Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

А.С. Ковалев, А.П. Лукьянова, А.А. Шалыто

Новый метод вычисления булевых формул

Проектная документация

Проект создан в рамках
“Движения за открытую проектную документацию”
http://is.ifmo.ru

Санкт-Петербург
2004

Оглавление

1. Введение

Известны различные методы программной и аппаратной реализаций булевых формул (БФ) [1]. В данной работе рассматривается наиболее интересный метод из описанных в работе [1], в котором объединяются идеи как программных, так и аппаратных методов реализации булевых формул. При этом сначала для формулы строится линейный бинарный граф (ЛБГ), по которому эта формула может быть программно вычислена. Граф преобразуется в схему – мультиплексорный каскад (МК) [1, 2]. Анализ формул, которые формируются на выходе каждого элемента каскада показывает, что при таком подходе булева формула реализуется не по подформулам, а по *фрагментам*. На основе возврата от аппаратной реализации к программной (моделируя МК) формируется принципиально новый способ программного вычисления булевых формул по фрагментам формулы, которые могут не являться ее подформулами.

Объясним разницу между подформулой и фрагментом формулы. Например, в формуле $x_1 \,\&\, x_2 \mid x_3 \,\&\, x_4$ выражение $x_2 \mid x_3 \,\&\, x_4$ не является подформулой, но является фрагментом формулы.

Вычисление БФ предлагаемым способом на каждом этапе добавляет по одному операнду в уже вычисленный фрагмент формулы, начиная с ее конца. Например, для формулы, указанной выше, будут последовательно вычислены значения следующих ее фрагментов: x_4, $x_3 \,\&\, x_4$, $x_2 \mid x_3 \,\&\, x_4$, и наконец, $x_1 \,\&\, x_2 \mid x_3 \,\&\, x_4$. Это позволяет вычислять БФ быстрее, чем при использовании операторного (например, польского) способа, и отказаться от вычисления по переходам, как это выполняется при применении ЛБГ.

1.1. Технологии, примененные в данной работе

При реализации данного проекта для построения линейного бинарного графа и мультиплексорного каскада были применены теория компиляторов [3, 4] и SWITCH-технология [5, 6]. Использование этих подходов позволило существенным образом сократить время, необходимое для разработки набора алгоритмов, решающих данную задачу.

1.2. Благодарности

Авторы выражают признательность А.А. Штучкину за образцовую работу [4], которая легла в основу второго раздела настоящего отчета. Стиль изложения материала в этой работе оказался настолько удачным, что авторы данного проекта решили заимствовать его.

Также авторы благодарны А.В. Ефремову, предоставившему возможность использовать разработанный им исходный код готового класса `vector`.

2. Проектирование

2.1. Постановка задачи

Задана произвольная булева формула в базисе {&, | , !}. Требуется решить три задачи. Первая состоит в построении для каждой формулы линейного бинарного графа [1]. Вторая задача – преобразование ЛБГ в линейную схему из мультиплексоров (multiplexer – MX) “2 в 1”, которая и называется мультиплексорным каскадом. И наконец, третья задача – визуализация нового способа вычисления булевых формул по их фрагментам, а не по подформулам.

2.2. Декомпозиция задачи

Разделим решение поставленной задачи на следующие четыре этапа (рис. 1).

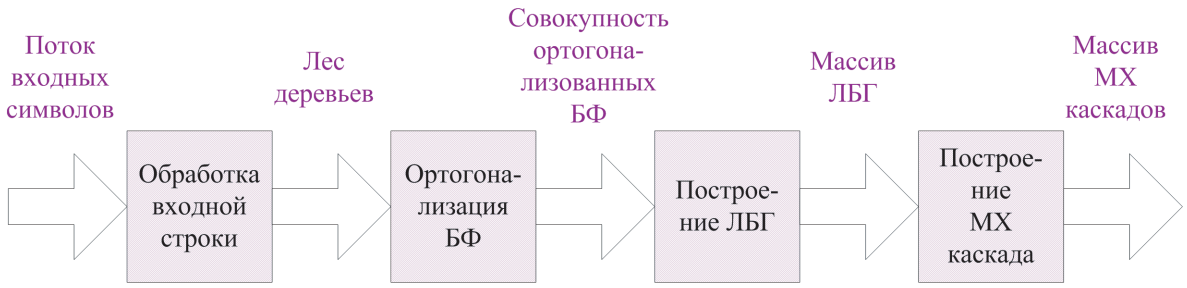

Рис. 1. Декомпозиция задачи

I. *Обработка входной строки* с набором булевых формул. Результатом является лес деревьев, каждое из которых однозначно восстанавливает булеву формулу [7].

II. *Преобразование булевой формулы.* Каждая булева формула сначала нормализуется, потом преобразуется в дизъюнктивную нормальную форму (ДНФ), которая приводится к ортогональной ДНФ (ОДНФ). Таким образом, результатом действий этого этапа является набор формул в ОДНФ.

III. *Построение линейного бинарного графа.* Результатом является набор линейных бинарных графов, каждый из которых соответствует одной из построенных на предыдущем этапе ОДНФ.

IV. *Построение мультиплексорного каскада.* Результат – набор мультиплексорных каскадов.

2.3. Обработка входной строки

Обработка входной строки выполняется на основе работы [4] и производится в три этапа (рис. 2).

Рис. 2. Обработка входной строки

I. *Лексический анализ.* На этом этапе из входного потока символов выделяются лексемы. Результат – поток лексем.

II. *Синтаксический анализ.* На этом этапе происходит разбор выражения и формируется программа для стековой машины. Параллельно проверяется корректность входного потока символов.

III. *Выполнение программы стековой машиной.* В результате работы стековой машины формируется лес двоичных деревьев, каждое из которых представляет булеву формулу.

2.4. Лексический анализатор

Лексический анализатор получает на вход последовательность символов и преобразует ее в поток лексем.

В данной программе лексемами являются:

- символы логических операций: **!**(логическое НЕ), **&**(логическое И), **|** (логическое ИЛИ);
- открывающая и закрывающая скобки: **(**, **)**;
- знак **;**, которым должна оканчиваться каждая формула;
- переменная **V**(с одним атрибутом);
- стоп-символ **$**.

Лексический анализатор читает посимвольно входной поток, “собирая” из символов лексемы. Распознавая переменную, он добавляет ее имя (совокупность символов, обозначающих ее) в словарь. Атрибут лексемы ‘переменная’ **V** представляет собой порядковый номер имени этой переменной в словаре. Таким образом, граф переходов для автомата, реализующего лексический разбор, должен выглядеть следующим образом (рис. 3).

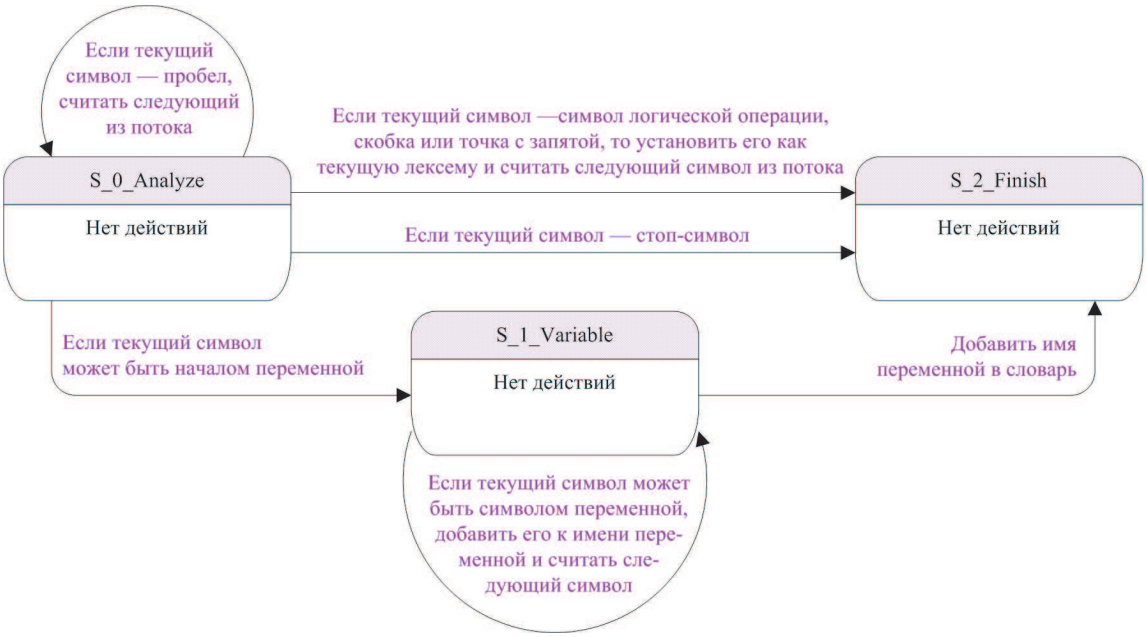

Рис. 3. Неформальный вид графа переходов для автомата A2_Lexical_analyzer

2.5. Синтаксический анализатор

Классически задача построения описанного выше леса решается построением AT-грамматики [3]. Используем решение [4], основанное на применении стековой машины.

Построим LL(1)-грамматику для описания списка булевых формул:

$$G = \{N, T, O, S, R\}.$$

Грамматика G состоит из набора нетерминальных символов

$$N = \{S, E, F, T, U, G\},$$

множества терминальных символов

$$T = \{\mathbf{!}, \&, \ |\ , (,), \mathbf{;}, \mathbf{V}, \mathbf{\$}\},$$

набора операционных символов

$$O = \{\mathbf{@A}, \mathbf{@C}, \mathbf{@n}, \mathbf{@a}, \mathbf{@o}\},$$

начального символа грамматики S и набора правил вывода

$$
\begin{array}{lrcl}
R = \{ & 1) & S & \to & E\mathbf{;@A}S \\
& 2) & S & \to & \varepsilon \\
& 3) & E & \to & TF \\
& 4) & F & \to & \ |\ T\mathbf{@o}F \\
& 5) & F & \to & \varepsilon \\
& 6) & T & \to & GU \\
& 7) & U & \to & \mathbf{\&}G\mathbf{@a}U \\
& 8) & U & \to & \varepsilon \\
& 9) & G & \to & \mathbf{!}G\mathbf{@n} \\
& 10) & G & \to & \mathbf{(}E\mathbf{)} \\
& 11) & G & \to & \mathbf{@CV} \qquad \}.
\end{array}
$$

Операционные символы служат для указания синтаксическому анализатору, что ему необходимо занести в стек определенные команды, которые впоследствии будут выполнены стековой машиной. В целях экономии памяти в данном проекте команды исполняются сразу же при обработке синтаксическим анализатором соответствующего операционного символа.

Опишем связанные с операционными символами команды:

- **@A**– добавить дерево в лес; адрес дерева считать из вершины стека стековой машины и вытолкнуть его;
- **@C**– создать дерево и поместить его адрес в стек стековой машины; корневую вершину пометить символом **V**; записать в поле атрибута атрибут текущей лексемы;
- **@a**– создать дерево; пометить его корневую вершину символом **&**; считать из стека адрес правого потомка для корневой вершины созданного дерева и вытолкнуть его; считать из стека адрес левого потомка для корневой вершины созданного дерева и вытолкнуть его; поместить в стек адрес этого дерева;
- **@o**– создать дерево; пометить его корневую вершину символом | ; считать из стека адрес правого потомка для корневой вершины созданного дерева и вытолкнуть его; считать из стека адрес левого потомка для корневой вершины созданного дерева и вытолкнуть его; поместить в стек адрес этого дерева;
- **@n**– создать дерево; пометить его корневую вершину символом **!**; считать из стека адрес правого потомка для корневой вершины созданного дерева и вытолкнуть его; поместить в стек адрес этого дерева.

Построенной грамматике G соответствует следующая управляющая таблица.

	!	&	|	(	)	V	;	$
S	1	ошибка	ошибка	1	ошибка	1	ошибка	2
E	3	ошибка	ошибка	3	ошибка	3	ошибка	ошибка
F	ошибка	ошибка	4	ошибка	5	ошибка	5	ошибка
T	6	ошибка	ошибка	6	ошибка	6	ошибка	ошибка
U	ошибка	7	8	ошибка	8	ошибка	8	ошибка
G	9	ошибка	ошибка	10	ошибка	11	ошибка	ошибка

Синтаксический анализатор работает следующим образом. Действия анализатора на каждой итерации зависят от текущей лексемы (ее синтаксический анализатор получает вызовом автомата лексического анализатора) и вершины стека синтаксического анализатора.

В том случае, если на вершине стека лежит нетерминал, анализатор обращается к управляющей таблице. Если на пересечении строки, соответствующей данному нетерминалу, и столбца, соответствующего текущей лексеме, лежит номер правила (не ошибка), то синтаксический анализатор выталкивает из стека этот нетерминал и заносит в стек соответствующую полученному номеру правую часть правила так, чтобы на вершине стека лежал самый левый символ. Если искомая клетка управляющей таблицы отмечена символом “ошибка”, то синтаксический анализатор выдает сообщение об ошибке.

В том случае, если на вершине стека лежит терминальный символ, синтаксический анализатор сравнивает его с текущей лексемой. В случае несовпадения символов анализатор выдает сообщение об ошибке. В противном случае анализатор выталкивает из стека этот терминальный символ и запускает лексический анализатор, который возвращает очередную лексему.

Если в вершине стека находится операционный символ, то анализатор дает стековой машине выполнить соответствующую этому символу команду.

Описанные действия повторяются до тех пор, пока на вершине стека не встретится стоп-символ $ или не возникнет ситуация, при которой автомат должен будет закончить работу с ошибкой.

Таким образом, поведение автомата, реализующего синтаксический анализ, может быть представлено следующим графом переходов (рис. 4).

Рис. 4. Неформальный вид графа переходов для автомата A1_Syntax_recognizer

2.6. Стековая машина

Стековая машина запрограммирована на выполнение пяти команд, описания которых были даны в предыдущем разделе. Подробную информацию об устройстве и принципах работы стековой машины можно найти в работах [3, 4]. Единственной особенностью данной стековой машины является дополнительный стек, в котором хранятся операнды, необходимые для выполнения команд.

2.7. Ортогонализация

Деревья используются для удобной ортогонализации булевых формул. Предлагается проводить ее в два этапа.

I. *Раскрытие скобок.* На этом этапе раскрываются скобки в формуле.

II. *Ортогонализация бесскобочного выражения.* На этом этапе происходит ортогонализация выражения вида

$$a_1 \& a_2 \& \ldots \& a_{n_1} \mid b_1 \& b_2 \& \ldots \& b_{n_2} \mid c_1 \& c_2 \& \ldots \& c_{n_3} \mid \ldots \tag{1}$$

Рассмотрим эти этапы подробнее.

Раскрытие скобок. Сначала необходимо преобразовать булеву формулу, “сдвинув” все отрицания непосредственно к переменным.

Например,

$$!(a_1 \mid a_2 \& a_3) \quad \rightarrow \quad (!a_1 \& (!a_2 \mid !a_3))$$

Теперь раскроем скобки. Сделать это можно, например, простым копированием:

$$a \& (a_1 \mid a_2 \mid ... \mid a_n) \& b \quad \rightarrow \quad a \& a_1 \& b \mid a \& a_2 \& b \mid ... \mid a \& a_n \& b$$

Оказывается, что при представлении булевой формулы в виде дерева, описанные выше операции раскрытия скобок и снятия отрицания реализуются достаточно просто. Проиллюстрируем следующими рисунками вторую из этих операций (рис. 5, рис. 6, рис. 7).

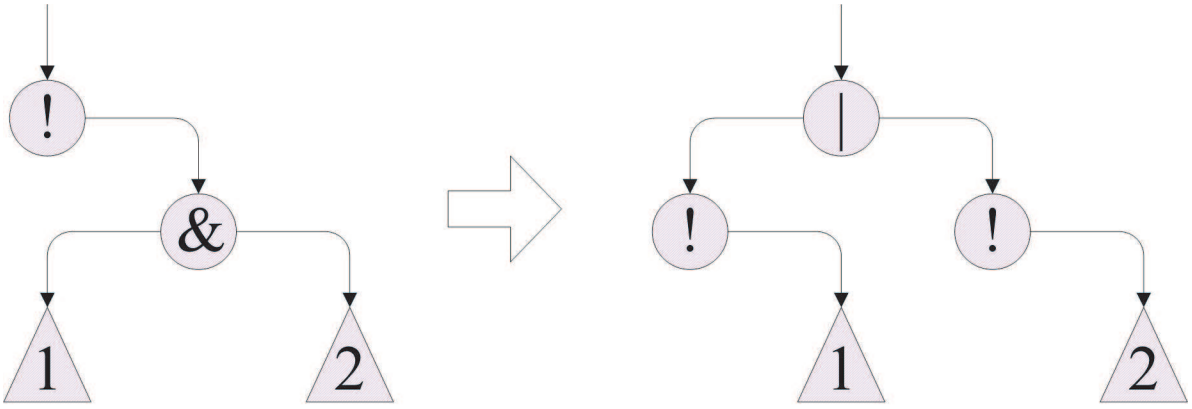

Рис. 5. Снятие отрицания для операции И

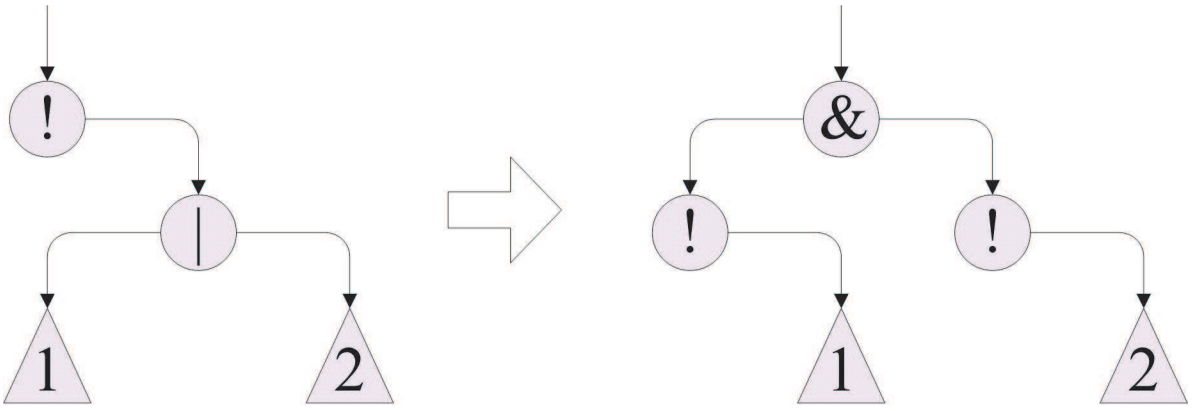

Рис. 6. Снятие отрицания для операции ИЛИ

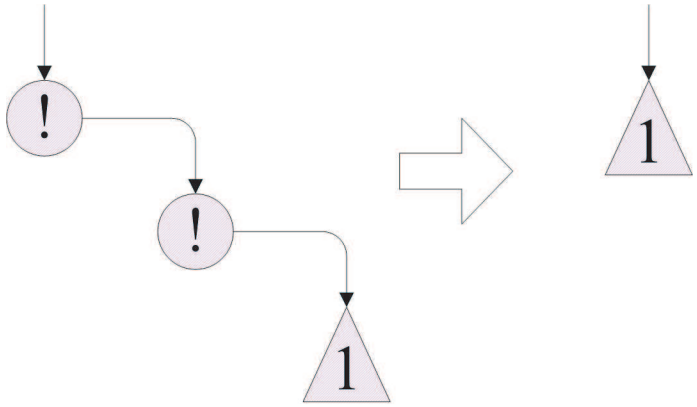

Рис. 7. Снятие отрицания для операции НЕ

Таким образом, как следует из рисунков, снятие отрицания при использовании древовидной структуры для хранения формул становится достаточно простой операцией. Такое снятие отрицания называется нормализацией. Теперь раскроем скобки (рис. 8).

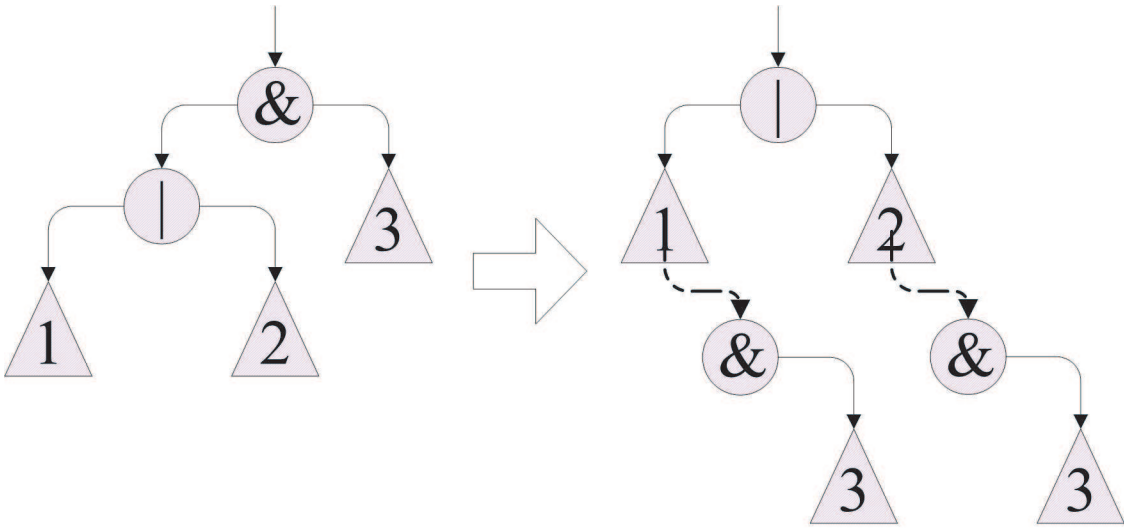

Рис. 8. Раскрытие скобок

Ортогонализация бесскобочного выражения. После раскрытия скобок ситуация резко упрощается. Ортогонализованная формула вида (1) вновь содержит скобки и выглядит следующим образом:

$$a_1\&a_2\&\ldots\&a_{n_1} \mid (!a_1 \mid a_1\&!a_2 \mid \ldots \mid a_1\&a_2\&\ldots\&a_{n_1-1}\&!a_{n_1})\&b_1\&b_2\&\ldots\&b_{n_2} \mid \ldots$$

Чтобы получить ОДНФ, остается вновь раскрыть скобки и убрать конъюнкции, тождественно равные нулю.

2.8. Построение линейного бинарного графа

Построим по полученной ортогональной дизъюнктивной нормальной форме линейный бинарный граф (ЛБГ). При этом для каждого дизъюнкта в формуле проведем сортировку переменных по их атрибутам. Далее построим остов графа, состоящий из условных вершин, помеченных буквами исходной формулы в порядке возрастания их атрибутов. При этом самыми последними выписываются операторные вершины “1” и “0” (TRUE и FALSE). Ребра соединяют вершины по следующему правилу: ребро соединяет две вершины тогда, когда существует дизъюнкт, в котором переменные, соответствующие этим вершинам, располагаются рядом. При этом ребро помечается единицей, если первая переменная в этом дизъюнкте объявлена без отрицания, и нулем в противном случае. Вершина соединяется ребром с вершиной TRUE, если существует дизъюнкт, в котором переменная, соответствующая этой вершине, стоит последней. При этом, если она объявлена с отрицанием, то ребро помечается нулем, иначе – единицей. Рассматриваемая вершина соединяется с вершиной FALSE в том случае, если из нее после всех описанных выше операций выходит только одно ребро. При этом ребро помечается единицей (нулем), если второе ребро помечено нулем (единицей).

Например, формула

$$a_1\&(a_2 \mid a_3) \tag{2}$$

после ортогонализации примет вид

$$a_1\&a_2 \mid a_1\&!a_2\&a_3. \tag{3}$$

Такой ОДНФ будет соответствовать следующий линейный бинарный граф (рис. 9).

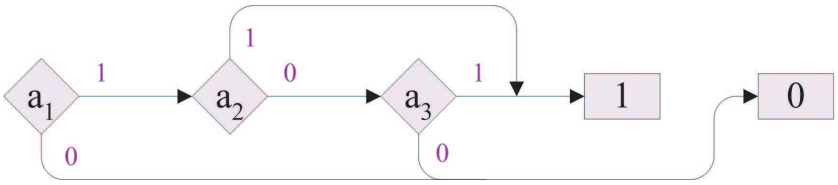

Рис. 9. Пример линейного бинарного графа

Заметим, что в линейном бинарном графе можно выделить так называемые единичные и нулевые пути. Единичным путем в линейном бинарном графе называется любой путь от самой левой (начальной) вершины до вершины, помеченной единицей (TRUE). Нулевым путем называется путь от начальной вершины до нулевой (FALSE). Особенность линейных бинарных графов состоит в том, что в соответствующей ему ортогональной ДНФ каждый из дизъюнктов всегда описывает один из единичных путей в графе. Например, в построенном графе единичный путь $a_1 \rightarrow a_2 \rightarrow 1$ соответствует первому дизъюнкту из ортогональной дизъюнктивной нормальной формы (3).

Отметим, что линейные бинарные графы имеют как практическую полезность (на их основе могут вычисляться булевы формулы без использования промежуточной памяти), так и теоретическую ценность (максимум числа путей в них определяется числами Фибоначчи) [1].

2.9. Построение каскада мультиплексоров

Построим каскад мультиплексоров по линейному бинарному графу, читая его в обратном порядке, справа налево, пропустив только вершины TRUE и FALSE. Каждой из вершин ЛБГ (кроме двух пропущенных) будет соответствовать мультиплексор “2 в 1”. Расположим эти мультиплексоры на прямой, в порядке убывания атрибутов соответствующих переменных (напомним, что атрибут переменной - это ее номер в словаре). На управляющий вход каждого мультиплексора подадим значение переменной. В общем случае на “нулевой” вход подается выход с мультиплексора, соответствующего вершине в линейном бинарном графе, соединенной с исходной вершиной нулевым ребром. На “единичный” вход подается выход с мультиплексора, соответствующего вершине в ЛБГ, соединенной с исходной вершиной ребром, помеченным единицей. Построим каскад так, чтобы на “единичный” вход всегда подавалось выходное значение с предыдущего мультиплексора. Для этого, возможно, придется инвертировать некоторые входные переменные.

На рис. 10 приведен мультиплексорный каскад, построенный для формулы (2).

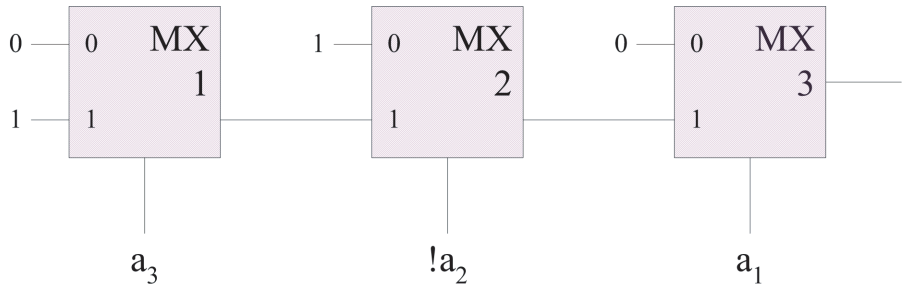

Рис. 10. Пример мультиплексорного каскада

3. Техническая документация

3.1. Описание входных переменных

$x101$ – Вершина стека – терминал.
$x102$ – Вершина стека – нетерминал.
$x103$ – Вершина стека – команда.
$x104$ – Вершина стека совпадает с текущей лексемой.
$x105$ – Вершина стека – стоп-символ и текущая лексема – стоп-символ.
$x106$ – В ячейке управлющей таблицы, соответствующей вершине стека и текущей лексеме стоит не ошибка.

$x200$ – Текущий символ – один из символов !, &, | , ;, (,).
$x201$ – Текущий символ – $.
$x202$ – Текущий символ может быть началом имени переменной.
$x203$ – Текущий символ может быть продолжением имени переменной.
$x204$ – Текущий символ – символ табуляции, перевода строки, возврата каретки или пробел.

3.2. Описание выходных воздействий

$z100$ – Инициализировать стек.
$z101$ – Вытолкнуть элемент из стека.
$z102$ – Выполнить команду, содержащуюся на вершине стековой машины, и вытолкнуть ее.
$z103$ – Поместить правило вывода в стек.

$z200$ – Считать следующий символ из потока.
$z201$ – Добавить символ в текущее имя переменной.
$z202$ – Добавить имя переменной в словарь.

3.3. Автомат синтаксического анализа A1_Syntax_recognizer

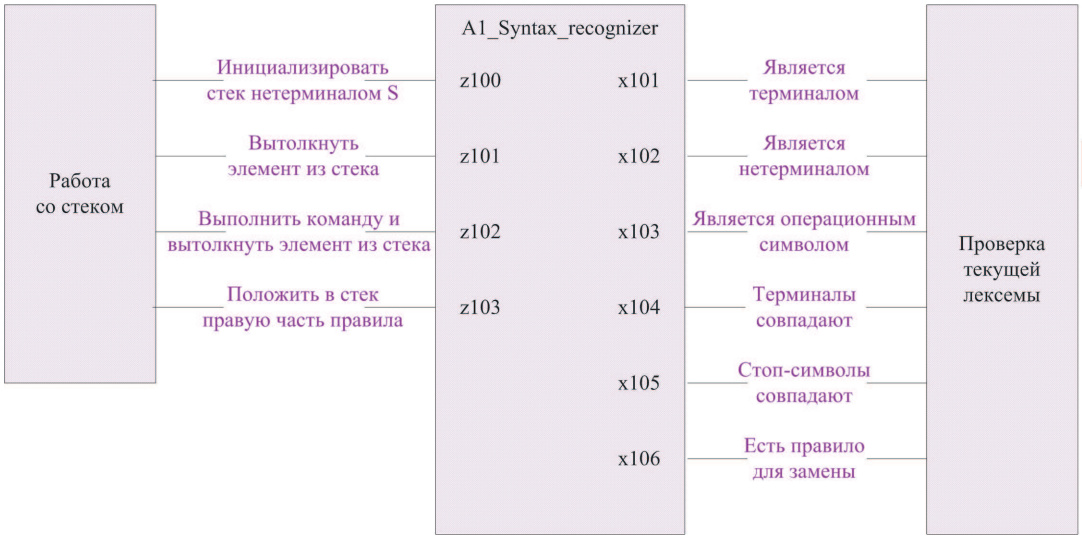

Рис. 11. Схема связей автомата A1_Syntax_recognizer

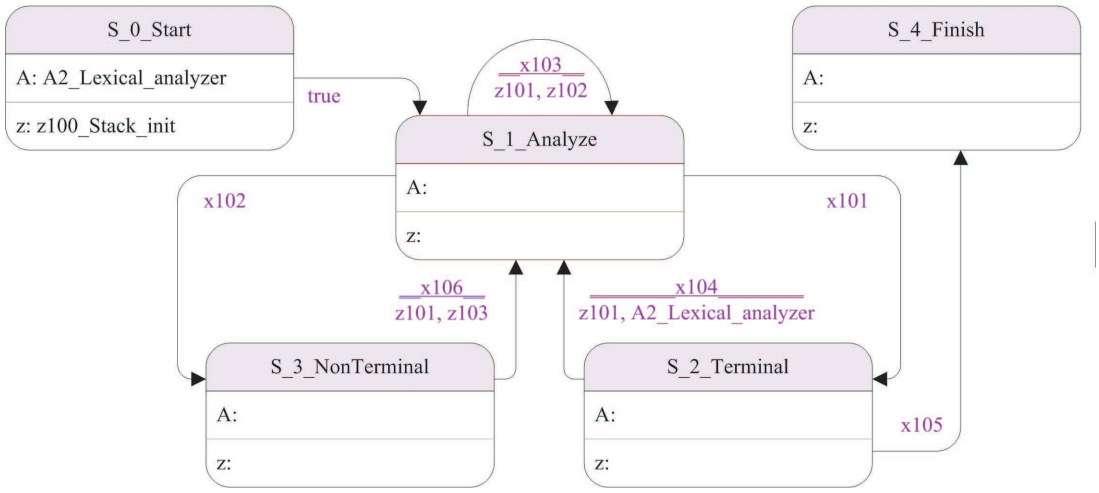

Рис. 12. Граф переходов автомата A1_Syntax_recognizer

3.4. Автомат лексического анализа A2_Lexical_analyzer

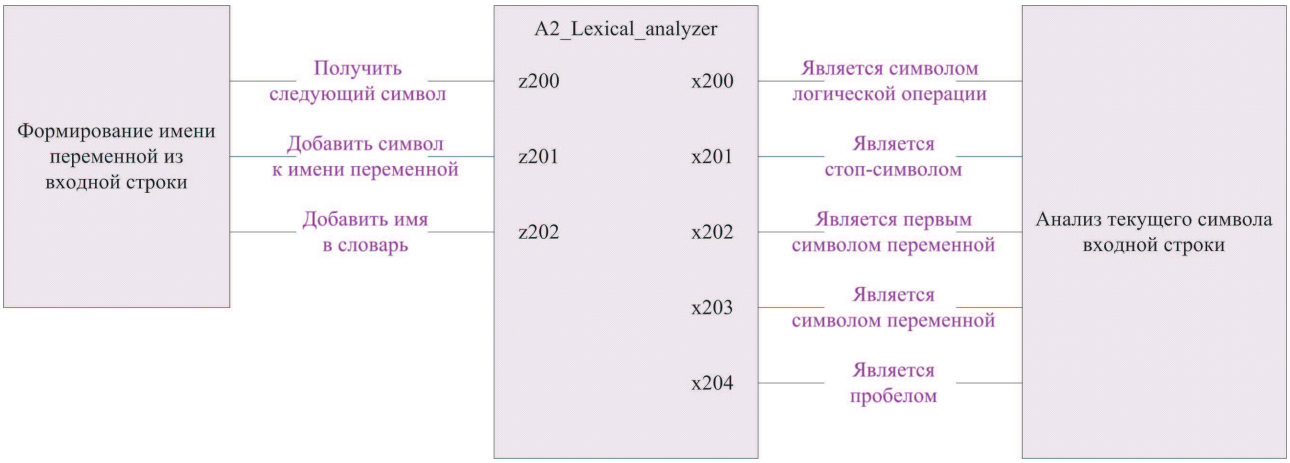

Рис. 13. Схема связей автомата A2_Lexical_analyzer

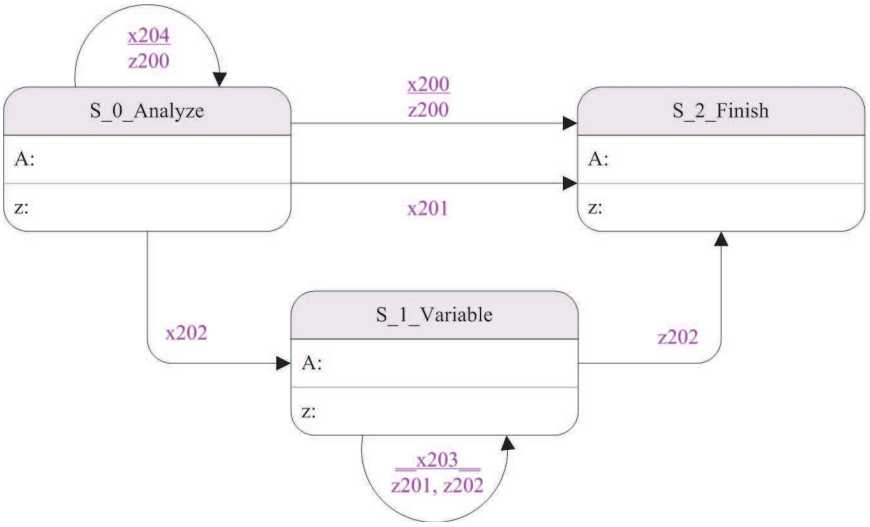

Рис. 14. Граф переходов автомата A2_Lexical_analyzer

4. Описание визуализатора

Опишем программу, визуализирующую изложенный в предыдущих разделах способ реализации БФ.

В левом верхнем окне (рис. 15) пользователь может ввести формулу. После нажатия кнопки “Add” введенная формула добавляется в список, отображаемый в окне, расположенном ниже. При этом формуле сопоставляется соответствующий ЛБГ (Graph) и

MX-каскад (MX-cascade), которые могут быть отображены в центральной области окна.

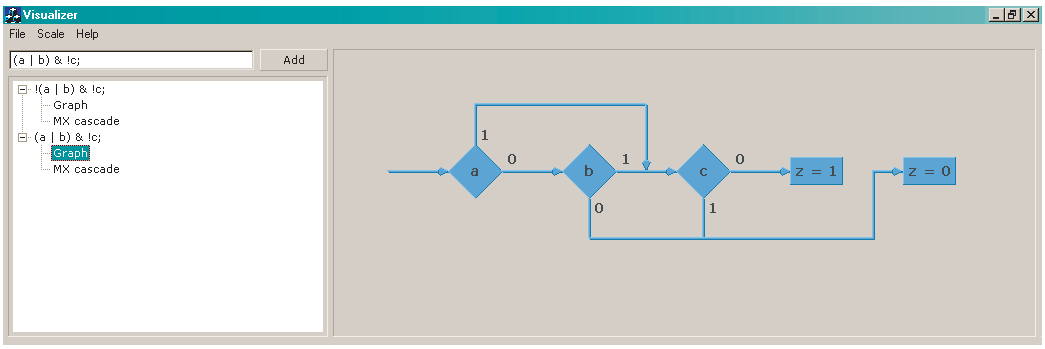

Рис. 15. Визуализатор. Линейный бинарный граф

Отображения линейного бинарного графа и MX-каскада различаются. Визуализация ЛБГ означает только изображение построенного графа на экране. Визуализация MX-каскада более сложна (рис. 16). Она предоставляет пользователю возможность пошагового исследования построенного каскада (один шаг на один мультиплексор в схеме).

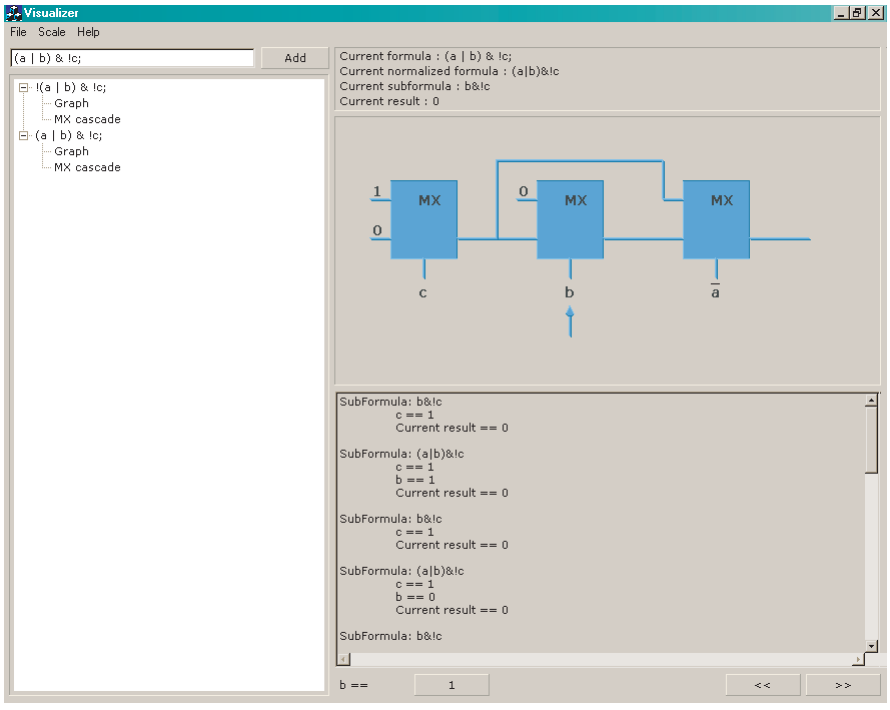

Рис. 16. Визуализатор. Мультиплексорный каскад

На каждом шаге визуализации MX-каскада:

- пользователь может ввести значение, поступающее на управляющий вход соответствующего текущему шагу мультиплексора (используя для этого кнопку внизу

экрана);

- в верхнем окне выводится значение на выходе текущего мультиплексора (Current result);
- в том же окне выводится фрагмент формулы, соответствующий текущему результату (Current subformula).

Переключение шагов производится с помощью кнопок в правом нижнем углу экрана.

Произведенные пользователем действия регистрируются в окне, расположенном под изображением МХ-каскада.

Опишем пример, приведенный на рис. 15, рис. 16. В данном случае пользователем были последовательно введены две формулы: $!(a \mid b)\,\&\,!c$ и $(a \mid b)\,\&\,!c$. Для последней на рис. 15 в центральном окне отображен ее линейный бинарный граф. Для этой же формулы на рис. 16 был рассмотрен МХ-каскад. Опишем этот рисунок более подробно.

Отметим, что введенная формула не изменилась после процесса нормализации, что отражено в верхнем окне.

Вертикальная стрелка под изображением МХ-каскада указывает, что программа находится на втором шаге визуализации, на котором пользователю дается возможность ввести значение второго с конца операнда формулы (b). Как мы видим внизу окна, на данный момент это значение равно единице. В окне регистрации действий пользователя можно увидеть, что значение первого с конца операнда формулы (c) также равно единице. В этом же окне, а также в окне сверху экрана, отображается текущий фрагмент формулы: $b\,\&\,!c$. В этих же окнах отражается соответствующий этому фрагменту результат, который равен нулю. Действительно, при введенных значениях

$$\begin{cases} c == 1 \\ b == 1 \end{cases}$$

результат вычисления формулы $b\,\&\,!c$ будет равен нулю.

Этот результат и будет получен на выходе со второго мультиплексора в схеме. Заметим, что на выходе с первого мультиплексора будет получено значение элемента c, а на выходе с третьего мультиплексора – результат вычисления всей введенной пользователем формулы. Таким образом, в схеме последовательно вычисляются следующие фрагменты: c, $b\,\&\,!c$, $(a \mid b)\,\&\,!c$, по этим же фрагментам вычисляется и формула.

Описанную программу можно скачать с сайта `http://is.ifmo.ru`, раздел “Проекты”.

В заключение отметим, в настоящей работе впервые осуществлена программная реализация метода вычисления БФ по фрагментам, который был предложен в работе [8].

Литература

[1] *Шалыто А.А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000. `http://is.ifmo.ru/books/log_upr/2`.

[2] *Таненбаум Э.* Архитектура компьютера. СПб.: Питер, 2003.

[3] *Ахо А., Сети Р., Ульман Д.* Компиляторы. Принципы, технологии, инструменты. М.: Вильямс, 2001.

[4] *Штучкин А.А., Шалыто А.А.* Совместное использование теории компиляторов и SWITCH-технологии (на примере проектирования калькулятора). 2003. `http://is.ifmo.ru` (раздел "Проекты").

[5] *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

[6] *Шалыто А.А., Туккель Н.И.* От тьюрингова программирования к автоматному. Мир ПК, 2002, №2. `http://is.ifmo.ru` (раздел "Проекты").

[7] *Кнут Д.Э.* Искусство программирования. Том 1. Основные алгоритмы. М.: Вильямс, 2000.

[8] *Артюхов В. Л., Шалыто А.А.* Судовые управляющие логические системы. Л.: Ин-т повышения квалификации руководящих работников и специалистов судостроительной промышленности, 1984.