УДК 681.3

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ТРАНСФОРМАЦИИ СХЕМ АЛГОРИТМОВ И ПРОГРАММ

А.С. Мохница

Институт программных систем НАН Украины, 03187, Киев, просп. Академика Глушкова, 40
e-mail: mohnitsa@isofts.kiev.ua

Статья посвящена инструментарию трансформации схем алгоритмов и программ, разрабатываемому в рамках исследований по алгебраической алгоритмике. Демонстрируется его применение на доказательстве гипотезы о тождественности двух булевых функций.

Paper is dedicated to the review of the toolkit for transformation of schemas of algorithms and programs, developed within the framework of studies on algebraic algorithmics. Use of the toolkit is demonstrated on the proof of hypothesis of Boolean functions equality.

Введение

Алгебраическая алгоритмика является одним из перспективных направлений современной компьютерной науки [1]. Основополагающие работы В.М. Глушкова по теории систем алгоритмических алгебр (САА) [2, 3] заложили фундамент для исследований в данной области в Украине. Современное состояние данных исследований отражено в [4, 5].

Немаловажным является также создание и развитие современных инструментальных средств синтеза алгоритмов и программ в объектных средах [6-8]. В число упомянутых также входит инструментарий, предназначенный для трансформации схем алгоритмов и программ (далее – Трансформатор) с целью улучшения по выбранным пользователем критериям.

Материал статьи подчинен следующей структуре. В разд. 1 приведено краткое описание архитектуры инструментальных средств синтеза алгоритмов и программ в объектных средах. Дается характеристика Трансформатора как составной части упомянутых средств. В разд. 2 продемонстрировано применение Трансформатора для доказательства гипотезы, представленной формулами в алгебре Буля. В заключении очерчены перспективы развития Трансформатора в рамках алгебраической алгоритмики.

1. Инструментальные средства синтеза алгоритмов и программ в объектных средах и трансформация схем

В данном разделе дано описание архитектуры инструментальных средств синтеза алгоритмов и программ в объектных средах. Приведена характеристика Трансформатора как составной части упомянутых инструментальных средств. Пояснена целесообразность интегрированного использования средств алгебры алгоритмики и объектно-ориентированного проектирования (диаграмм языка UML). Описаны стратегии подобного совместного использования.

1.1. Инструментальные средства синтеза алгоритмов и программ в объектных средах базируются на алгебро-алгоритмических спецификациях [6], позволяя строить синтаксически правильные последовательные и параллельные САА-схемы алгоритмов и выполнять синтез соответствующих объектно-ориентированных программ [7, 8]. Кроме того, поддерживаются современные системы визуализированного проектирования объектно-ориентированных программ (UML, Rational Rose и др.) [9]. Подробнее об этом – в пп. 1.4.

Особенность инструментария алгебры алгоритмики состоит во взаимосвязанном применении всех трех представлений алгоритма [8], что дает более полную картину о конструируемом алгоритме, чем каждое из них в отдельности. При этом изменение любого из этих представлений в инструментарии автоматически приведет к соответствующему преобразованию остальных.

1.2. Инструментальные средства синтеза алгоритмов и программ в объектных средах состоят из компонентов, представленных на рис. 1.

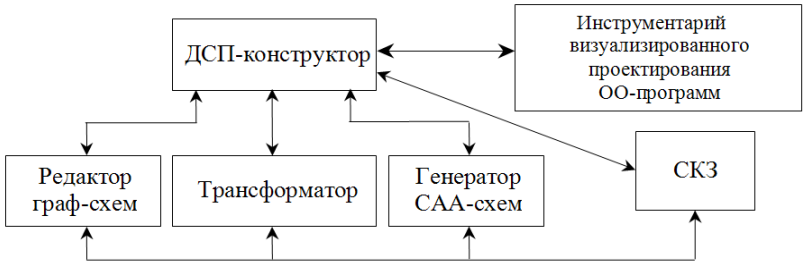

Рис. 1. Архитектура инструментария синтеза алгоритмов и программ в объектных средах

Первый компонент инструментария – ДСП-конструктор [6]. В основу его функционирования положен диалоговый режим с использованием меню подстановок и FIFO (память типа очередь). Меню состоит из операторных и логических конструкций, суперпозиция которых позволяет создавать алгоритмы в упомянутых выше взаимосвязанных представлениях. Операторные и логические переменные запоминаются в очереди с дальнейшей их детализацией. В зависимости от типа переменной, считанной из очереди, система предлагает соответствующую компоненту меню или открывает для пользователя архив базисных понятий.

Отметим поуровневый стиль конструирования алгоритма, а также возможность переходов на различные уровни с продолжением процесса диалогового конструирования, причем подобный переход сопровождается соответствующим изменением состояния FIFO.

Назначение Трансформатора – преобразование аналитических спецификаций алгоритмов (формул в различных алгебрах) с использованием соотношений и тождеств соответствующей алгебры. Подробнее об этом компоненте – в пп. 1.3.

Генератор САА-схем ориентирован на параметрически управляемый синтез формализованных спецификаций параллельных и последовательных алгоритмов и программ [7, 8]. Генерация выполняется посредством схем более высокого уровня, называемых регулярными гиперсхемами (РГС). РГС применяются, в частности, для представления алгоритмов управления выводом в грамматиках структурного проектирования (ГСП). Проектирование гиперсхем, как и САА-схем, выполняется в диалоговом режиме.

Редактор граф-схем предназначен для редактирования внешнего вида граф-схем алгоритмов и программ, создаваемых в процессе ДСП-конструирования и, возможно, модифицируемых при диалоговой трансформации. При этом изменения, внесенные в процессе редактирования, соответствующим образом отображаются на остальных представлениях (аналитическом, естественнолингвистическом).

Следующий компонент, среда конструирования алгоритмических знаний (СКЗ) инструментария, содержит в себе следующие элементы, предназначенные для построения регулярных схем алгоритмов, а также синтеза программ:

- стратегии обработки (регулярные схемы, описывающие классы алгоритмов);
- метаправила свертки, развертки и трансформации (обеспечивают абстрагирование, детализацию и переинтерпретацию схем);
- базисные предикаты и операторы, представленные в трех формах, и их программные реализации (ориентированные на проектирование алгоритмов и синтез программ в данной предметной области на избранном целевом языке);
- алгоритмы символьной обработки, представленные в трех формах;
- распределенные гиперсхемы.

1.3. Как известно, аппарат алгебры алгоритмики, на который опираются разрабатываемые инструментальные средства, позволяет представлять алгоритмы в трех взаимосвязанных формах: аналитической (формульной), удобной для трансформации, в частности для оптимизации по выбранным критериям (память, быстродействие и др.), граф-схемной, преимуществом которой является наглядность, и текстовой (естественнолингвистической), удобной для диалогового проектирования.

Кроме того, алгебра алгоритмики использует метаправила проектирования схем: свертку (абстрагирование), развертывание (детализацию), переинтерпретацию (сочетание свертки и развертки) и трансформацию, заключающуюся в улучшении схем на основе применения аппарата тождеств, квазитождеств и соотношений, характеризующих свойства операций алгебры алгоритмов и избранной предметной области. Метаправило трансформации вместе со сверткой и разверткой представляет собой мощный инструмент для установления взаимосвязей между различными алгоритмами и конструирования новых алгоритмов, более эффективных по тем или иным критериям.

В рамках работы над инструментальными средствами синтеза алгоритмов и программ в объектных средах с помощью упомянутого аппарата был спроектирован и реализован на языке программирования Delphi инструментарий трансформации схем алгоритмов и программ (Трансформатор).

Основным назначением Трансформатора является преобразование аналитических спецификаций алгоритмов (формул в различных алгебрах, например в САА Глушкова или в алгебре Буля) с использованием соотношений и тождеств соответствующей алгебры.

В основу функционирования Трансформатора положен алгоритм последовательной статической декомпозиции ДЕК/С [6]. Во время статической декомпозиции форма (левая или правая часть тождества) накладывается на цепочку (преобразуемую формулу) или ее уже зафиксированную подцепочку с разбиением на составляющие, служащие значениями параметров формы. Полученные в результате декомпозиции значения переменных формы присваиваются соответствующим переменным противоположной части (правой или левой части) тождества, которая заменяет найденную подцепочку.

Внешний вид Трансформатора представлен на рис. 2. Интерфейс приложения состоит из следующих элементов:

- меню тождеств и соотношений, используемых в процессе трансформации и хранящихся в базе данных Трансформатора (а также в СКЗ);

- поля редактирования, содержащего трансформируемое аналитическое выражение на данном этапе преобразования, а также на всех предыдущих этапах;
- поля редактирования, отображающего информацию о правилах (соотношениях), примененных к трансформируемому выражению для каждого этапа процесса преобразования.

Кроме того, программа предлагает отдельное окно для ввода новых или редактирования существующих тождеств с сохранением в базе тождеств и соотношений.

Кратко опишем каждый из упомянутых выше элементов.

Меню тождеств и соотношений представляет собой таблицу, каждая строка которой содержит в отдельных колонках правую часть формулы тождества (соотношения), левую его часть, контекст применения (управляющий контекст) выбранного правила к трансформируемому выражению для любого нетерминального символа данного правила, а также строку комментария.

Характерной особенностью данного элемента является то, что на любом этапе преобразования для любого тождества (соотношения) существует возможность изменения управляющего контекста с сохранением в базе тождеств. Кроме того, существует возможность применения выбранного правила к выделенному фрагменту преобразуемого выражения.

Одной из особенностей поля, отображающего текущее состояние трансформируемого аналитического выражения (и, кроме того, истории предыдущих состояний), является возможность в любой момент вернуться на любой из предшествующих этапов с восстановлением соответствующего состояния остальных элементов интерфейса.

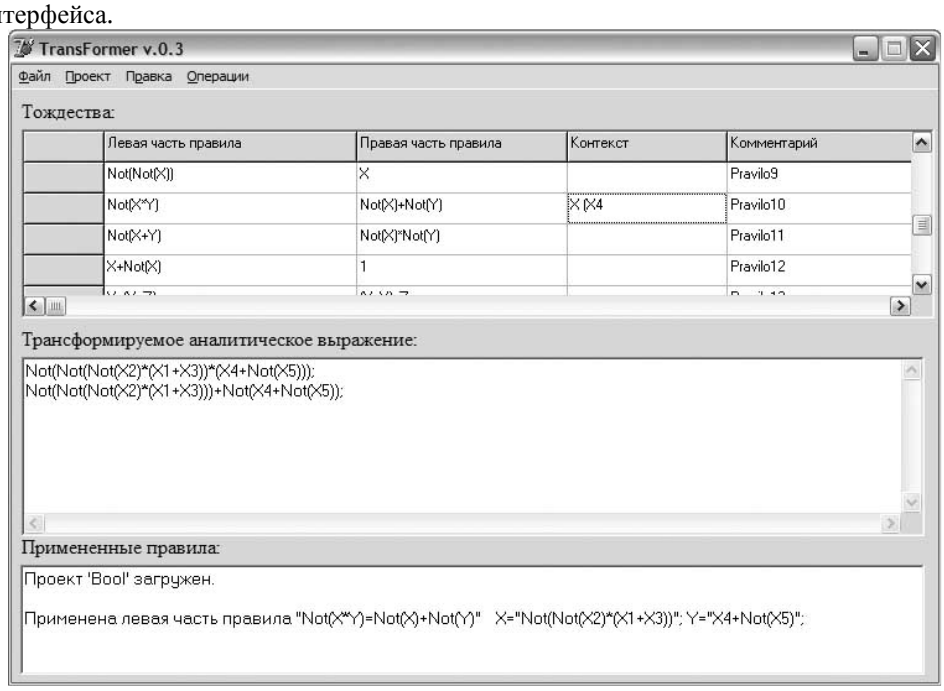

Рис. 2. Внешний вид инструментальных средств трансформации схем алгоритмов и программ

Третий элемент является служебным полем и отображает информацию о том, правая или левая часть тождества была применена на данном этапе, а также значения, полученные переменными правила в процессе его применения.

Процесс трансформации начинается с загрузки выражения, подлежащего преобразованию, и базы тождеств и соотношений, которые будут использованы в процессе работы.

Далее пользователю предлагается меню тождеств для применения к трансформируемому алгоритму с последующим выбором правой или левой его части и/или редактированием управляющего контекста. Если выбранное тождество применимо к преобразуемому выражению, то результат (проинтерпретированная левая или правая часть равенства) отображается в соответствующих полях, иначе выводится сообщение о неприменимости данного тождества при существующем контексте.

Как уже упоминалось, на любом этапе работы существует возможность отменить результаты последнего преобразования и вернуться на предыдущий шаг либо на один из предшествующих. Также процесс трансформации может быть прерван по желанию пользователем с сохранением полученных результатов, состояния служебных полей и базы тождеств и продолжен в любое удобное для него время с восстановлением сохраненных данных.

1.4. Язык UML является общецелевым языком визуального моделирования и предназначен для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем [9]. Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее общей и абстрактной концептуальной модели исходной системы к логической, а потом физической модели соответствующей программной системы. Все

представления о модели сложной системы фиксируются в виде специальных графических конструкций, называемых диаграммами. Непосредственно в инструментарии используются диаграммы классов, деятельности и компонентов [8, 9].

Диаграмма классов предназначена для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования и может отображать, в частности, различные связи между отдельными сущностями предметной области, такими, как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений.

Диаграммы деятельности наиболее подобны граф-схемам алгоритмов и используются, если при моделировании поведения проектируемой системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. В частности, диаграммы деятельности удобны для представления параллельных программ, поскольку они позволяют изобразить графически ветви процесса и их синхронизацию по времени выполнения.

Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и выполняемый код. Упомянутая диаграмма обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода.

К инструментальным средствам поддержки языка UML относится, в частности, семейство продуктов IBM Rational Rose [9], обеспечивающих разработчика программ полным набором инструментов визуального моделирования для эффективного решения сложных бизнес-задач с использованием архитектуры клиент-сервер, распределенных сред и систем реального времени. Rational Rose поддерживает генерацию кодов на различных языках программирования (Java, Visual Basic, C++), обратную генерацию диаграмм (реинжиниринг) на основе программного кода, выпуск проектной документации.

Следует отметить концептуальную близость средств проектирования программ, развиваемых в рамках алгебры алгоритмики, и языка моделирования UML.

Упомянутым средствам присущи:

- ориентация на описание программ в визуализированном виде (граф-схемы и диаграммы соответственно);
- независимость средств проектирования от конкретного языка программирования;
- ориентация на разнообразные прикладные задачи;
- развитие инструментальных средств для автоматизированной (а также автоматической) генерации текстов программ.

Результат моделирования в Rational Rose избранной предметной области с использованием диаграмм классов представляет собой сгенерированный “каркас” программного продукта на целевом языке программирования. Однако разработкой и кодированием собственно методов этих классов приходится заниматься вручную. Помочь разработчику в данном случае предназначен предлагаемый в данном разделе инструментарий.

Необходимо отметить, что диаграммы UML, дополненные граф-схемами алгоритмов, существенным образом обогащают средства визуального проектирования при реализации программных модулей. Благодаря наглядности граф-схем (при условии их компактности) и многоуровневости представления алгоритмов достигается независимость созданного таким образом проекта от кодирования на конкретном языке программирования. Кроме того, на этом этапе при переходе от диаграмм к программному коду упомянутый инструментарий может быть использован в качестве средства автоматизации.

Отметим, что дополняющим к изложенному подходу является использование средств алгебры алгоритмики (включая граф-схемы и интегрированный инструментарий) на начальных этапах проектирования с дальнейшим использованием диаграмм UML и изложенным выше переходом к объектно-ориентированным программам. Таким образом, речь идет о взаимодополняющих стратегиях проектирования объектно-ориентированных программ: “вниз” и “вверх”, а также об их комбинированном применении.

Объединение разработанного интегрированного инструментария, использующего аппарат граф-схем, и средств визуализированного проектирования объектно-ориентированных программ (Rational Rose) предоставляет пользователю комплект, позволяющий сопровождать разрабатываемый проект на всех стадиях жизненного цикла программной системы.

2. Применение инструментальных средств трансформации

Проиллюстрируем, каким образом Трансформатор может быть использован для диалогового доказательства гипотез.

Пусть необходимо доказать тождественность двух пятиместных булевых функций

$$f_1 = \overline{\overline{x}_2 \cdot (x_1 + x_3)} \cdot (x_4 + \overline{x}_5), \tag{1}$$

$$f_2 = \overline{x}_2 \overline{x}_4 x_5 + x_1 \overline{x}_2 + \overline{x}_2 x_4 x_5 + \overline{x}_2 x_3 . \tag{2}$$

Для доказательства необходимо привести обе функции к канонической форме, например к СДНФ. И если в результате этого получим одну и ту же СДНФ с точностью до перестановки конституэнт, значит, исходные функции тождественны.

Пример. Приведем (1) к каноническому виду.

Применим правило де Моргана для произведения к (1) (см. рис. 2):

$$\overline{\overline{\bar{x}_2 \cdot (x_1 + x_3)}} + \overline{(x_4 + \bar{x}_5)}.$$

Затем используем закон двойного отрицания для первого слагаемого и раскроем скобки, а ко второму слагаемому применим правило де Моргана для суммы, после чего применим закон двойного отрицания ко второму множителю:

$$x_1\bar{x}_2 + \bar{x}_2x_3 + \bar{x}_4x_5.$$

Теперь, в соответствии с алгоритмом канонизации, необходимо дополнить полученные слагаемые до пятиместных конъюнкций:

$$x_1\bar{x}_2(x_3 + \bar{x}_3)(x_4 + \bar{x}_4)(x_5 + \bar{x}_5) + \bar{x}_2x_3(x_1 + \bar{x}_1)(x_4 + \bar{x}_4)(x_5 + \bar{x}_5) + \bar{x}_4x_5(x_1 + \bar{x}_1)(x_2 + \bar{x}_2)(x_3 + \bar{x}_3).$$

Раскрыв скобки и использовав закон идемпотентности, получим (см. рис. 3):

$$x_1x_2x_3\bar{x}_4x_5 + x_1x_2\bar{x}_3\bar{x}_4x_5 + \bar{x}_1x_2x_3\bar{x}_4x_5 + \bar{x}_1x_2\bar{x}_3\bar{x}_4x_5 + x_1\bar{x}_2x_3\bar{x}_4x_5 + x_1\bar{x}_2\bar{x}_3\bar{x}_4x_5 + \bar{x}_1\bar{x}_2x_3\bar{x}_4x_5 +$$

$$+ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5 + x_1\bar{x}_2x_3x_4x_5 + x_1\bar{x}_2x_3x_4\bar{x}_5 + x_1\bar{x}_2x_3\bar{x}_4\bar{x}_5 + x_1\bar{x}_2\bar{x}_3x_4x_5 + x_1\bar{x}_2\bar{x}_3x_4\bar{x}_5 + x_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5 +$$

$$+ \bar{x}_1\bar{x}_2x_3x_4x_5 + \bar{x}_1\bar{x}_2x_3x_4\bar{x}_5 + \bar{x}_1\bar{x}_2x_3\bar{x}_4\bar{x}_5. \qquad (3)$$

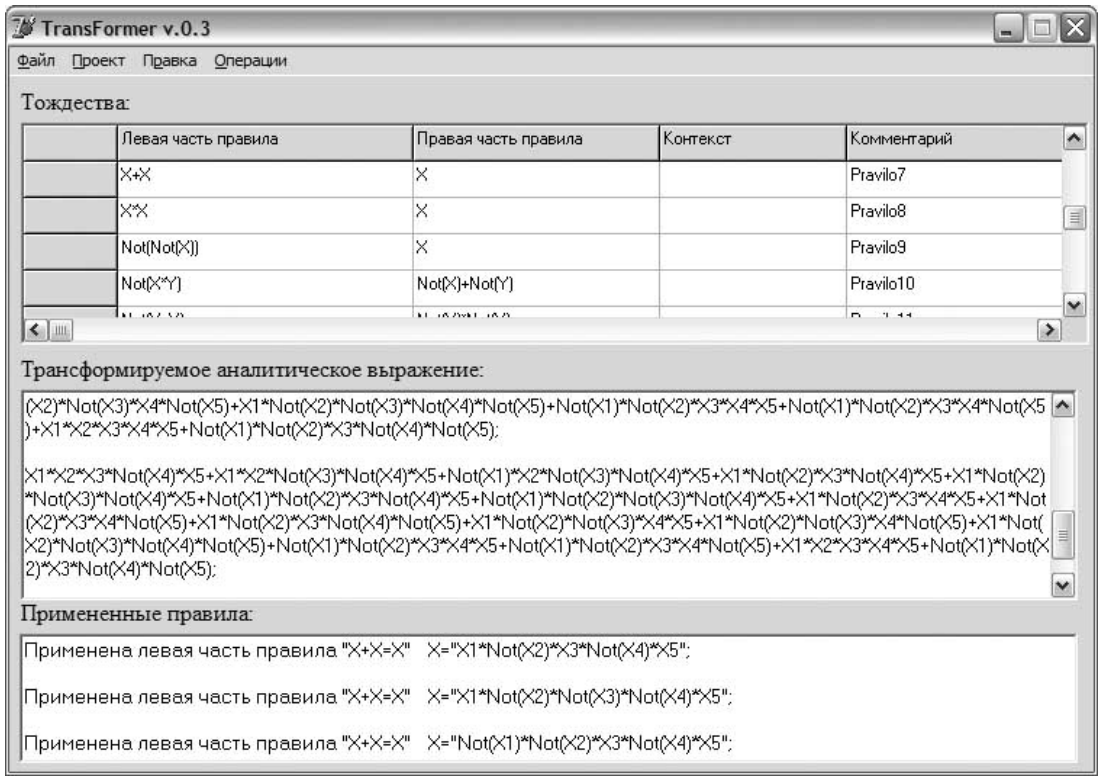

Рис. 3. Результат канонизации первой функции

Формула (3) представляет собой искомую СДНФ функции (1).

Приведем (2) к каноническому виду. Дополним слагаемые формулы до пятиместных конъюнкций:

$$\bar{x}_2\bar{x}_4x_5(x_1 + \bar{x}_1)(x_3 + \bar{x}_3) + x_1\bar{x}_2(x_3 + \bar{x}_3)(x_4 + \bar{x}_4)(x_5 + \bar{x}_5) +$$

$$+ \bar{x}_2x_4x_5(x_1 + \bar{x}_1)(x_3 + \bar{x}_3) + \bar{x}_2x_3(x_1 + \bar{x}_1)(x_4 + \bar{x}_4)(x_5 + \bar{x}_5).$$

Как и в случае с первой функцией, раскрываем скобки и используем закон идемпотентности (см. рис. 4):

$$x_1\bar{x}_2x_3x_4x_5 + x_1\bar{x}_2x_3x_4\bar{x}_5 + x_1\bar{x}_2x_3\bar{x}_4x_5 + x_1\bar{x}_2x_3\bar{x}_4\bar{x}_5 + x_1\bar{x}_2\bar{x}_3x_4x_5 + x_1\bar{x}_2\bar{x}_3x_4\bar{x}_5 + x_1\bar{x}_2\bar{x}_3\bar{x}_4x_5 +$$

$$+ \bar{x}_1\bar{x}_2x_3x_4x_5 + \bar{x}_1\bar{x}_2x_3x_4\bar{x}_5 + \bar{x}_1\bar{x}_2x_3\bar{x}_4x_5 + \bar{x}_1\bar{x}_2x_3\bar{x}_4\bar{x}_5 + x_1x_2x_3\bar{x}_4x_5 + x_1x_2\bar{x}_3\bar{x}_4x_5 + \bar{x}_1x_2x_3\bar{x}_4\bar{x}_5 +$$

$$+ \bar{x}_1x_2x_3\bar{x}_4x_5 + \bar{x}_1x_2\bar{x}_3\bar{x}_4x_5 + \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5\,. \qquad (4)$$

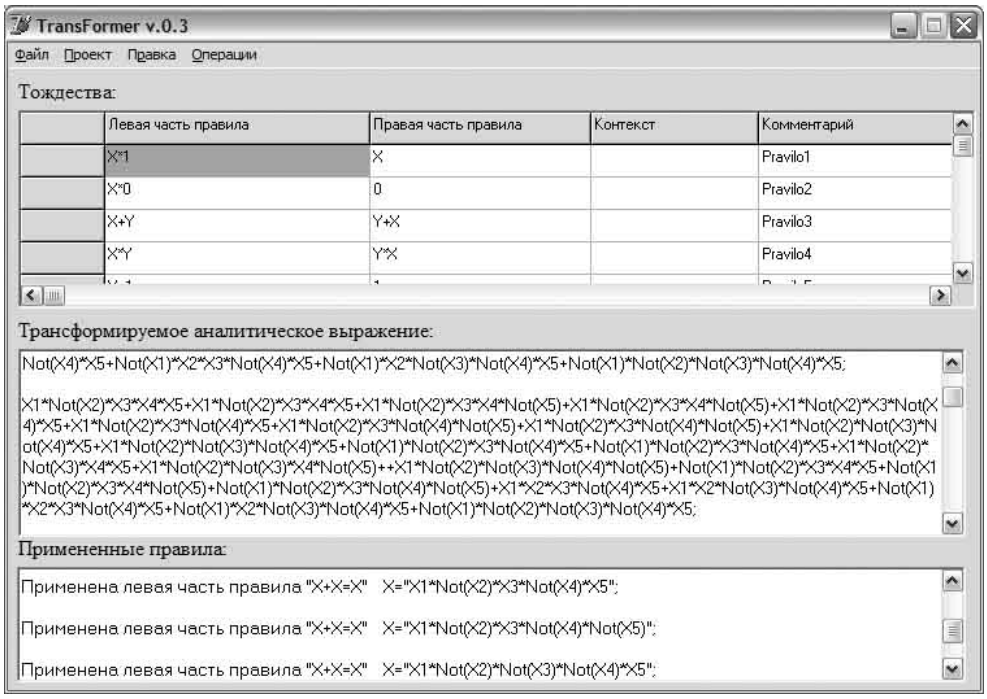

Рис. 4. Результат канонизации второй функции

Полученное выражение – СДНФ для функции (2). Сравнив (3) и (4), приходим к выводу, что функции (1) и (2) тождественны.

Приведенный пример демонстрирует, что Трансформатор может быть использован не только для оптимизации схем алгоритмов по выбранным критериям, но и для доказательства или опровержения гипотез. Необходимо отметить, что Трансформатор может быть использован и для доказательства в алгебре трехзначной логики, ассоциированной с алгеброй Глушкова.

Заключение

Таким образом, в статье дано описание архитектуры инструментальных средств синтеза алгоритмов и программ в объектных средах. Приведена характеристика инструментария трансформации схем алгоритмов и программ как составной части упомянутых инструментальных средств. Пояснена целесообразность интегрированного использования средств алгебры алгоритмики и объектно-ориентированного проектирования (диаграмм языка UML). Описаны стратегии подобного совместного использования. Проиллюстрированы на примере многоместных булевых функций возможности Трансформатора по доказательству гипотез.

Отметим, что Трансформатор может применяться и для доказательства в алгебре трехзначной логики, ассоциированной с алгеброй Глушкова.

В перспективе планируется ориентировать Трансформатор на решение ряда проблем параллельного программирования (в частности, выявление клинчей и фиктивных циклов).

1. *Ноден П., Китте К.* Алгебраическая алгоритмика (с упражнениями и решениями). – М.: Мир, 1999. – 720 с.
2. *Глушков В.М.* Теория автоматов и формальное преобразование микропрограмм // Кибернетика. – 1965. – №5. – С. 1-10.
3. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Алгебра. Языки. Программирование: 3-е изд., перераб. и доп. – Киев: Наук. думка, 1989. – 376 с.
4. *Цейтлин Г.Е. Алгебры* Глушкова и теория клонов // Кибернетика и системный анализ. – 2003. – №4. – С. 48-58.
5. *Цейтлин Г.Е., Мохница А.С.* Что такое алгебраическая алгоритмика? // Пробл. программирования. Спец. выпуск по материалам 4-й Междунар. науч.-практ. конф. по программированию УкрПРОГ'2004. – 2004. – №2-3. – С. 52-59.
6. *Цейтлин Г.Е.* Введение в алгоритмику – К.: Сфера, 1999. – 720 с.
7. *Цейтлин Г.Е., Яценко Е.А.* Элементы алгебраической алгоритмики и объектно-ориентированный синтез параллельных программ // Математические машины и системы. – 2003. – № 2. – С. 64-76.
8. *Яценко Е.А., Мохница А.С.* Инструментальные средства конструирования синтаксически правильных параллельных алгоритмов и программ // Проблемы Пробл. программирования. Спец. выпуск по материалам 4-й Междунар. науч.-практ. конф. по программированию УкрПРОГ'2004. – 2004. – №2-3. – С. 444-450.
9. *Боггс У., Боггс М.,* UML и Rational Rose. – М.: ЛОРИ, 2000. – 582 с.