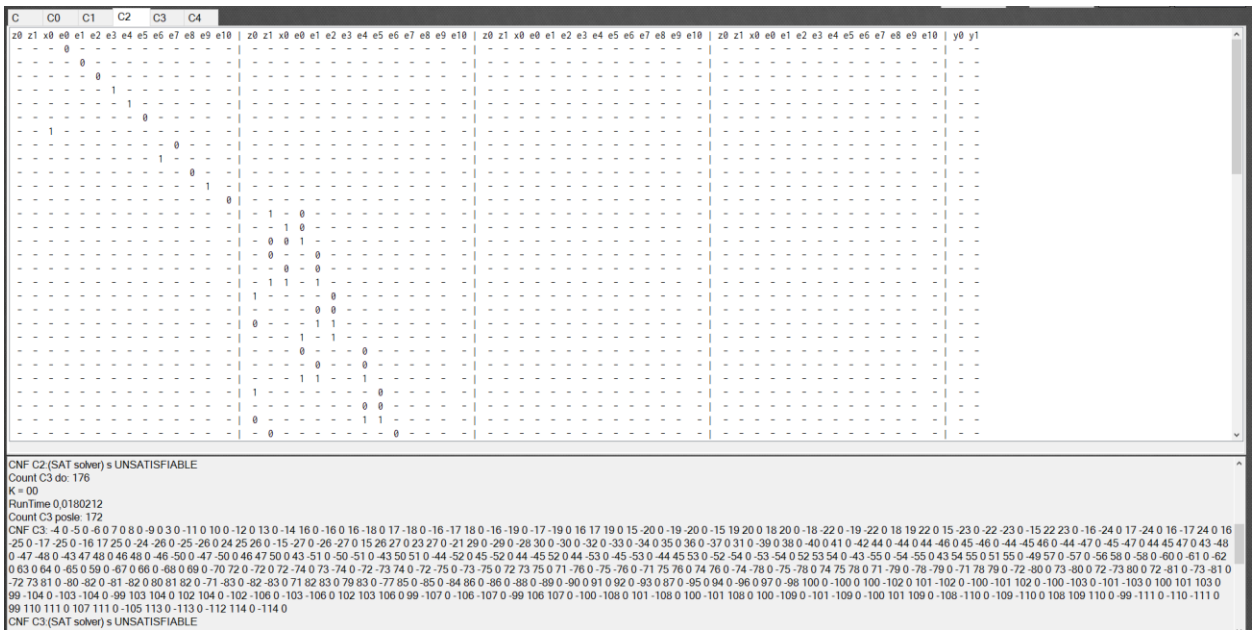


Рисунок 18 – Пример не выполнимой КНФ разрешения

4.7 Проверка на корректность установочной последовательности

В результате всех упрощений КНФ разрешения происходит формирование файла «*.cnf» для проверки на выполнимость и построения установочной последовательности с помощью SAT-решателей. Полученную установочную последовательность необходимо проверить на корректность. То есть установочную последовательность необходимо подставить в исходную КНФ разрешения, представленную в виде таблицы. Если на выходе после упрощения получается «1», то установочная последовательность корректна, иначе нет.

4.8 Выводы

Результаты выполнения выше описанного приложения представлены в таблице 7.

Введем обозначения:

- «t» - время выполнения упрощений в секундах;
- «SAT/UN» - результат выполнения SAT решателей. «SAT» - найдена выполнимая установочная последовательность для КНФ разрешения, «UN» КНФ разрешения не выполнима;
- «N» - заданное количество комбинационных блоков;
- «K» - длина установочной последовательности.

Таблица 7 – Результаты выполнения приложения поиска установочной последовательности

Название примера	Кол-во входов	Кол-во выходов	Название SAT решателя	Кол-во строк до упрощения	Кол-во строк после упрощения	t	SAT/ UN	N	K
majority	5	1	Riss7	25	20	0.005	SAT	10	1
			candy			0.006	SAT		1
C17	5	2	Riss7	20	9	0.005	UN	10	2
				28	24	0.006	SAT		
			candy	20	9	0.005	UN		2
				28	24	0.006	SAT		
parity	16	1	Riss7	136	130	0.005	SAT	10	1
			candy			0.005	SAT		1
pm1	16	13	Riss7	149	51	0.006	UN	10	2
				199	119	0.009	SAT		
			candy	149	51	0.006	UN		2
				199	119	0.008	SAT		
cu	14	11	Riss7	166	59	0.006	SAT	10	1
			candy			0.006	SAT		1
Cm150a	21	1	Riss7	184	179	0.006	SAT	10	1

			candy			0.006	SAT		1
ITC_b04	76	74	Riss7	1402	516	0.07	UN	10	2
				1847	1351	0.307	SAT		
			candy	1402	516	0.07	UN		2
				1847	1351	0.311	SAT		
01-adder	3	2	Riss7	35	13	0.006	UN	10	-
				47	43	0.006	UN		
				90	86	0.007	UN		
				176	172	0.008	UN		
				348	344	0.014	UN		
				692	688	0.034	UN		
				1380	1376	0.099	UN		
				2756	2752	0.343	UN		
				5508	5504	0.202	UN		
				11012	10991	0.74	UN		
			candy	35	13	0.005	UN		-
				47	43	0.006	UN		
				90	86	0.006	UN		
				176	172	0.008	UN		
				348	344	0.013	UN		
				692	688	0.031	UN		
				1380	1376	0.095	UN		
				2756	2752	0.337	UN		
				5508	5504	0.194	UN		
				11012	10991	0.682	UN		
C8	28	18	Riss7	517	267	0.014	UN	10	2
				776	545	0.031	SAT		
			candy	517	267	0.014	UN		2
				776	545	0.03	SAT		
z4ml	43	4	Riss7	145	51	0.007	UN	10	-
				195	187	0.008	UN		
				382	374	0.014	UN		
				756	748	0.036	UN		
				1504	1496	0.113	UN		
				3000	2992	0.374	UN		
				5992	5984	1.294	UN		
				11976	11968	5.059	UN		
				23944	23936	20.02	UN		
				47880	47872	80.15	UN		
			candy	145	51	0.006	UN		-
				195	187	0.008	UN		
				382	374	0.013	UN		
				756	748	0.033	UN		
				1504	1496	0.109	UN		
				3000	2992	0.375	UN		
				5992	5984	1.315	UN		
				11976	11968	4.936	UN		
				23944	23936	19.92	UN		
				47880	47872	79.86	UN		

Исходя из полученных результатов следует, что для сложных схем упрощения являются незначительными. А при использование разных SAT-решателей было выявлено, что решение и скорость выполнения одинаковые.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был произведен обзор существующих методов построения установочных последовательностей и представлен оригинальный метод, рассмотрены несколько методов упрощения КНФ.

Методы построения установочных последовательностей:

- поиск установочной последовательности;
- метод регулярных безусловных установочных экспериментов;
- метод построения установочной последовательности для конечных автоматов с помощью прямого различающего дерева;
- метод поиска кратчайшей установочной последовательности посредством сведения к выполнимости КНФ разрешения комбинационного блока, реализующего функции возбуждения триггеров;
- оригинальный метод поиска кратчайшей установочной последовательности.

Так как необходимо найти кратчайшую установочную последовательность, то последний метод поиска удовлетворяет этому требованию.

Методы упрощения КНФ:

1. методы, ориентированные на произвольные КНФ:
   - метод резолюции;
   - минимизации функции CQ;
   - Horn минимизация путем итеративного разложения;
2. методы, учитывающие специфику КНФ:
   - метод упрощения КНФ разрешения, представленный в работе [5];
   - метод упрощения, представленный в оригинальном методе поиска кратчайшей установочной последовательности.

Реализовано приложение, которое выполняет следующие операции:

- преобразование файла формата «*.blif» в нужную форму КНФ разрешения для каждого блока комбинационной логики комбинационного эквивалента, представленную в виде таблицы;
- реализация метода вывода КНФ разрешения в виде таблицы;
- упрощение КНФ разрешения;
- преобразование КНФ разрешения к сертифицированному формату DIMACS CNF;
- проверка на выполнимость с помощью SAT решателей (Riss7 или candy);

- нахождение кратчайшей установочной последовательности с помощью SAT решателя;
- проверка установочной последовательности на корректность.

Было принято участие в VI-й Молодежной научной конференции «Математическое и программное обеспечение информационных, технических и экономических систем» с докладом «Исследование и разработка алгоритмов упрощения КНФ при решении проблемы выполнимости» в рамках секции «Математическое и программное обеспечение вычислительных машин и компьютерных сетей». По окончании конференции планируется печатное издание материалов докладов в сборнике «Труды Томского государственного университета» (с постатейным размещением в базе РИНЦ), а также размещение электронного варианта издания в открытом доступе (домен «tsu.ru»).

ЛИТЕРАТУРА

1. A. Matrosova, V. Andreeva, A. Melnikov. ROBDDs Application for Finding the Shortest Transfer Sequence of Sequential Circuit or Only Revealing Existence of this Sequence without Deriving the Sequence itself // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2016). Kharkov: IEEE Computer Society, 2016. P. 513–516.

2. Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions / Department of Computer Science Series of Publications B, University of Helsinki, 2017. – 201 p.

3. Lee D. Principles and methods of testing finite state machine – a survey / D. Lee, M. Yannakakis // Proc. of the IEEE. – 1996. – P. 1090–1123.

4. Гилл А. Введение в теорию конечных автоматов / А. Гилл. – М. : Наука, 1966. – 272 с.

5. Черемисинова Л.Д. Поиск кратчайшей установочной последовательности схемы с памятью на D-триггерах // ВЕСЦІ НАЦЫЯНАЛЬНАЙ АКАДЭМІІ НАВУК БЕЛАРУСІ. СЕРЫЯ ФІЗІКА-МАТЭМАТЫЧНЫХ НАВУК. – 2015. – С. 119–128.

6. SAT Competition 2017 [Электронный ресурс]. – URL: https://baldur.iti.kit.edu.

7. Хныкин И.Г. Модификация КНФ, эквивалентных задачам криптоанализа асимметричных шифров методом резолюции // ИТМУ. – 2007. – № 8. – 2 с.

8. Boros E. Horn minimization by iterative decomposition / E. Boros, O. Cepek, A. Kogan // Annals of Mathematics and Artificial Intelligence. – 1998. – V. 23. – P. 321–343.

9. A subclass of Horn CNFs optimally compressible in polynomial time / E. Boros [et al] // Annals of Mathematics and Artificial Intelligence. – 2009. – V. 57. – P. 249–291.

10. Ubuntu [Электронные ресурс] // Википедия: Свободная энцикл. – Электрон.дан. – [Б. м.], 2018. URL: https://ru.wikipedia.org/wiki/Ubuntu.

11. C Sharp [Электронные ресурс] // Википедия: Свободная энцикл. – Электрон.дан. – [Б. м.], 2018. URL: https://ru.wikipedia.org/wiki/C_Sharp.

12. Microsoft Visual Studio [Электронные ресурс] // Википедия: Свободная энцикл. – Электрон.дан. – [Б. м.], 2018. URL: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio.

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: neldnelaps@gmail.com / ID: 1896880
Проверяющий: (neldnelaps@gmail.com / ID: 1896880)

Отчет предоставлен сервисом «Антиплагиат»- http://www.antiplagiat.ru

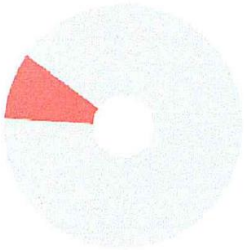
ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 19
Начало загрузки: 04.06.2018 12:22:50
Длительность загрузки: 00:03:28
Имя исходного файла:
BKP_mag_PashkovaNN_1162m
Размер текста: 1828 кБ
Символов в тексте: 58122
Слов в тексте: 7317
Число предложений: 279

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 04.06.2018 12:26:19
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска:

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
9,22%	0%	90,78%



Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Доля в тексте	Источник	Ссылка	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте
[01]	1,69%	1,69%	Регулярные безусловные ус...	http://refwin.ru	30 Мая 2016	Модуль поиска Интернет	9	9
[02]	1,6%	1,6%	ВЫЧИСЛИТЕЛЬНЫЕ МЕТОД...	http://lib.tsu.ru	раньше 2011	Модуль поиска Интернет	6	6
[03]	0%	1,6%	ЭКВИВАЛЕНТНОЕ ПРЕОБРАЗ...	http://cyberleninka.ru	13 Июн 2013	Модуль поиска Интернет	0	6

Еще источников: 12
Еще заимствований: 5,91%

С. А. Останин