ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 8
N 7
2017

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.052.42, 004.4'233 DOI: 10.17587/prin.8.291-299

Д. А. Орлов, канд. техн. наук, доц., e-mail: orlovdmal@mpei.ru,
Национальный исследовательский университет "МЭИ", г. Москва

Статический анализ исходных текстов программ методом обратного выполнения на наличие в них ошибок доступа к памяти

Рассмотрена возможность применения метода обратного выполнения для статического анализа исходных текстов программ, реализующих алгоритмы обработки строк. Предложен алгоритм выявления ошибок доступа к памяти в исходных текстах таких программ. Представлены результаты работы программной реализации предложенного алгоритма.

Ключевые слова: *статический анализ исходных текстов программ, ошибки доступа к памяти, программирование, теория алгоритмов, тестирование программ, верификация программ, обратное выполнение*

Введение

В связи с ростом сложности программ задача тестирования программного обеспечения становится более актуальной. Помимо тестирования, в частности, проведения модульных (*unit*) и объемных (*volume*) тестов, большое значение приобретает как статический, так и динамический анализ программ. Динамический анализ состоит в исследовании поведения программы во время выполнения. Статический анализ состоит в исследовании кода программы без ее запуска. Применение данных методов позволяет обнаружить большее число ошибок в исходном коде программ.

Преимуществом статического анализа исходных текстов программ является отсутствие необходимости готовить входные данные для выполняемой программы. Кроме того, некоторые из алгоритмов статического анализа позволяют получить доказательство выполнения части ограничений для всех возможных наборов входных данных программы.

Один из наиболее распространенных классов ошибок в исходных текстах программ — ошибки доступа к памяти, прежде всего, доступ к памяти за границами массивов. В этих случаях происходит считывание или запись элемента, находящегося за границей массива. В случае чтения это приводит к получению неверного значения, в случае записи — к повреждению структур данных, расположенных в памяти рядом с массивом. В обоих случаях такие ошибки могут привести к аварийному завершению программы. Отметим, что выявление ошибок подобного класса — достаточно сложная задача. Во-первых, ошибки рассматриваемого класса не выявляются при синтаксическом анализе исходного текста программ. Во-вторых, компилятор может обнаружить только простейшие случаи возникновения подобных ошибок и сообщить о них в виде предупреждения.

Данная статья посвящена разработке алгоритма статического анализа исходных текстов программ на предмет наличия ошибок доступа к памяти за границами массивов. Предлагаемый в настоящей статье алгоритм использует идею обратного выполнения (*backwards execution*) для проверки того, можно ли в данное ошибочное состояние программы прийти из ее точки входа. На начальной стадии проверки предполагается, что программа уже находится в ошибочном состоянии. Далее, алгоритм определяет множество состояний, из которых можно перейти в ошибочное, затем — множество состояний, предшествующих найденным и т. д., до тех пор, пока либо не будет достигнуто начальное состояние, либо множество состояний не окажется пустым. Если ограничиться исследованием ошибок доступа к памяти, то сразу будут ясны возможные ошибочные состояния проверяемой программы — это все точки, где осуществляется доступ к элементам массива (или разыменование указателей).

1. Ошибки доступа к памяти

Ошибки доступа к памяти составляют один из наиболее распространенных классов ошибок при программировании. Примерами таких ошибок является доступ к памяти за границами динамических структур данных (прежде всего, массивов), двойное освобождение памяти, чтение из неинициализированной области памяти. Подобные ошибки могут вызвать аварийное завершение работы программы. Стандарты языков программирования явно указывают, что в таких случаях поведение программы не определено [1].

```
int test_array[10]={0,1,2,3,4,5,6,7,8,9};
int s=0;
for(int i=0; i<=10; ++i){
    s+=test_array[i];
}
```

Рис. 1. Фрагмент исходного кода, содержащий ошибку чтения за границей массива

Ситуация доступа за границу массива может возникнуть, например, в случае ошибки вычисления индекса элемента массива. Фрагмент исходного кода на языке C + +, демонстрирующий чтение за границей массива, приведен на рис. 1.

В приведенном примере индекс элемента массива изменяется от 0 до 10 включительно, при этом индекс элемента массива `test _ array` должен находиться в пределах от 0 до 9 включительно. Таким образом, при обращении к `test _ array[10]` происходит чтение из памяти за границей `test _ array`. Поведение программы в такой ситуации не определено. Возможно как чтение данных за границей массива, при котором значение полученного выражения будет произвольным и, как следствие, результат вычислений будет испорчен, так и срабатывание защиты памяти, когда программа будет аварийно завершена.

Представленный на рис. 1 фрагмент кода синтаксически корректен, поэтому для выявления ошибок подобного класса необходим дополнительный анализ. Синтаксические анализаторы и компиляторы способны выявить лишь простейшие случаи таких ошибок.

В исследовании, результаты которого представлены далее, рассмотрены ошибки доступа за границы массива, размер которого не изменяется в пределах функции.

2. Методы выявления ошибок

Методами выявления ошибок в программах являются тестирование и анализ кода программы. Тестирование не может покрыть всех возможных случаев ошибок, поэтому его желательно использовать совместно с анализом кода программы. В свою очередь, методы анализа кода программы можно разделить на два класса: статический анализ и динамический анализ.

Динамический анализ состоит в исследовании поведения программы во время выполнения. Например, во время выполнения программы может проводиться контроль значений переменных. Широко применяемым инструментарием динамического анализа программ является Valgrind [2]. Программные механизмы Valgrind позволяют выполнять машинный код исполняемого файла в специальной виртуальной машине. Они позволяют обнаружить ошибки доступа к памяти за границами массивов, случаи использования неинициализированных переменных и т. д. Программный комплекс Valgrind имеет модульную структуру и открытый исходный код, что позволяет разработчикам писать расширения для него.

Статический анализ состоит в исследовании исходного текста программы без ее запуска. Это позволяет определить некоторые свойства программы без подбора соответствующих входных данных. Кроме того, для проверки некоторых свойств как при динамическом анализе, так и при тестировании программы может потребоваться ее запуск для всех возможных наборов входных данных, что для реальных приложений практически невозможно.

Среди методов статического анализа исходных текстов программ можно выделить перечисленные далее.

- **Эвристические подходы** реализуют наиболее распространенные утилиты статического анализа, например, cppcheck [3]. Эти подходы направлены на выявление наиболее распространенных шаблонов ошибок, например отсутствие закрытия файла, или двойное освобождение памяти. Данные подходы используют наиболее простые средства, а именно анализ синтаксического дерева и регулярные выражения.
- **Метод формальной верификации** состоит в построении доказательства соответствия программы ее формальной спецификации, задаваемой на специальном языке. В качестве примера языка спецификаций можно привести язык ACSL [4]. Сложность применения методов формальной верификации связана с тем, что задача автоматического доказательства теорем алгоритмически неразрешима. Следовательно, в общем случае процесс верификации может быть лишь автоматизированным, т. е. с обязательным участием человека. Таким образом, формальная верификация даже простых участков кода превращается в сложную задачу [5]. Поэтому формальная верификация программ применяется только в критически важном программном обеспечении, например, в программном обеспечении бортовых систем самолетов.
- **Подход на основе абстрактной интерпретации** программы состоит в отображении конкретных состояний программы в пространство абстрактных состояний. Размерность пространства абстрактных состояний значительно меньше размерности пространства конкретных состояний, что позволяет выполнить программу для всех возможных отображений входных данных [6]. Примером подобного отображения может служить отображение множества целых чисел на множество из двух значений: четное и нечетное. В этом случае мощность множества входных данных значительно сокращается. При этом в указанном пространстве абстрактных состояний сохраняется возможность анализа четности суммы и произведения.
- **Методы, основанные на анализе распространения данных**, состоят в построении графа связей переменных по данным и в дальнейшем анализе этого графа.
- **Методы**, **основанные на анализе путей выполнения,** реализуют анализ всех возможных путей выполнения программы и проводят проверку того,

существует ли путь, который приводит программу в ошибочное состояние. Технически это может быть реализовано построением графа состояний, достижимых из начального состояния программы [7]. Вершинами этого графа являются состояния, дуги графа соответствуют возможностям непосредственного перехода программы из одного состояния в другое. При каждом условном переходе в граф добавляются две дуги и два состояния. Подробнее анализ путей выполнения программы будет рассмотрен в следующем разделе.

- **Метод обратного выполнения** состоит в определении состояний, из которых достижимо заданное состояние программы. На каждой итерации определяется множество состояний, предшествующих текущему состоянию программы. Таким образом, каждая команда анализируемой программы "выполняется назад". Сложность реализации алгоритмов, основанных на методе обратного выполнения, состоит в том, что большинство команд необратимы. Поэтому одному состоянию может предшествовать множество других состояний [8]. Чаще всего обратное выполнение применяют при анализе причин аварийного завершения программ.

При разработке программного обеспечения все шире применяют статические анализаторы кода, реализующие некоторые из представленных выше подходов. Прежде всего, это анализ путей выполнения, например, в clang-analyzer [7], и эвристические подходы. Среди применяющихся программных средств наиболее распространены clang-analyzer, cppcheck, pvs-studio и coverity. В России также проводят исследования в области статического анализа и верификации программного обеспечения. Среди таких проектов отметим проект по верификации ядра Linux [9], а также разработку статического анализатора SVACE [10]. Оба проекта выполняются в Институте системного программирования РАН.

3. Анализ путей выполнения программы и символьное выполнение

Путем выполнения программы назовем последовательность команд программы (или инструкций языка высокого уровня, в зависимости от того, анализируется двоичный код или исходный код программы), которая должна быть выполнена при запуске программы с некоторыми входными данными. Основной проблемой, возникающей при применении анализа путей выполнения, является комбинаторный взрыв количества путей (в англоязычной литературе подобную ситуацию называют *path explosion*). Например, участок кода, имеющий n последовательных ветвлений, может иметь до 2^n путей выполнения. Для программ, содержащих циклы с предусловием, число путей выполнения потенциально бесконечно [11]. Поэтому при анализе путей выполнения применяют различные методы сокращения числа исследуемых путей.

Как правило, при анализе путей выполнения применяется символьное выполнение программы [12]. Под символьным выполнением понимается метод исследования программы, при котором конкретные входные данные заменяют на символьные выражения. В результате выходными данными программы будут не конкретные значения, а функции от входных символьных выражений. Результатом каждой операции является преобразование имеющихся символьных переменных или введение новых (если операция состоит в получении входных данных). Выполнение каждого ветвления накладывает ограничения на переменные. Таким образом, символьным состоянием программы можно назвать имеющиеся значения символьных переменных, а также совокупность имеющихся ограничений. Отметим, что в отличие от обычного ("конкретного") выполнения, в случае, если порожденные ограничения совместны, необходимо выполнить обе ветки ветвления. Таким образом, результатом символьного выполнения будут пути выполнения программы, причем для каждого из путей будет дан результат в виде символьных выражений, зависящих от входных данных, а также набор ограничений на входные данные, при которых реализуется именно данный путь выполнения программы. Если для всех имеющихся наборов ограничений построить набор входных данных, который удовлетворяет данным ограничениям, то можно получить набор тестов, при выполнении которых реализуется каждый из путей выполнения исследуемой программы [11]. Основным из ограничений для применения символьного выполнения является комбинаторный взрыв количества анализируемых путей. Для сокращения количества анализируемых путей применяют различные способы, например, использование резюме функций [13] или отказ от межпроцедурного анализа [10].

Символьное выполнение может происходить как в прямом, так и в обратном направлениях. В существующих программных средствах статического анализа программ используется прямое символьное выполнение. Метод обратного выполнения чаще всего применяют при анализе причин аварийного завершения программы. В этом случае разработчик программы, как правило, располагает снимком содержимого памяти исследуемой программы (*coredump*), сделанным в момент аварийного завершения программы. Метод обратного выполнения позволяет восстановить условия, приведшие к аварийному завершению программы.

В обоих случаях при символьном выполнении возможны бесконечные циклы, например, при анализе циклов while [11]. Отметим, что в обратном направлении анализ некоторых синтаксических конструкций может оказаться более удобен. Например, символьное выполнение цикла while позволит анализировать состояние программы сразу после выхода из этого цикла, поэтому упрощается анализ нарушения условий, которые должны выполняться после выхода из цикла. Таким образом, при статическом анализе программ предпочтительным может оказаться одновременное использование как прямого, так и обратного выполнения.

4. Алгоритм обнаружения ошибок доступа к памяти

В центре внимания данной статьи находится разработка алгоритма обнаружения ошибок доступа к памяти, основанного на методе обратного выполнения программы.

4.1. Предлагаемый подход

Предлагаемый алгоритм будет анализировать исходный текст программы, выдавая в качестве результата заключение о возможности или невозможности возникновения ошибок доступа к памяти в данной программе. Кроме того, каждое сообщение о возможности ошибки будет сопровождаться списком ограничений на входные данные программы, при выполнении которых возможна ошибка доступа к памяти, что облегчит разработчику отладку программы и написание тестов.

Перечислим существующие подходы к обратному выполнению программ:

- генерация ограничений при обратном выполнении команд, использование решателей для проверки противоречивости сгенерированных ограничений;
- создание обратимого кода, например использование специальных языков программирования, команды которых обратимы; в качестве примера такого языка можно привести язык Janus [8].

Предлагаемый подход состоит в анализе методом обратного выполнения только программ определенного вида. Практически все операции таких программ будут обратимы, а сгенерированные ограничения будут иметь простой вид.

Пусть есть алгоритм, заданный в виде функции на императивном языке высокого уровня. Потребуем, чтобы все операции, изменяющие переменные (за исключением присваивания), были обратимыми. Более того, ограничим такие операции сложением с константами. Такие требования сужают множество исходных текстов программ, которые могут быть исследованы. Однако множество исследуемых исходных текстов остается достаточно обширным. Среди исходных текстов, обладающих такими свойствами, прежде всего можно выделить исходные тексты функций обработки строковых данных и разбора текстовых форматов. Например, такими свойствами обладают программные реализации алгоритмов построения синтаксического дерева по текстовой строке.

Для удобства рассуждений будем считать, что исследуемая функция записана на языке C. Будем также считать, что исследуемая функция не изменяет массивы, с которыми она работает, а также не выделяет и не освобождает память. Будем также считать, что функция принимает на вход массив, заданный двумя указателями: на первый элемент массива (обозначим как b) и на элемент, следующий за последним (обозначим как e). Ошибкой будет считаться чтение из данного массива по указателю p, при условии, что $p < b$ или $p >= e$. Таким образом, ошибки доступа к памяти возможны только в тех участках программы, где происходит разыменование указателей.

Для реализации обратного выполнения программы ее исходный текст необходимо преобразовать в пронумерованную последовательность команд, и для каждой из команд определить способ ее обратного выполнения. В текущей реализации используется модельная система команд, которая включает в себя циклы (с предусловием, с постусловием), условные переходы, возвраты, арифметические операции. Из арифметических операций поддерживаются присваивание, сравнение и сложение с константой. Поддерживающиеся виды условий также ограничены сравнением с константами (для переменных, представляющих числовые значения) и сравнением с другим указателем (для указателей).

Введем следующие определения. *Программой* назовем пронумерованную последовательность команд. Команды пронумерованы в порядке следования соответствующих им конструкций исходного текста на языке C. Пусть i — номер текущей команды, тогда $Prev(i)$ — *множество номеров предшествующих команд*, т. е. команд, из которых возможен переход в рассматриваемую команду.

Назовем *состоянием программы* следующую конструкцию: $S = <i, R>$, где i — номер текущей команды программы, $0 \leqslant i < N$, где N — число команд; R — ограничения, накладываемые на значения переменных и указателей в программе.

Начальным состоянием программы назовем состояние вида $S_0 = <0, R>$, т. е. состояние, соответствующее начальной команде и ограничениям, накопленным в ходе обратного выполнения.

Ошибочным состоянием программы назовем состояние программы в момент возникновения ошибки доступа к памяти.

При обратном выполнении каждой команды в состояние программы могут быть добавлены новые ограничения. Если система условий, соответствующая данному состоянию программы, несовместна, то такое состояние назовем *невозможным*. При обнаружении невозможного состояния следует остановить обратное выполнение. Таким образом, важной подзадачей, подлежащей решению в процессе реализации обратного выполнения программы, является задача определения совместности полученной системы ограничений. Задача определения совместности систем ограничений подобного вида называется задачей определения выполнимости в теориях, или SMT-задачей. Для решения задач подобного вида применяют специальный класс программ — SMT-решатели [14]. Чтобы решение задачи не занимало много времени, вид условий упрощают, например, ограничиваясь линейными функциями. Поэтому для упрощения системы условий, совместность которой необходимо проверять, целесообразно ввести дополнительные ограничения на класс исследуемых программ.

4.2. Формат ограничений и модельная система команд

Рассмотрим подробно вид ограничений, которые становятся известными в ходе обратного выполнения программы. В текущей реализации используется разработанный автором алгоритм проверки совместности системы ограничений, который допускает использование ограничений лишь определенного вида, что позволяет упростить процедуру проверки их совместности. Расширить возможный вид ограничений позволит применение сторонних SMT-решателей.

В текущей реализации алгоритма анализа исходных текстов программ ограничения состояния программы будут иметь следующий вид. Рассматриваются ограничения только на целые типы данных. Ограничения могут быть наложены как на значение указателя, так и на значение переменной. В текущей реализации ограничения выражаются через множество значений переменной, которое представляется интервалами.

В текущей реализации ограничение на переменную состоит из адреса переменной и множества значений, которое может принимать переменная, доступная по этому адресу. Адреса переменных, а также значения указателей задают как сумму базы и смещения. Смещение задается целым числом. База задается одной из следующих именованных констант:

B — адрес начала массива (значение указателя *b*, подаваемого на вход исследуемой функции);

E — адрес элемента, следующего за концом массива (значение указателя *e*, подаваемого на вход исследуемой функции);

Z — нулевой указатель;

*X*0 — адрес, по которому произошел ошибочный доступ к памяти.

Кроме того, при обнаружении указателя, не известного ранее, ему присваивается значение вида *Xi*, где *i* — номер, который будет присвоен указателю, целое положительное число.

При выявлении ошибок доступа к памяти за границей массива важна только разность адреса начала массива и адреса, по которому осуществляется доступ к памяти, а не значения этих адресов (т. е. анализируется значение *Xi*-*B*, при этом сами значения *Xi* и *B* не важны). Поэтому имеет смысл запоминать ограничения не на базы указателей, а на их разности. Очевидно, что если база указателя — это некоторый адрес в памяти, то разность баз указателей будет целым числом со знаком. Таким образом, ограничение на указатель будет состоять из разности баз и множества значений, которое может принимать разность баз.

При проверке совместности системы условий в данной реализации также не проводится решение задачи псевдонимов, т. е. не проводится отождествление адресов, вычисленных исходя из разных выражений [10], что также упрощает анализ. Изложенные выше ограничения связаны с используемым решателем. При замене данного решателя ограничения могут быть сняты.

Рассмотрим модельную систему команд, в последовательность которых должен быть преобразован исходный код исследуемой функции. Модельная система команд включает в себя перечисленные далее команды.

- `Statement expression` — выполнить выражение expression.
- `Nop` — пустая операция, никаких действий не проводится.
- `Block index` и `Endblock index` — задают границы блоков команд, блоки могут вкладываться друг в друга. Данные команды примерно соответствуют фигурным скобкам языка С, однако в модельной системе команд ветвления и циклы требуют обязательного наличия блока команд, ограниченного командами `Block` и `Endblock`. Для удобства анализа программы, `Block` в качестве операнда `index` содержит индекс соответствующего `Endblock`.
- `If expression` — выполнить следующий блок только в случае, если выражение expression истинно. После данной команды в программе обязательно должна присутствовать последовательность команд, ограниченная командами `Block` и `Endblock`. После этой последовательности возможно задание команды `Else`.
- `Elsce` — выполнить следующий блок только в случае, если выражение, соответствующее команде, `If` ложно. После данной команды в программе обязательно должна присутствовать последовательность команд, ограниченная командами `Block` и `Endblock`.
- Следующие команды соответствуют аналогичным операторам языка С: `While` задает цикл с предусловием, `Dowhile` задает цикл с постусловием, `Break` обеспечивает выход из цикла, `Continue` — продолжение выполнения, `Return` — выход из функции.

Рассмотрим формат выражения, использующегося в командах `Statement`, `If`, `While`, `Dowhile`. Выражение может состоять из одной из следующих операций.

◇ Операция `Assign` — присваивание, когда переменной присваивается значение заданного выражения. В текущей реализации указателю может быть присвоено только значение другого указателя, а переменной может быть присвоено значение только константы. Это сделано для упрощения вида получаемых ограничений.

◇ Операция `AddAssign` — составное присваивание, которое реализует операцию сложения с целой константой. Операндом может быть как указатель, так и переменная.

◇ Операция `Compare` — сравнение двух операндов.

4.3. Проверка достижимости начального состояния

Пусть при выполнении некоторой команды исследуемой программы произошла ошибка доступа к памяти. Тогда ошибочное состояние программы будет включать в себя номер текущей команды и условия чтения за границей массива. Задачу проверки возможности ошибочного доступа к памяти в некоторой команде программы

можно переформулировать следующим образом: достижимо ли данное ошибочное состояние из начального состояния программы. Проверку достижимости можно провести, используя обратное выполнение программы, начиная с данного ошибочного состояния.

При проведении проверки достижимости следует ввести еще одно допущение. Будем считать, что ошибка доступа к памяти произошла в данной программе впервые. Это связано с тем обстоятельством, что доступ к памяти за границами массива приводит к неопределенному поведению программы [1]. Следовательно, не имеет смысла исследовать выполнение программы после возникновения ошибки доступа к памяти.

Проведя проверку достижимости начального состояния программы из всех возможных ошибочных состояний программы (а именно состояний, соответствующих командам доступа к памяти массива, при условии доступа за его границами), можно узнать, возможны ли в данной программе ошибки доступа к памяти. Проверка достижимости состоит в обратном выполнении команд программы до тех пор, пока не будет достигнуто начальное состояние программы, или не будут исчерпаны все варианты обхода.

При обратном выполнении команды с номером *i* следующей будет рассматриваться одна из команд множества *Prev*(*i*), т. е. команда, из которой возможен переход в команду с номером *i*. Алгоритм, реализующий проверку, должен обойти все возможные пути, поэтому будет выбрана одна из возможных предыдущих команд. Состояния, соответствующие остальным командам из *Prev*(*i*), будут сохранены в стеке состояний. Состояние будет извлекаться из стека в том случае, если дальнейшее обратное выполнение для текущего состояния будет невозможно. Обратное выполнение будет проводиться до тех пор, пока система условий, содержащаяся в состоянии программы, совместна, либо до достижения начального состояния программы.

Определим способы обратного выполнения синтаксических конструкций языка C++. Пусть текущая команда имеет номер *i*.

Следование. В этом случае программа должна быть переведена в состояние, из которого данная команда позволяет получить текущее состояние. При обратном выполнении команд, содержащих обращение к памяти по указателю, в систему ограничений состояния программы необходимо добавить условия корректности этих обращений. Такие действия необходимы, поскольку предполагается, что до вхождения программы в ошибочное состояние других ошибок обращения к памяти не было.

Ветвления. Рассмотрим фрагмент кода, представленный на рис. 2.

На рис. 2 `prev _ instruction` и `post _ instruction` — некоторые операторы языка C, расположенные непосредственно перед и после `if`; `if _ instruction _ block` и `else _ instruction _ block` — операторы, находящиеся в ветвях, соответствующих истинному и ложному значению условия (*condition*) соответственно.

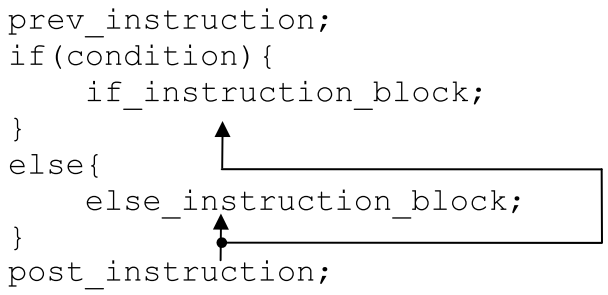

Рис. 2. Обратное выполнение ветвления

Выполнению `post _ instruction` должно предшествовать либо выполнение блока `if`, либо выполнение блока `else`. На рис. 2 эти направления показаны стрелками. Алгоритм анализа должен перейти по одной из стрелок, а состояние, соответствующее другой, — сохранить в стеке состояний программы.

По выходу из блока if алгоритм должен добавить новое ограничение — `condition == true`, поскольку только в этом случае выполнился бы блок `if`. Аналогично, по выходу из блока else алгоритм должен добавить новое ограничение — `condition == false`.

Цикл с предусловием. Рассмотрим фрагмент кода, представленный на рис. 3.

Поскольку `condition` — это условие входа в цикл, то выполнению `post _ instruction` должен предшествовать выход из цикла `while` при условии ложного значения `condition` (на рис. 3 это показано стрелкой справа). Проверке условия цикла может предшествовать либо выполнение `prev _ instruction`, либо выполнение тела цикла `while`. При перемещении от `post _ instruction` к `while` алгоритм анализа должен добавить условие `condition == false` к имеющимся ограничениям, так как произошел выход из цикла. При перемещении от `while` к `prev _ instruction` алгоритм анализа должен помещать в стек состояние программы, соответствующее положению, сразу после `while _ block`. К ограничениям состояния, помещаемого в стек, должно быть добавлено условие `condition == true`.

Цикл с постусловием. Рассмотрим фрагмент кода, представленный на рис. 4.

При переходе от `post _ instruction` к `while _ block` к уже имеющимся ограничениям должно быть добавлено условие `condition == false`, поскольку только при этом условии цикл может быть завершен. При движении от `while _ block` к `prev _ instruction` в стек состояний должно быть поме-

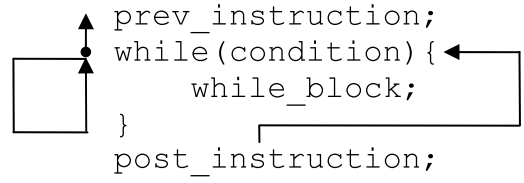

Рис. 3. Обратное выполнение цикла с предусловием

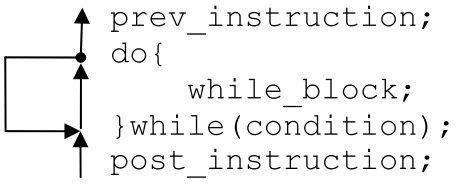

Рис. 4. Обратное выполнение цикла с постусловием

щено состояние, соответствующее положению, сразу после `while _ block`. К ограничениям состояния, помещаемого в стек, должно быть добавлено условие `condition == true`.

Для предотвращения зацикливания алгоритма анализа следует предусмотреть возможность его досрочного завершения. В качестве критерия останова предлагается использовать количество сохранений состояний, полученных при обходе циклов.

5. Опытная реализация и эксперименты

В ходе исследований, результаты которых представлены в настоящей статье, разработана опытная программная реализация алгоритма статического анализа исходных текстов на наличие ошибок доступа к памяти за границами массивов. Для разработки этой программы использован язык C++. Результатом работы программной реализации алгоритма являются:

- либо сообщение о том, что ошибки доступа к памяти при выполнении исследуемой программы не обнаружены;
- либо сообщение об обнаружении ошибки доступа к памяти, список ограничений на входные данные, при этом любые наборы входных данных, удовлетворяющие указанному списку ограничений, приводят к ошибке доступа к памяти при выполнении исследуемой программы;
- либо сообщение о том, что анализ прерван, чтобы предотвратить зацикливание программы.

При анализе исходного текста программы считается, что на вход переданы корректные значения указателей, задающие начало и конец массива. Как следствие, перед началом анализа программы в состояние программы добавляется ограничение $B <= E$.

В качестве тестовых были выбраны исходные тексты программ, реализующих три функции. Первая функция реализует алгоритм определения длины С-строки (строки, заканчивающейся нулевым байтом), не учитывающий границу массива. Очевидно, что такая программа может породить ошибку чтения за границей массива, если до его конца (задан указателем `e`) не встретится ни одного нулевого байта. Исходный текст данной функции представлен на рис. 5.

Рассматриваемый исходный текст содержит единственную инструкцию, где происходит доступ к памяти по указателю — проверка условия в цикле `while`. Результат проверки исходного текста данной функции, а именно вывод разработанной программной реализации, представлен на рис. 6.

```
int mystrlen(const char *b, const char *e){
    int i=0;
    const char *p = b;
    while (*p) { // dereference
                 // here is the possible error point
        ++p;
        ++i;
    }
    return i;
}
```

Рис. 5. Исходный текст тестовой функции определения длины строки

```
Executing test mystrlen
Error can be obtained
Pointer conditions are:
    0<=E-B
    X0-B==0
    0<=X0-E
Variable conditions are:
```

Рис. 6. Результат проверки исходного текста тестовой функции определения длины строки

Вывод программной реализации алгоритма анализа исходных текстов содержит все ограничения, накопленные в ходе обратного выполнения анализируемой программы. Ограничения разделены на две группы: ограничения на указатели (`pointer conditions`) и ограничения на переменные (`variable conditions`). Ошибка доступа к памяти будет воспроизведена, если входные данные анализируемой программы будут соответствовать всем выведенным ограничениям. Таким образом, для некоторых наборов входных данных при выполнении функции, исходный код которой представлен на рис. 5, будет возникать ошибка доступа к памяти. Условиям, представленным на рис. 6, соответствует массив нулевой длины.

Вторая тестовая функция реализует алгоритм определения длины С-строки, проверяющий границу массива. Выполнение такой программы не должно приводить к ошибкам доступа к памяти, что подтверждается результатом проверки. Исходный текст данной функции представлен на рис. 7.

```
int mystrlen(const char *b, const char *e){
    int i=0;
    const char p = b;
    while (p<e){
        if(!*p){ // dereference
            break;
        }
        ++p;
        ++i;
    }
    return i;
}
```

Рис. 7. Исходный текст тестовой функции определения длины строки с проверкой границы массива

```
Executing test mystrlen_border_check
Error wasn't found
```

Рис. 8. Результат проверки исходного текста тестовой функции определения длины строки с проверкой границы массива

Рассматриваемый исходный текст содержит единственную инструкцию, где происходит доступ к памяти по указателю — проверка условия в операторе `if`. Результат проверки исходного текста данной функции представлен на рис. 8. Программная реализация алгоритма анализа не обнаружила ошибок доступа к памяти. Действительно, поскольку выход указателя `p` за границу массива проверяется перед выполнением следующей итерации цикла, а увеличение указателя `p` происходит в конце итерации, то в данном случае ошибка чтения за границей массива невозможна.

Третья тестовая функция реализует подсчет числа символов кодировки utf-8 во входной строке. Рассматриваемая функция проводит проверку выхода за границу строки, но, тем не менее, содержит ошибку доступа к памяти. Исходный текст данной функции представлен на рис. 9. Как видно из исходного текста, ошибка проявляется в случае, если последний символ строки кодирует дополнительные байты utf-8 [15] (т. е. строка содержит последовательность байт, для которых

```
int utf_count(const char *b, const char *e){
     const char *p = b;
     int n=0;
     while (p < e){
         if(*p<128){
             ++p;
             ++n;
         }
         else{
             if(*p>=0xC0){
                 ++p;
                 ++n;
             }
             else{
                 while(*p>=128&&*p<0xC0){
                     ++p;
                 }
             }
         }
     }
     return n;
```

Рис. 9. Исходный текст тестовой функции подсчета числа символов utf-8 во входной строке

```
Executing test utf_count_state_2
Error can be obtained
Pointer conditions are:
    0<=E-B
    X0-B==1
    X0-E==0
Variable conditions are:
    128<=X0[-1]<=191
```

Рис. 10. Результат проверки исходного текста тестовой функции подсчета числа символов utf-8

верно условие `*p> = 128&&*p<0xC0`). В этом случае выполняется увеличение указателя `p` во вложенном цикле `while`, в котором проверка границ не проводится.

Результат проверки исходного текста данной функции представлен на рис. 10.

Условиям, представленным на рис. 10, соответствует последовательность из одного символа, значение которого находится в диапазоне $128 <= *b < 192$.

Заключение

Рассмотрены методы статического анализа исходных текстов программ на предмет наличия в них ошибок доступа к памяти. Предложен и реализован алгоритм поиска ошибок доступа к памяти за границами массивов, основанный на методе обратного выполнения анализируемой программы. На тестовых примерах продемонстрирована работоспособность программной реализации разработанного автором алгоритма анализа.

Список литературы

1. **Международный** стандарт ISO/IEC 9899:2011 Programming languages — C / ISO, 2011. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber = 57853
2. **Официальный** сайт проекта Valgrind. URL: http://valgrind.org/
3. **Cppcheck.** A tool for static C/C++ code analysis. URL: http://cppcheck.sourceforge.net/
4. **Balduin P., Cuoq P., Filliâtre J.** ACSL: ANSI/ISO C Specification Language Version 1.8 — Neon-20140301. CEA LIST and INRIA, 2013. URL: http://frama-c.com/acsl_tutorial_index.html
5. **Marché C.** Verification of the functional behavior of a floating-point program: An industrial case study // Science of Computer Programming. 2014. N. 96. P. 279—296.
6. **Hermenegildo M., Puebla G., Bueno F., López-García P.** Integrated program debugging, verification, and optimization using abstract interpretation (and the Ciao system preprocessor) // Science of Computer Programming. 2005. Vol. 58, N. 1—2. P. 115—140.
7. **Zaks A., Rose J.** Clang Static Analyzer: How to Write a Checker in 24 Hours. URL: http://llvm.org/devmtg/2012-11/Zaks-Rose-Checker24Hours.pdf
8. **Sauciuc R., Necula G.** Reverse Execution With Constraint Solving, Technical Report No. UCB/EECS-2011-67. University of California at Berkeley, 2011. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-67.html
9. **Центр** верификации операционной системы Linux. URL: http://linuxtesting.ru/astraver
10. **Бородин А. Е., Белеванцев А. А.** Статический анализатор Svace как коллекция анализаторов разных уровней сложности // Труды Института системного программирования РАН. 2015. Т. 27, № 6. С. 111—134.
11. **Jaffar J., Navas J. A., Santosa A. E.** Unbounded symbolic execution for program verification // RV'11 Proceedings of the Second international conference on Runtime verification. 2011. P. 396—411.
12. **Cadar C., Sen K.** Symbolic Execution for Software Testing: Three Decades Later // Communications of the Association for Computing Machinery (CACM 2013). 2013. Vol. 56, N. 2. P. 82—90.
13. **Дергачев А. В., Сидорин А. В.** Основанный на резюме метод реализации произвольных контекстно-чувствительных проверок при анализе исходного кода посредством символьного выполнения // Труды Института системного программирования РАН. 2016. Т. 28, № 1. С. 41—62.
14. **De Moura L., Bjørner N.** Z3: An efficient SMT solver // Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, LNCS, Vol. 4963. P. 337—340.
15. **The Unicode®** Standard Version 9.0. URL: http://www.unicode.org/versions/Unicode9.0.0/ch03.pdf#G7404

Program Source Code Static Analysis for Memory Access Error Detection Using Backwards Execution

D. A. Orlov, orlovdmal@mpei.ru, National Research University "Moscow Power Engineering Institute", Moscow, 111250, Russian Federation

Corresponding author:

Orlov Dmitry A., Assistant Professor, National Research University "Moscow Power Engineering Institute", 111250, Moscow, Russian Federation
E-mail: orlovdmal@mpei.ru

Received on September 04, 2016
Accepted on April 05, 2017

This paper is dedicated to static source code analysis for memory access error detection. Brief review of static source code analysis methods is given. Backwards execution method is considered. Such technique is often used in debuggers to make inspection of program state preceding to error state possible. But this method is also applicable to static source code analysis. Symbolic backward execution in some cases can simplify static analysis. For example, it simplifies analysis of conditions which must be satisfied after the loop execution. An algorithm of static source code analysis using backwards symbolic execution is proposed. The algorithm in consideration is finding if out of bounds memory access is possible in input program. The usage of algorithm proposed is restricted to a subset of programs which can be effectively executed backwards, i.e., algorithms which operations are restricted to loops, conditional jumps, indirect memory access, value copying and addition with constant. This restriction simplifies conditions generated during symbolic execution. Most of text data processing algorithms and file format parsing functions belong to this subset. A test implementation of the proposed algorithm was created. The implementation is tested using input programs which implement the following algorithms: strlen without border checking, strlen with border checking and algorithm counting number of utf-8 codepoints in string.

Keywords: *static source code analysis, memory access errors, programming, algorithm theory, program testing, program verification, backwards execution, symbolic execution*

For citation:

Orlov D. A. Program Source Code Static Analysis for Memory Access Error Detection Using Backwards Execution, *Programmnaya Ingeneria,* 2017, vol. 8, no. 7, pp. 291—299.

DOI: 10.17587/prin.8.291-299

References

1. **IEC 9899:2011** Programming languages — C, 2011, available at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber = 57853
2. **Valgrind** homepage, available at: http://valgrind.org/
3. **Cppcheck.** A tool for static C/C++ code analysis, available at: http://cppcheck.sourceforge.net/
4. **Balduin P., Cuoq P., Filliatre J.** ACSL: ANSI/ISO C Specification Language, 2013, CEA LIST and INRIA, available at: http://frama-c.com/acsl_tutorial_index.html
5. **Marche C.** Verification of the functional behavior of a floating-point program: An industrial case study, *Science of Computer Programming*, 2014, no. 96, pp. 279—296.
6. **Hermenegildo M., Puebla G., Bueno F., Lopez-Garcia P.** Integrated program debugging, verification, and optimization using abstract interpretation (and the Ciao system preprocessor), *Science of Computer Programming*, 2005, vol. 58, no. 1—2, pp. 115—140.
7. **Zaks A., Rose J.** Clang Static Analyzer: How to Write a Checker in 24 Hours, available at: http://llvm.org/devmtg/2012-11/Zaks-Rose-Checker24Hours.pdf
8. **Sauciuc R., Necula G.** Reverse Execution With Constraint Solving Technical Report No. UCB/EECS-2011-67, 2011, University of California at Berkeley, available at: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-67.html
9. **Linux** Verification Center, available at: http://linuxtesting.ru/astraver
10. **Borodin A., Belevancev A.** Staticheskij analizator SVACE kak kollekcia analizatorov raznyh urovnej slozhnosti (A Static Analysis Tool SVACE as a Collection of Analyzers with Various Complexity Levels), *Proceedings of ISP RAS*, 2016, vol. 27, no. 6, pp. 111—134 (in Russian).
11. **Jaffar J., Navas J., Santosa A.** Unbounded symbolic execution for program verification, *Proceedings of the Second international conference on Runtime verification*, 2011, pp. 396—411.
12. **Cadar C., Sen K.** Symbolic Execution for Software Testing: Three Decades Later, *Communications of the Association for Computing Machinery (CACM 2013)*, 2013, vol. 56, no. 2, pp. 82—90.
13. **Dergachev A., Sidorin A.** Osnovannyj na rezume metod realizacii proizvolnyh contekstno-chuvstvitelnyh proverok pri analize ishodnogo koda posredstvom simvolnogo vipolnenija (Summary-based method of implementing arbitrary context-sensitive checks for source-based analysis via symbolic execution), 2016, *Proceedings of ISP RAS*, vol. 28, no. 1, pp. 41—62 (in Russian).
14. **De Moura L., Bjorner N.** Z3: An efficient SMT solver, *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, 2008, pp. 337—340.
15. **The Unicode®** Standard Version 9.0, available at: http://www.unicode.org/versions/Unicode9.0.0/ch03.pdf#G7404

УДК 519.722 DOI: 10.17587/prin.8.300-309

А. М. Кукарцев, канд. физ.-мат. наук, ст. преподаватель, e-mail: amkukarcev@yandex.ru,
А. А. Кузнецов, д-р физ.-мат. наук, проф., e-mail: kuznetsov@sibsau.ru,
Сибирский государственный аэрокосмический университет
имени академика М. Ф. Решетнева, г. Красноярск

Об эквиморфном вычислении действия элемента группы Джевонса над булевыми функциями[1]

Рассмотрены алгоритмы вычисления действия множителей канонического представления элемента группы Джевонса над булевыми функциями. Предлагаемые алгоритмы позволяют обрабатывать параллельно массивы значений булевой функции. В результате производительность вычислений повышается в сотни и тысячи раз по отношению к алгоритмам обработки отдельных значений функции. Они сформулированы для абстрактного арифметико-логического устройства, которое соответствует как архитектуре IA-32/IA-64 процессоров ЭВМ, так и другим архитектурам. Для возможности работы с различными разрядностями приведены необходимые рекомендации и расчетные формы для формирования констант под конкретную архитектуру. Также приведены доказательства корректности алгоритмов и оценка их сложности. Они могут быть использованы как для решения уравнений действия группы Джевонса над булевыми функциями, так и для реализации самих таких действий.

Ключевые слова: *действие группы на множестве, группа Джевонса, булевы функции, уравнения действия группы на множестве, эквиморфные группы, абстрактные машины*

Введение

Группа Джевонса действует на множестве булевых функций (далее — БФ) путем отрицания и/или перестановки их аргументов интранзитивно. Определение эквивалентности двух БФ относительно действия группы Джевонса и поиск связывающих их элементов этой группы являются сложнорешаемыми математическими задачами. Они относятся к задачам классификации и сформулированы в работах [1—4].

Решение указанных задач тривиальным алгоритмом (т. е. полным перебором всех элементов группы) имеет временную сложность, соизмеримую с порядком самой группы Джевонса $O(2^n \cdot n!)$, где n — степень группы и/или число аргументов БФ. По результатам численных экспериментов предложенный в работе [5] алгоритм эффективного решения уравнения действия элемента группы Джевонса над БФ обеспечивает в подавляющем большинстве случаев временную сложность $O(n^3)$. При этом в обоих случаях сложность выражается в числе действий элементов группы Джевонса над БФ.

Такое действие не тривиально и сводится к вычислению каждого значения БФ. Пусть временная сложность вычисления результата одного действия будет $\tau(n) = 10^{-9} \cdot 2^n$, где 2^n — число значений БФ и коэффициент 10^{-9} — средняя скорость вычисления одного значения БФ на ЭВМ. Скорость выбрана из расчета частоты ЭВМ в 1 ГГц (с учетом только части накладных расходов для работы самой ЭВМ). Даже при таких допущениях среднее время вычисления решения уравнения при $n = 23$ (эквивалентно 1 мебибайт данных) составляет примерно 30 мин. Для решения многих задач такое время является недопустимо большим.

Современные процессоры ЭВМ [6] позволяют проводить вычисления не одного значения БФ, а сразу их массива. При этом элементы такого массива не могут быть произвольны. Эффективный алгоритм, предложенный в работе [5], основывается на каноническом представлении элемента группы Джевонса, предложенного в работе [7]. Доказанный в работе [8] эквиморфизм группы Джевонса и симметрической группы степени 2^n позволяет вычислять действия множителей канонического представления в виде массивов значений БФ. Вследствие этого временная сложность решения указанных задач может быть снижена в сотни раз. Это не позволит устранить экспоненту из $\tau(n)$, но позволит проводить вычисления для реальных данных ($n < 31$) за разумное время.

В настоящей статье предлагается модель эквиморфного вычислителя, который основывается на трех алгоритмах вычисления действий трех типов множителей канонического представления группы

[1] Работа выполнена при поддержке РФФИ и правительства Красноярского края (проект № 17-47-240318).

Джевонса над БФ. Алгоритмы включают в себя элементарные логические операции, выполняемые над машинными словами: логическое умножение, логическое сложение, правые и левые логические сдвиги и присвоение. Эти операции применяются к разбиению на машинные слова данных, эквивалентных булевым функциям. В результате производительность вычислений повышается пропорционально размеру машинного слова по отношению к обработке единичных значений БФ, а также машинные слова могут обрабатываться параллельно. Это позволяет дополнительно повысить производительность вычислений за счет увеличения числа вычислительных ядер.

Цель настоящей статьи: предложить эквиморфные алгоритмы вычисления действия множителей канонического представления элемента группы Джевонса над БФ и доказать их корректность.

Необходимые определения и опорные суждения

Далее по тексту пусть $k = 2^n$ — число значений БФ; i, j — целые неотрицательные индексы (счетчики), не превосходящие k. Определим $E = \{0, 1\}$ — множество, состоящее из двух элементов. Декартово произведение этого множества на себя определим как $E^n = \underbrace{E \times \cdots \times E}_{n}$. Элементом такого множества будет бинарный вектор (далее — БВ). Для его координат будем использовать числовую нотацию L2R (*left to right*), т. е. большие координаты находятся левее. Обозначим произвольный элемент множества как $x \in E^n$, $x = \{x_{n-1}, \ldots, x_i, \ldots, x_0\}$. Для векторов-аргументов БФ также будем применять нотацию L2R.

Булева функция n аргументов

$$f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$$

реализует отображение $E^n \to E$:

$$\begin{gathered} y = \{f(11\ldots11),\ f(11\ldots10),\ \ldots, \\ f(00\ldots01),\ f(00\ldots00)\}, \\ y_j = f(j),\ 0 \leqslant j < k. \end{gathered} \tag{1}$$

Пусть БФ задана через таблицу истинности [9]. Если однозначно определить порядок следования аргументов, то каждой БФ по формуле (1) можно поставить в соответствие БВ y_f длины k (столбец значений). Все множество БФ n аргументов обозначим как $B(n)$.

Группу инвертирования переменных обозначим как $E_n = (E^n, \oplus)$ [1]. Группу перестановок переменных (симметрическую группу [10]) обозначим как S_n. Группу Джевонса обозначим как D_n [1]. Структурно она является внешним полупрямым произведением, т. е. $D_n = E_n \leftthreetimes S_n$ [11]. В силу специфики целевых объектов условимся, что симметрическая группа действует на множестве чисел $[0; n-1]$. Обозначим нейтральные элементы групп E_n и S_n как e_E и e_S соответственно.

Пусть $f, g \in B(n)$. Обозначим через $z \in E_n$ — отрицание аргументов и через $\pi \in S_n$ — перестановку аргументов. Совместно отрицание и/или перестановку аргументов обозначим как $(z\pi) \in D_n$, тогда уравнение действия элемента группы Джевонса на БФ относительно неизвестного $(z\pi)$ представим в виде

$$f^{(z\pi)} = g. \tag{2}$$

Важно разделять групповую операцию в симметрической группе и ее действие на БВ. Эмпирически показано, что для задач действия группы Джевонса на БФ удобно задавать гомоморфизм внешнего полупрямого произведения группы Джевонса из симметрической группы в группу автоморфизмов группы инвертирования переменных через само действие элемента на БФ. Такое действие может быть задано не единственным способом. Эмпирически показано, что можно выделить два набора действий: типа А и типа Б. Оба типа действий выражаются друг через друга. При действии элемента группы Джевонса на БВ предпочтителен тип А [8, 12], но при действии на БФ — тип Б [8, 12]. Поэтому невозможно выделить какой-то один тип действий как основной, а второй как сводимый к нему. В то же время разработчик может выбирать тип действия по своему усмотрению в зависимости от решаемой задачи.

Действием типа А элемента группы $\pi \in S_n$ над БВ $x \in E^n$ будем называть вычисление результата $x' \in E^n : x' = x^\pi$ по следующему правилу: в верхней строке подстановки находятся "старые" индексы, а в нижней — "новые" или в символьном виде

$$x'_{\pi(i)} = x_i. \tag{3.А}$$

Действием типа Б элемента группы $\pi \in S_n$ над БВ $x \in E^n$ будем называть вычисление результата $x' \in E^n : x' = x^\pi$ по следующему правилу: в верхней строке подстановки находятся "новые" индексы, а в нижней — "старые" или в символьном виде:

$$x'_i = x_{\pi(i)}. \tag{3.Б}$$

Для доказательства теорем потребуются следующие леммы из работы [12].

Лемма 1.А (О действии типа А). Если подстановки $\pi, \rho \in S_n$ действуют последовательно (π и затем ρ) над элементом $x \in E^n$, то результат действия эквивалентен действию подстановки-произведения $\pi\rho$ или в символьном виде $(x^\pi)^\rho = x^{(\pi\rho)}$.

Лемма 1.Б (О действии типа Б). Если подстановки $\pi, \rho \in S_n$ действуют последовательно (π и затем ρ) над элементом $x \in E^n$, то результат действия эквивалентен действию подстановки-произведения $\rho\pi$ или в символьном виде $(x^\pi)^\rho = x^{(\rho\pi)}$.

Так как в одном выражении указываются как групповые операции, так и действия, например $x^{\pi\rho}$, то возникает неоднозначность в приоритете выпол-

нения операций. Для определенности примем, что действие имеет минимальный приоритет, т. е. для $x^{\pi\rho}$ нужно сначала вычислить произведение $\pi\rho$, а только затем действие над x результатом произведения. Если необходим другой порядок выполнения операций, то условимся использовать скобки для изменения приоритета. Например, $(x^{\pi})^{\rho}$ обозначает последовательное применение действий, сначала π над БВ, а затем ρ над результатом первого действия.

Два типа действия дают два различных представления группы Джевонса. В работе [12] доказана корректность следующих определений групповых операций для $z_0, z_1 \in E_n$ и $\pi_0, \pi_1 \in S_n$:

$$(z_0\pi_0)(z_1\pi_1) = \left(z_0 z_1^{\pi_0^{-1}} \pi_0\pi_1\right); \tag{4.А}$$

$$(z_0\pi_0)(z_1\pi_1) = \left(z_0 z_1^{\pi_0^{-1}} \pi_1\pi_0\right). \tag{4.Б}$$

Определим действие элемента группы Джевонса над БВ как

$$x^{(z\pi)} = (x^z)^{\pi}. \tag{5}$$

Действие над булевой функцией элементом группы Джевонса задается сложнее (необходимо учесть перестановочность действующих элементов при вычислении композиции):

$$f^{(z\pi)} = f\left(x^{(z\pi)}\right) = f\left(\left(x^z\right)^{\pi}\right) = \left(f^{(e_E\pi)}\right)^{(ze_S)}. \tag{6}$$

В работе [12] доказаны изоморфизмы (для типа А) и антиизоморфизмы (для типа Б) множителей полупрямого произведения подгрупп Джевонса. Элементы z, π подгрупп Джевонса E_n и S_n изоморфно (антиизоморфно) отображаются на элементы $\varphi_z, \varphi_\pi \in \beta_n : \beta_n < S_k$ соответственно:

$$\varphi_z = \begin{pmatrix} 2^n - 1 & \cdots & j & \cdots & 0 \\ \sum\limits_{i=n-1}^{0} \overline{z_i} \cdot 2^i & \cdots & \sum\limits_{i=n-1}^{0} j_i \cdot 2^i & \cdots & \sum\limits_{i=n-1}^{0} z_i \cdot 2^i \end{pmatrix}; \tag{7}$$

$$\varphi_\pi = \begin{pmatrix} 2^n - 1 & \cdots & j & \cdots & 0 \\ 2^n - 1 & \cdots & \sum\limits_{i=n-1}^{0} j_i \cdot 2^{\pi(i)} & \cdots & 0 \end{pmatrix}; \tag{8.А}$$

$$\varphi_\pi = \begin{pmatrix} 2^n - 1 & \cdots & j & \cdots & 0 \\ 2^n - 1 & \cdots & \sum\limits_{i=n-1}^{0} j_{\pi(i)} \cdot 2^i & \cdots & 0 \end{pmatrix}. \tag{8.Б}$$

При этом действие образов φ_z, φ_π по формуле (1) на БВ y_f, эквивалентный БФ $f \in B(n)$, соответствует действию прообразов на эту БФ. В работе [8] доказано изоморфное и эквиморфное вложение группы Джевонса в подгруппу β_n.

Определение 1. Две группы G, G', действующие на некотором множестве M, будем называть **эквиморфными**, если существует биекция $\varphi : G \to G'$, такая, что для любых $a, b \in G$ и $m \in M$:

$$\left(m^a\right)^b = \left(m^{\varphi(a)}\right)^{\varphi(b)}. \tag{9}$$

При этом отображение φ будем называть эквиморфизмом.

Теорема 1 (Об эквиморфизме групп Джевонса и β_n). *Группа Джевонса D_n эквиморфна группе β_n по действию над БФ по типу Б и эквиморфна в обратные (отрицательные) элементы β_n по действию над БФ по типу А, и эквиморфизмами будут композиции отображений (7), (8.А) и (8.Б).*

Эквиморфизмом типа А будет $(z\pi) \to \varphi_{(z\pi)}^{-1} = \varphi_\pi^{-1}\varphi_z^{-1}$, последние отображения должны вычисляться по формулам (7) и (8.А). Эквиморфизмом типа Б будет $(z\pi) \to \varphi_{(z\pi)} = \varphi_z\varphi_\pi$, последние отображения должны вычисляться по формулам (7) и (8.Б).

Пусть $b_{n-1}, \ldots, b_i, \ldots, b_0, z \in E_n$, причем $b_i = \{0, \ldots, 0, \ldots, z_i, \ldots, 0, \ldots, 0\}$, т. е. содержит на позиции i значение координаты i БВ z, а на остальных — нули. Тогда, опираясь на работу [7], определим каноническое представление элемента группы Джевонса:

$$\begin{aligned}(z\pi) = \left(b_{n-1}(n-1, j_{n-1})\right)\cdots\left(b_{i_1}(i_1, j_{i_1})\right)\cdots\left(b_i(i, j_i)\right)\cdots \\ \cdots\left(b_{i_0}(i_0, j_{i_0})\right)\cdots\left(b_0(0, j_0)\right).\end{aligned} \tag{10}$$

В формуле (10) $(i, j_i) : 0 \leqslant i \leqslant j < n$ — транспозиции группы S_n и $0 \leqslant \ldots < i_0 < \ldots < i < \ldots < i_1 < \ldots < n-1$.

Определим множество векторов $c_{n-1}, \ldots, c_0 \in E^n$, где каждый c_i имеет на позиции i значение 1, а на остальных — 0, т. е. все множество c_i представляет собой порождающее множество группы E_n.

Эквиморфный вычислитель

Рассмотрим принцип работы эквиморфного вычислителя на примере действия элементом $\{010\} \in E_3$ над БВ $y \in E^8 : y = \{y_7, y_6, y_5, y_4, y_3, y_2, y_1, y_0\}$, эквивалентным некоторой БФ $f(x_2, x_1, x_0) \in B(3)$. Согласно формуле (6) результатом будет $y' \in E^8 : y' = \{y_5, y_4, y_7, y_6, y_1, y_0, y_3, y_2\}$.

Фактически y' получен перестановкой четных и нечетных пар значений y. Рассмотрим детально и разделим значения БВ y как $y_{\text{lo}} = \{0, 0, y_5, y_4, 0, 0, y_1, y_0\}$ и $y_{\text{hi}} = \{y_7, y_6, 0, 0, y_3, y_2, 0, 0\}$. Далее сдвинем координаты y_{lo} в сторону больших индексов на две позиции, а y_{hi} — в сторону меньших: $y_{\text{lo}}^{\leftarrow} = \{y_5, y_4, 0, 0, y_1, y_0, 0, 0\}$ и $y_{\text{hi}}^{\rightarrow} = \{0, 0, y_7, y_6, 0, 0, y_3, y_2\}$. В результате непосредственно заключаем, что $y' = y_{\text{lo}}^{\leftarrow} \oplus y_{\text{hi}}^{\rightarrow}$. Аналогично можно вычислить каждое действие на любой итерации предложенного в работе [5] эффективного алгоритма решения уравнения действия элемента группы Джевонса над БФ. Для этого потребуется некоторый абстрактный вычислитель, который

может обрабатывать подмножество смежных значений БВ применительно к примеру — восемь значений. Рассмотрим операции, которые должен выполнять абстрактный вычислитель. Всего потребуется пять элементарных операций. Эти операции могут указываться совместно в одном выражении, поэтому определим для каждой операции приоритет выполнения. Условимся, что приоритет может меняться с помощью применения скобок. Приведем определения для указанных элементарных операций:

а) логическое покомпонентное умножение векторов (логическое "И") или в символьном виде $x \,\&\, y = \{x_{n-1} \,\&\, y_{n-1},\ ...,\ x_i \,\&\, y_i,\ ...,\ x_0 \,\&\, y_0\},\ \forall x, y \in E^n$; имеет высокий приоритет;

б) логическое покомпонентное сложение векторов (логическое "ИЛИ") или в символьном виде $x \vee y = \{x_{n-1} \vee y_{n-1},\ ...,\ x_i \vee y_i,\ ...,\ x_0 \vee y_0\},\ \forall x, y \in E^n$; имеет средний приоритет;

в) логический левый сдвиг вектора на целое неотрицательное число или в символьном виде $x \ll w = \{x_{n-w-1},\ ...,\ x_w, \underbrace{0, \ldots, 0}_{w}\}, \forall x \in E^n, 0 \leqslant w < n$; имеет низкий приоритет;

г) логический правый сдвиг вектора на целое неотрицательное число или в символьном виде $x \gg w = \{\underbrace{0,\ ...,\ 0}_{w},\ x_{n-1},\ ...,\ x_w\}, \forall x \in E^n, 0 \leqslant w < n$; имеет низкий приоритет;

д) покомпонентное присвоение вектора $y = x : y = \{x_{n-1},\ ...,\ x_i,\ ...,\ x_0\}, \forall x,\ y \in E^n$; имеет низший приоритет.

Применительно к предыдущему примеру верно $y' = \left(y \,\&\, \{0011\ 0011\} \ll 2\right) \vee \left(y \,\&\, \{1100\ 1100\} \gg 2\right)$, т. е. логическое поразрядное сложение двух поразрядных логических умножений со сдвигом.

Определение 2. Слово — бинарный вектор длины $2^{n'}$, где n' — степень вычислителя, над которым вычислителем выполняются элементарные операции *а—д*.

Определение 3. Символ — подмножество смежных значений булева вектора, такие, что при перестановке всех значений БВ соответствующие символу значения сохраняют относительный порядок. Другими словами, перестановка значений БВ, соответствующих слову, сводится к их линейному сдвигу на одинаковое значение.

Определение 4. Блок — подмножество смежных символов БВ, перестановка значений каждого из которых сводится к нескольким линейным сдвигам на минимальное значение. Значение величины сдвига выбирается наименьшее, но сохраняющее символы, и определяется типом множителя канонического представления элемента группы Джевонса.

Применительно к предыдущему примеру символами будут четыре пары смежных значений БВ, имеющих четные и нечетные индексы: $\{\{y_7, y_6\}, \{y_5, y_4\}, \{y_3, y_2\}, \{y_1, y_0\}\}$. Блок будет содержать пару символов, причем одна будет сдвигаться на размер двух символов влево, другая — вправо. Всего будет два блока $\left\{\left[\{y_7, y_6\}, \{y_5, y_4\}\right], \left[\{y_3, y_2\}, \{y_1, y_0\}\right]\right\}$, которые составляют БВ целиком, и он совпадает по размеру со словом степени вычислителя $n' = \log_2 8 = 3$. Степень эквиморфного вычислителя определяется аппаратными возможностями процессора, на котором он будет реализован, например, для архитектур IA-32 и IA-64/AMD64 степени будут 5 и 6 соответственно.

Рассмотрим произвольный множитель канонического представления элемента группы Джевонса $\left(b_i(i, j_i)\right)$. Из определения b_i возможно только два значения: e_E и c_i. Для транспозиции также возможны два варианта: e_S (в случае $i = j_i$) и (i, j_i) в остальных случаях. Тогда возможны четыре вида действия:

а) тривиальное — в случае $(e_E e_S)$; алгоритмическое преобразование также тривиально и специальные алгоритмы не требуются;

б) только инверсия координаты — $(c_i e_S)$;

в) только транспозиция координат —

$$\left(e_E(i, j_i)\right) : i < j_i;$$

г) инверсия и транспозиция координат —

$$\left(c_i(i, j_i)\right) : i < j_i.$$

Для каждого типа действия создан реализующий алгоритм и доказана его корректность. Для рассматриваемых далее алгоритмов предполагается разбиение БВ $y, y' \in E^k$ на векторы-слова $y_{index}, y'_{index} \in E^{n'} : 0 \leqslant index < w$ соответственно, где $w = 2^{n-n'}$ (если $n < n'$, то принять что $w = 1$) — число слов.

Алгоритм 1 ("Об эквиморфном вычислении действия E_n").

Вход: бинарный вектор $y \in E^k$ длины k, эквивалентный $f \in B(n)$, и значение $i : 0 \leqslant i < n$.

Выход: бинарный вектор $y' \in E^k$ длины k, эквивалентный $f^{(c_i e_S)} \in B(n)$.

Для эквиморфного вычисления БВ $y' \in E^k$ длины k, эквивалентного БФ f^{c_i} — результату действия элемента $c_i \in E_n$ над БФ $f \in B(n)$, заданной в виде эквивалентного БВ $y \in E^k$ длины k, нужно выполнить следующие шаги.

Шаг 1. Разбить БВ y, y' на слова. Если $i < n'$, то перейти к шагу 2, иначе перейти к шагу 3.

Шаг 2. $\forall u : 0 \leqslant u < w$ вычислить

$y'_u = \left(y_u \,\&\, \mathrm{lo} \ll s_E\right) \vee \left(y_u \,\&\, \mathrm{hi} \gg s_E\right)$, где

$$s_E = 2^i,\ \ w_b = 2^{n'-i-1},\ \ \mathrm{lo} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_b}\}$$

$$\text{и}\ \ \mathrm{hi} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_b}\}.$$

Вернуть y' и завершить алгоритм.

Шаг 3. $\forall v,u: 0 \leqslant v < b_c, 0 \leqslant u < s_w$ вычислить $y'_{idx} = y_{idx+s_w}$ и $y'_{idx+s_w} = y_{idx}$, где $idx = v \cdot b_w + u$, $b_c = 2^{n-i-1}$, $b_w = 2^{i+1-n'}$ и $s_w = 2^{i-n'}$. Вернуть y' и завершить алгоритм.

Теорема 2 (Об эквиморфном вычислении действия E_n). *Преобразование БВ $y \in E^k$, эквивалентного БФ $f \in B(n)$, согласно алгоритму 1 равносильно действию эквиморфизма φ_{c_i} над ним, получаемого по формуле (7). Число операций эквиморфного вычислителя для $i < n'$ составит $2w$ сдвигов, $2w$ логических умножений, w логических сложений, w присвоений, а для $i \geqslant n'$ — w присвоений.*

Доказательство. Рассмотрим цикловую структуру образа-подстановки $\varphi_{c_i} \in S_k$, получаемой по формуле (7) из прообраза $c_i \in E_n$. Во-первых, каждая точка $x: 0 \leqslant x < k$ подстановки переходит в какую-то другую точку посредством инверсии координаты i, т. е. $\varphi_{c_i}(x) = \{x_{n-1}, \ldots, \overline{x_i}, \ldots, x_0\}$, и тогда верно $\varphi_{c_i}(\varphi_{c_i}(x)) = x$ и, как следствие, φ_{c_i} — инволюция и все точки подстановки подвижны. Поэтому вся подстановка состоит из 2^{n-1} независимых циклов длины 2 вида $(\{x_{n-1}, \ldots, x_i, \ldots, x_0\}\{x_{n-1}, \ldots, \overline{x_i}, \ldots, x_0\})$. Во-вторых, инверсия координаты точки есть фактически либо арифметическое сложение вида $(x, x+2^i)$ в случае $x_i = 0$, либо арифметическое вычитание вида $(x, x-2^i)$ в случае $x_i = 1$.

Разобьем все множество точек на 2^{n-i} подмножеств так, что внутри них точки имеют одинаковые координаты $x_{n-1}, \ldots, x_i$ и упорядочены по координатам $x_{i-1}, \ldots, x_0$. Тогда при перестановке подмножеств будут переставляться их точки, не нарушая порядок, т. е. подмножество является символом размером $s_E = 2^i$ и индексом символа будут являться значения координат точек $x_{n-1}, \ldots, x_i$. При этом каждый символ с индексом $x_{n-1}, \ldots, x_{i+1}, 0$ будет отображаться в символ $x_{n-1}, \ldots, x_{i+1}, 1$ и наоборот. Действие локализовано внутри пары символов, которые составляют блок размером 2^{i+1} бит. Индексом блока будут координаты $x_{n-1}, \ldots, x_{i+1}$.

Для $i < n'$ размер блока меньше или равен размеру слова, поэтому при обработке одного слова будет обработано $w_b = 2^{n'-i+1}$ блоков. Нужно выделить в каждом блоке слова символы с четными и нечетными индексами и поменять их местами. Выделение символов эквивалентно логическому умножению на константы: для выделения символов с четными индексами (младших полублоков) $\mathrm{lo} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_b}\}$ и символов с нечетными индексами (старших полублоков) $\mathrm{hi} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_b}\}$. Для взаимной перестановки достаточно символы с нечетными индексами сдвинуть влево на размер символа в битах, а с четными — вправо. Откуда получаем $y'_u = (y_u \,\&\, \mathrm{lo} \ll s_E) \vee (y_u \,\&\, \mathrm{hi} \gg s_E)$. Слово содержит один или несколько блоков, как следствие, действие локализовано внутри слова. Всего слов $w = 2^{n-n'}$. В случае если размер БВ y меньше размера блока ($n < n'$), то нужно БВ дополнить нулями слева до размера слова и принять, что $w = 1$ и в результирующем слове отбросить это дополнение. Все результирующие слова нужно обработать в любом порядке (допустима параллельная обработка по всем u) $y'_u: 0 \leqslant u < 2^{n-n'}$. Всего будет затрачено (исходя из преобразования одного слова): $2w$ логических сдвигов, $2w$ логических умножений, w логических сложений, w присвоений.

Для $i \geqslant n'$ размер блока больше размера слова, а размер символа не меньше размера слова. Правила перестановки будут аналогичны случаю $i < n'$, но при этом ориентированы на размер символа в словах $s_w = 2^{i-n'}$. Блок, также как и в случае $i < n'$, состоит из четного и нечетного символов и имеет размер в словах $b_w = 2^{n-i-1}$. При этом символы являются полублоками. Индекс каждого четного слова и соответственный ему индекс нечетного слова различаются на размер символа s_w. Для выполнения действия нужно переставить в любом порядке (допускается параллельная обработка по всем v, u) все соответственные четные и нечетные слова для каждого блока $y'_{idx+u} = y_{idx+u+s_w}$ и $y'_{idx+u+s_w} = y_{idx+u}$. Всего блоков (исходя из размера символов) будет $b_c = 2^{n-i-1}$ и индекс первого слова каждого блока будет $idx = v \cdot b_w$, где $0 \leqslant v < b_c$ — индекс блока. Так как обрабатывается сразу пара символов блока, то на обработку всего блока требуется $b_w/2$ шагов, что эквивалентно размеру символа в словах s_w, и индекс шага будет $u: 0 \leqslant u < s_w$. При этом будет обработано $w = 2^{n-n'}$ слов. Так как обработка сводится к перестановке всех слов, то всего будет затрачено w присвоений. Что и требовалось доказать.

Алгоритм 2 ("Об эквиморфном вычислении действия S''_n").

Вход: бинарный вектор $y \in E^k$ длины k, эквивалентный $f \in B(n)$, и значения $i, j: 0 \leqslant i < j < n$.

Выход: бинарный вектор $y' \in E^k$ длины k, эквивалентный $f^{(e_E(i,j))} \in B(n)$.

Для эквиморфного вычисления БВ $y' \in E^k$ длины k, эквивалентного БФ $f^{(i,j)}$ — результату действия транспозиции $(i,j) \in S_n$ над БФ $f \in B(n)$, заданной в виде эквивалентного БФ $y \in E^k$ длины k, нужно выполнить следующие шаги.

Шаг 1. Разбить БВ y, y' на слова. Если $j < n'$, то перейти к шагу 2. В противном случае, если $i < n'$, то перейти к шагу 3, иначе перейти к шагу 4.

Шаг 2. $\forall u: 0 \leqslant u < w$ вычислить $y'_u = y_u \,\&\, \mathrm{st} \vee \vee (y_u \,\&\, \mathrm{lo} \ll val_E) \vee (y_u \,\&\, \mathrm{hi} \gg val_E)$, где

$$\mathrm{st} = \overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{sb_{ps}}}^{w_b},\quad \mathrm{lo} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}}^{w_b},$$

$$\mathrm{hi} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}}^{w_b},\quad s_E = 2^i,\quad w_b = 2^{n'-j-1},$$

$$sb_{ps} = 2^{j-i-1} \text{ и } val_E = 2^j - 2^i.$$

Вернуть y' и завершить алгоритм.

Шаг 3. $\forall v, u: 0 \leqslant v < b_c,\ 0 \leqslant u < sb_w$ вычислить $y'_{idx} = y_{idx} \& \mathrm{st_{lo}} \vee \left(y_{idx+sb_w} \& \mathrm{hi} \ll s_E\right)$ и $y'_{idx+sb_w} = y_{idx+sb_w} \& \mathrm{st_{hi}} \vee \vee \left(y_{idx} \& \mathrm{lo} \gg s_E\right)$, где $idx = vb_w + u$, $s_E = 2^i$, $w_{ps} = 2^{n'-i-1}$,

$$\mathrm{st_{lo}} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_{ps}}\},\quad \mathrm{st_{hi}} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_{ps}}\},\quad \mathrm{lo} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_{ps}}\},$$

$$\mathrm{hi} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_{ps}}\},\quad b_c = 2^{n-j-1},\quad b_w = 2^{j-n'+1} \text{ и } sb_w = 2^{j-n'}.$$

Вернуть y' и завершить алгоритм.

Шаг 4. $\forall v, p, u: 0 \leqslant v < b_c,\ 0 \leqslant p < sb_{ps},\ 0 \leqslant u < s_w$ вычислить

$y'_{idx} = y_{idx}$, $y'_{idx+s_w} = y_{idx+s_w+val_w}$, $y'_{idx+s_w+val_w} = y_{idx+s_w}$ и $y'_{idx+2s_w+val_w} = y_{idx+2s_w+val_w}$, где $idx = v \cdot b_w + p \cdot ps_w + u$, $s_w = 2^{i-n'}$, $b_c = 2^{n-j-1}$, $b_w = 2^{j+1-n'}$, $sb_{ps} = 2^{j-i-1}$, $ps_w = 2^{i+1-n'}$ и $val_w = 2^{j-n'} - 2^{i-n'}$.

Вернуть y' и завершить алгоритм.

Теорема 3 (Об эквиморфном вычислении действия S_n). *Преобразование БВ $y \in E^k$, эквивалентного БФ $f \in B(n)$, согласно алгоритму 2 равносильно действию эквиморфизма $\varphi_{(i,j)}$ над ним, получаемого по формуле (8.А) или (8.Б). Число операций эквиморфного вычислителя для $j < n'$ составит 2w сдвигов, 3w логических умножений, 2w логических сложений, w присвоений, а для $j \geqslant n'$ при $i < n'$ — w сдвигов, 2w логических умножений, w логических сложений, w присвоений, и при $i \geqslant n'$ — w присвоений.*

Доказательство. Рассмотрим цикловую структуру образа-подстановки $\varphi_{(i,j)} \in S_k$, получаемой по формуле (8.А) или (8.Б) (для транспозиции результат обеих формул будет совпадать) из прообраза $(i,j) \in S_n$. Во-первых, каждая точка $x: 0 \leqslant x < k$ подстановки переходит в какую-то другую точку посредством перестановки координат i и j, т. е. $\varphi_{(i,j)}(x) = \{x_{n-1}, \ldots, x_i, \ldots, x_j, \ldots, x_0\}$ и тогда верно $\varphi_{(i,j)}\left(\varphi_{(i,j)}(x)\right) = x$ и, как следствие, $\varphi_{(i,j)}$ — инволюция и половина точек подстановки подвижны ($x_i \neq x_j$). Поэтому вся подстановка состоит из 2^{n-2} независимых циклов длины 2 вида $\left(\{x_{n-1}, \ldots, x_j, \ldots, x_i, \ldots, x_0\}\{x_{n-1}, \ldots, x_i, \ldots, x_j, \ldots, x_0\}\right)$.

Во-вторых, перестановка координат точки для $x_i \neq x_j$ есть фактически либо арифметическая операция вида $\left(x, x + 2^j - 2^i\right)$ в случае $x_i = 1,\ x_j = 0$ либо арифметическая операция вида $\left(x,\ x - 2^j + 2^i\right)$ в случае $x_i = 0,\ x_j = 1$. Другими словами, точка будет сдвигаться на постоянное значение $\left|2^j - 2^i\right|$. Несмотря на то что по условию $i < j$, абсолютная величина показывает, что значение сдвига есть расстояние.

Разобьем все множество точек на 2^{n-i} подмножеств так, что внутри них точки имеют одинаковые координаты $x_{n-1}, \ldots, x_i$ и упорядочены по координатам $x_{i-1}, \ldots, x_0$. Тогда при перестановке подмножеств будут переставляться их точки, не нарушая порядок, т. е. подмножество является символом размером $s_E = 2^i$ и индексом символа будут являться значения координат точек $x_{n-1}, \ldots, x_i$. При этом символы с индексами $x_i = x_j$ будут оставаться на месте, а символы $x_i \neq x_j$ — меняться местами. В результате действие сводится к работе с четверками символов и локализуется внутри их массива с общими координатами $x_{n-1}, \ldots, x_{j+1}$, т. е. он составляет блок размером 2^{j+1} бит. Всего четверок в блоке будет $\dfrac{2^{j+1}/2^i}{4} = 2^{j-i-1}$.

Индексом блока будут координаты $x_{n-1}, \ldots, x_{j+1}$. Блок состоит из двух полублоков: старшего ($x_j = 1$) и младшего ($x_j = 0$). Полублоки состоят из $sb_{ps} = \dfrac{2^{j+1}/2}{2^i}\Big/2 = 2^{j-i-1}$ пар символов, причем каждая пара старшего полублока содержит неподвижный ($x_i = 1$) и подвижный ($x_i = 0$) символы, а младшего наоборот — подвижный ($x_i = 1$) и неподвижный ($x_i = 0$) символы.

Для $j < n'$ и из условия $i < j$ верно, что $i < n'$. Размер блока меньше или равен размеру слова, поэтому при обработке одного слова будет обработано $w_b = 2^{n'-j-1}$ блоков. Нужно выделить в каждом блоке слова неподвижные(ый) и подвижные(ый) символы, соответствующие старшим и младшим полублокам. Выделение символов эквивалентно логическому умножению на константы. Размер блока меньше либо равен размеру слова, поэтому неподвижные символы младших(его) и старших(его) полублоков можно выделить вместе одной константой $\mathrm{st} = \overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{sb_{ps}}}^{w_b}$.

Для выделения подвижных символов старших(его) и младших(его) полублоков подходят константы

$$\mathrm{lo} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}}^{w_b} \quad \text{и} \quad \mathrm{hi} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}}^{w_b}.$$

Для взаимной перестановки подвижных символов младших(его) и старших(его) полублоков достаточно их сдвинуть на размер символа в битах $val_E = 2^j - 2^i$ влево и вправо соответственно. Откуда получаем $y'_u = y_u \,\&\, \mathrm{st} \vee (y_u \,\&\, \mathrm{lo} \ll val_E) \vee (y_u \,\&\, \mathrm{hi} \gg val_E)$. Слово содержит один или несколько блоков, как следствие, действие локализовано внутри слова. Всего слов $w = 2^{n-n'}$. В случае если размер БВ y меньше размера блока ($n < n'$), то нужно БВ дополнить нулями слева до размера слова и принять, что $w = 1$, и в результирующем слове отбросить это дополнение. Все результирующие слова нужно обработать в любом порядке (допустима параллельная обработка по всем u) $y'_u : 0 \leqslant u < 2^{n-n'}$. Всего будет затрачено (исходя из преобразования одного слова): $2w$ сдвигов, $3w$ логических умножений, $2w$ логических сложений, w присвоений.

Для $j \geqslant n'$ возможны два случая. Первый случай — $i < n'$, второй случай — $i \geqslant n'$. В первом случае размер символа меньше либо равен размеру слова, а размер блока больше размера слова. Поэтому слово содержит символы либо младшего, либо старшего полублока. Всего блоков будет $b_c = 2^{n-j-1}$ размером $b_w = 2^{j+1-n'}$ слов. Каждое слово состоит из пар подвижных(ого) и неподвижных(ого) символов и в каждом слове этих пар будет $w_{ps} = \dfrac{2^{n'}/2^i}{2} = 2^{n'-i-1}$. Выделение символов эквивалентно логическому умножению на константы. Для слов младшего полублока константы выделения неподвижных и подвижных символов будут $\mathrm{st_{lo}} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_{ps}}\}$ и $\mathrm{lo} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_{ps}}\}$ соответственно, для слов старшего полублока — $\mathrm{st_{hi}} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_{ps}}\}$ и $\mathrm{hi} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_{ps}}\}$. Неподвижные символы должны остаться на месте, а подвижные — сместиться на значение $\left|2^j - 2^i\right|$. Подвижные символы младшего полублока сместятся влево, старшего — вправо. Для удобства значение смещения можно распределить следующим образом: на 2^j будут смещаться целые слова, поэтому можно просто смещать слово на значение размера полублока в словах $sb_w = 2^{j-n'}$. Вторую отрицательную часть значения смещения — $2^i = -s_E$ в битах нужно учитывать внутри слов. Откуда имеем $y'_{idx} = y_{idx} \,\&\, \mathrm{st_{lo}} \vee \left(y_{idx+sb_w} \,\&\, \mathrm{hi} \ll s_E\right)$ и $y'_{idx+sb_w} = y_{idx+sb_w} \,\&\, \mathrm{st_{hi}} \vee \left(y_{idx} \,\&\, \mathrm{lo} \ll s_E\right)$, где $idx = v \cdot b_w + u$ — базовый индекс каждого слова $u : 0 \leqslant u < sb_w$ в полублоке каждого блока $v : 0 \leqslant v < b_c$. Все результирующие слова нужно обработать в любом порядке (допустима параллельная обработка по всем u, v). Всего будет затрачено (исходя из преобразования одного слова): w сдвигов, $2w$ логических умножений, w логических сложений, w присвоений.

Во втором случае — $i < n'$ размер символа больше либо равен размеру слова, а размер блока и полублока больше размера слова. Поэтому нужно переставлять символы по словам целиком, при этом размер символа в словах будет $s_w = 2^{i-n'}$. Всего блоков будет $b_c = 2^{n-j-1}$ размером $b_w = 2^{j+1-n'}$ слов. В каждом блоке должны обрабатываться четверки символов так, что одна пара относится к младшему полублоку, а вторая — к старшему. Всего пар символов в полублоке будет $sb_{ps} = \dfrac{2^{j+1}/2}{2^i} \Big/ 2 = 2^{j-i-1}$ и размер каждой пары в словах составит $ps_w = s_w = 2^{i-n'+1}$. Младший символ в первой паре и старший символ во второй паре неподвижны, остальные подвижны. Подвижные символы сдвигаются на значение $\left|2^j - 2^i\right|$, что в пересчете на слова составляет $val_w = 2^{j-n'} - 2^{i-n'}$. Откуда имеем $y'_{idx} = y_{idx}$, $y'_{idx+s_w} = y_{idx+s_w+val_w}$, $y'_{idx+s_w+val_w} = y_{idx+s_w}$ и $y'_{idx+2s_w+val_w} = y_{idx+2s_w+val_w}$, где $idx = v \cdot b_w + p \cdot ps_w + u$ — базовый индекс слова четверки слов каждого блока $v : 0 \leqslant v < b_c$ каждой пары символов полублока $p : 0 \leqslant p < sb_{ps}$ каждого слова символа $u : 0 \leqslant u < s_w$. Все результирующие слова нужно обработать в любом порядке (допустима параллельная обработка по всем u, p, v). Всего будет затрачено (исходя из преобразования одного слова) w присвоений. Что и требовалось доказать.

Алгоритм 3 ("Об эквиморфном вычислении действия D_n").

Вход: бинарный вектор $y \in E^k$ длины k, эквивалентный $f \in B(n)$, и значения $i, j : 0 \leqslant i < j < n$.

Выход: бинарный вектор $y' \in E^k$ длины k, эквивалентный $f^{(c_i(i,j))} \in B(n)$.

Для эквиморфного вычисления БВ $y' \in E^k$ длины k, эквивалентного БФ $f^{(c_i(i,j))}$ — результату действия элемента $(c_i(i,j)) \in D_n$ над БФ $f \in B(n)$, заданной в виде эквивалентного БФ $y \in E^k$ длины k, нужно выполнить следующие шаги.

Шаг 1. Разбить БВ y, y' на слова. Если $j < n'$, то перейти к шагу 2. В противном случае, если $i < n'$, то перейти к шагу 3, иначе перейти к шагу 4.

Шаг 2. $\forall u : 0 \leqslant u < w$ вычислить $y'_u = (y_u \,\&\, \mathrm{st_{lo}} \ll s_E) \vee (y_u \,\&\, \mathrm{st_{hi}} \gg s_E) \vee (y_u \,\&\, \mathrm{lo} \ll val_E) \vee \vee (y_u \,\&\, \mathrm{hi} \gg val_E)$, где $w_b = 2^{n'-j-1}$, $sb_{ps} = 2^{j-i-1}$,

$\mathrm{st_{lo}} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_b}$, $\mathrm{st_{hi}} = \overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_b}$,

$$\mathrm{lo} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}}^{w_b}, \qquad \mathrm{hi} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}}^{w_b},$$

$val_E = 2^j$. Вернуть y' и завершить алгоритм.

Шаг 3. $\forall v,u : 0 \leqslant v < b_c,\ 0 \leqslant u < sb_w$ вычислить $y'_{idx} = (y_{idx} \,\&\, \mathrm{st_{lo}} \ll s_E) \vee y_{idx+sb_w} \,\&\, \mathrm{hi}$ и $y'_{idx+sb_w} = \left(y_{idx+sb_w} \,\&\, \mathrm{st_{hi}} \gg s_E\right) \vee y_{idx} \,\&\, \mathrm{lo}$, где $idx = v \cdot b_w + u$, $s_E = 2^i$, $w_{ps} = 2^{n'-i-1}$, $\mathrm{st_{lo}} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_{ps}}\}$, $\mathrm{st_{hi}} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_{ps}}\}$, $\mathrm{lo} = \{\overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{w_{ps}}\}$, $\mathrm{hi} = \{\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{w_{ps}}\}$, $b_c = 2^{n-j-1}$, $b_w = 2^{j+1-n'}$, $sb_w = 2^{j-n'}$.

Вернуть y' и завершить алгоритм.

Шаг 4. $\forall v,\ p,\ u : 0 \leqslant v < b_c,\ 0 \leqslant p < sb_{ps}, 0 \leqslant u < s_w$ вычислить $y'_{idx} = y_{idx+s_w+val_w}$, $y'_{idx+s_w} = y_{idx}$, $y'_{idx+s_w+val_w} = y_{idx+2s_w+val_w}$ и $y'_{idx+2s_w+val_w} = y_{idx+s_w}$, где $idx = v \cdot b_w + p \cdot ps_w + u$, $s_w = 2^{i-n'}$, $b_c = 2^{n-j-1}$, $b_w = 2^{j+1-n'}$, $sb_{ps} = 2^{j-i-1}$, $ps_w = 2^{i+1-n'}$ и $val_w = 2^{j-n'} - 2^{i-n'}$.

Вернуть y' и завершить алгоритм.

Теорема 4 (Об эквиморфном вычислении действия D_n). *Преобразование БВ $y \in E^k$, эквивалентного БФ $f \in B(n)$, согласно алгоритму 3 равносильно действию эквиморфизма $\varphi_{(c_i(i,j))}$ над ним, получаемого по теореме 1. Число операций эквиморфного вычислителя для $j < n'$ составит 4w сдвигов, 4w логических умножений, 3w логических сложений, w присвоений, а для $j \geqslant n'$ при $i < n'$ — w сдвигов, 2w логических умножений, w логических сложений, w присвоений, и при $i \geqslant n'$ — w присвоений.*

Доказательство. Согласно теореме 1 эквиморфизм $\varphi_{(c_i(i,j))}$ для типа А раскроется как $\varphi^{-1}_{(i,\,j)}\varphi^{-1}_{c_i} = \varphi_{(i,\,j)}\varphi_{c_i}$, потому что элементы c_i и (i,j) — инволюции. Согласно лемме 1.А получим $y^{\varphi_{(i,j)}\varphi_{c_i}} = \left(y^{\varphi_{(i,j)}}\right)^{\varphi_{c_i}}$. Для типа Б эквиморфизм раскроется как $\varphi_{c_i}\varphi_{(i,j)}$ и тогда по лемме 1.Б верно $y^{\varphi_{c_i}\varphi_{(i,j)}} = \left(y^{\varphi_{(i,j)}}\right)^{\varphi_{c_i}}$. Откуда заключаем, что преобразование по алгоритму 3 должно быть равносильно последовательному действию алгоритма 2 и алгоритма 1. Для доказательства этого будем опираться на доказательства теоремы 3 и теоремы 2 и совместим их шаги. Это можно выполнить двумя способами: непосредственной композицией расчетных формул шагов алгоритмов или качественным анализом перестановок символов. Далее рассматривается способ качественного анализа. Отметим, что во всех шагах алгоритмов размеры символов совпадают.

Для случая $j < n'$ за основу возьмем формулу шага 2 алгоритма 2 $y'_u = y_u \,\&\, \mathrm{st} \vee (y_u \,\&\, \mathrm{lo} \ll val_E) \vee \vee (y_u \,\&\, \mathrm{hi} \gg val_E)$. При последующем действии над y'_u согласно теореме 2 соответственные четные и нечетные символы должны быть переставлены. Неподвижные символы младших(его) и старших(его) полублоков должны дополнительно сдвинуться на размер символа в битах. Для этого слагаемое $y_u \,\&\, \mathrm{st}$ нужно заменить на $(y_u \,\&\, \mathrm{st_{lo}} \ll s_E) \vee (y_u \,\&\, \mathrm{st_{hi}} \gg s_E)$, где

$$\mathrm{st_{lo}} = \overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{1\ldots1}_{s_E}}^{sb_{ps}}}^{w_b} \text{ и } \mathrm{st_{hi}} = \overbrace{\underbrace{1\ldots1}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}\overbrace{\underbrace{0\ldots0}_{s_E}\underbrace{0\ldots0}_{s_E}}^{sb_{ps}}}^{w_b}$$

константы для выделения неподвижных символов младших(его) и старших(его) полублоков соответственно. Подвижные символы в y'_u должны так же сдвинуться на значение размера символа. Поэтому значение сдвига символов должно быть скорректировано с $2^j - 2^i$ на $val_E = 2^j$. Откуда получаем $y'_u = (y_u \,\&\, \mathrm{st_{lo}} \ll s_E) \vee (y_u \,\&\, \mathrm{st_{hi}} \gg s_E) \vee (y_u \,\&\, \mathrm{lo} \ll val_E) \vee \vee (y_u \,\&\, \mathrm{hi} \gg val_E)$. Остальные константы, включая lo и hi, соответствуют шагу 2 алгоритма 2. Всего будет затрачено (исходя из преобразования одного слова): 4*w* сдвигов, 4*w* логических умножений, 3*w* логических сложений, *w* присвоений.

Для случая $j \geqslant n'$ так же как и в теореме 3 возможно два варианта. В первом варианте $i < n'$ константы будут эквивалентны шагу 3 алгоритма 2, но расчетные формулы $y'_{idx} = y_{idx} \,\&\, \mathrm{st_{lo}} \vee \left(y_{idx+sb_w} \,\&\, \mathrm{hi} \ll s_E\right)$ и $y'_{idx+sb_w} = y_{idx+sb_w} \,\&\, \mathrm{st_{hi}} \vee (y_{idx} \,\&\, \mathrm{lo} \gg s_E)$ нужно заменить на $y'_{idx} = (y_{idx} \,\&\, \mathrm{st_{lo}} \ll s_E) \vee y_{idx+sb_w} \,\&\, \mathrm{hi}$ и $y'_{idx+sb_w} = \left(y_{idx+sb_w} \,\&\, \mathrm{st_{hi}} \gg s_E\right) \vee y_{idx} \,\&\, \mathrm{lo}$, потому что символы с четными и нечетными индексами должны быть переставлены согласно теореме 2. Неподвижные символы младших(его) полублоков(а) сдвинуться влево, старших(его) — вправо. Аналогично для подвижных символов. Отметим, что для них дополнительный сдвиг слова на символ эквивалентен его отсутствию, потому что значение сдвига есть разность $sb_w = 2^j$ минус $s_E = 2^i$. В результате значение сдвига — только sb_w. Сложность эквивалентна сложности алгоритма 2 для случая $j \geqslant n'$ и $i < n'$.

Во втором варианте $i \geqslant n'$ константы будут эквивалентны шагу 4 алгоритма 2, но расчетные формулы $y'_{idx} = y_{idx}$, $y'_{idx+s_w} = y_{idx+s_w+val_w}$, $y'_{idx+s_w+val_w} = y_{idx+s_w}$ и $y'_{idx+2s_w+val_w} = y_{idx+2s_w+val_w}$ должны быть заменены на $y'_{idx} = y_{idx+s_w+val_w}$, $y'_{idx+s_w} = y_{idx}$, $y'_{idx+s_w+val_w} = y_{idx+2s_w+val_w}$ и $y'_{idx+2s_w+val_w} = y_{idx+s_w}$, потому что символы, имеющие четные и нечетные ин-

дексы, должны быть переставлены согласно теореме 2. Применительно к этому варианту должны быть попарно переставлены исходные слова. Сложность эквивалентна сложности алгоритма 2 для случая $i \geqslant n'$. Что и требовалось доказать.

Заключение

В статье предложены три эквиморфных алгоритма вычисления действия множителей канонического представления элемента группы Джевонса, и доказана их корректность. Алгоритмы основаны на теореме об эквиморфизме [8]. Приведем оценку повышения производительности вычисления действия группы Джевонса над БФ по отношению к вычислению результатов действия отдельных значений БФ. Отметим, что одновременное число обрабатываемых значений БФ не меньше $2^{n'}$, что для реальных процессоров составит 32 и 64 [6], и для некоторых современных расширений процессоров, например SSE 3.0, может достигать 512 бит, т. е. $n' = 9$. Для практического применения предложенных алгоритмов наибольший интерес представляет их апробация именно на этих процессорах. В частности, для $n = 23$ (обработка 1 мебибайта данных) производительность вычисления результатов действия элемента группы Джевонса над БФ по данным многих миллионов экспериментов повышена в 750 раз.

Это объясняется тем, что вычисления каждого значения БФ по формуле (6) сводятся к следующей цепочке: вычисление индекса слова исходного значения по аргументу БФ, извлечение этого слова из оперативной памяти, извлечение значения БФ из этого слова, вычисление нового значения аргумента БФ, вычисление индекса слова результирующего значения, извлечение результирующего слова из оперативной памяти, изменение конкретного значения БФ в нем и установка этого нового слова в оперативную память. Для эквиморфных алгоритмов цепочка вычислений короче: подготовка констант алгоритмов (выполняется единожды и этим можно пренебречь), извлечение не более четырех слов последовательно (линейным увеличением индексов) из оперативной памяти, их обработка внутри вычислителя и установка результата обработки в оперативную память.

Список литературы

1. **Логачев О. А., Сальников А. А., Ященко В. В.** Булевы функции в теории кодирования и криптологии. М.: МЦМНО, 2004. 470 с.
2. **Глухов М. М., Ремизов А. Б., Шапошников В. А.** Обзор по теории *k*-значных функций. Часть 1. Справочное пособие. Ред. Н. Р. Емельянов. Заказ № 163ф. М.: Типография в/ч 33965, 1988. 153 с.
3. **Golomb S. W.** On classification of Boolean functions // IRE, Trans. circuit theory, N6, Spec. Suppl., 1959. P. 176—186.
4. **Якубайтис Э. А.** Субклассы и классы булевых функций // Автоматика и вычислительная техника. 1974. N 1. С. 1—8.
5. **Кукарцев А. М., Кузнецов А. А.** О применении частотного анализа для решения некоторых групповых уравнений индукции действия группы Джевонса и ее подгрупп на множестве булевых функций // Дискретные модели в теории управляющих систем: IX Международная конференция, Москва и Подмосковье, 20—22 мая 2015 г.: Труды / Отв. ред. В. Б. Алексеев, Д. С. Романов, Б. Р. Данилов, М.: МАКС Пресс, 2015. С. 136—138.
6. **Intel® 64** and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture, Order Number: 253665-060US September 2016. 2016, 482 p.
7. **Кукарцев А. М.** О частотных свойствах действий группы Джевонса на булевых функциях // Программная инженерия. 2016. № 11. С. 515—521.
8. **Кукарцев А. М., Кузнецов А. А.** О действиях группы Джевонса на множествах бинарных векторов и булевых функций для инженерно-технических решений обработки информации // Программная инженерия. 2016. № 1 С. 29—36.
9. **Марченко С. С.** Замкнутые классы булевых функций. М.: ФИЗМАТЛИТ, 2000. 128 с.
10. **Супруненко Д. А.** Группы подстановок. Мн.: Навука і тэхніка, 1996. 366 с.
11. **Каргаполов М. И., Мерзляков Ю. И.** Основы теории групп. 3-е изд., перераб. и доп. М.: Наука, 1982. 288 с.
12. **Кукарцев А. М., Кузнецов А. А.** О конструктивном представлении группы Джевонса для инженерно-технических решений обработки информации // Программная инженерия. 2015. № 11. С. 25—33.

On the Equimorphic Computation of the Jevons Group Element Action over Boolean Functions

A. M. Kukartsev, amkukarcev@yandex.ru, **A. A. Kuznetsov,** kuznetsov@sibsau.ru, Siberian State Aerospace University named after academician M. F. Reshetnev, Krasnoyarsk, 660014, Russian Federation

Corresponding author:

Kukartsev Anatolii M., Senior Lecturer, Siberian State Aerospace University named after academician M. F. Reshetnev, Krasnoyarsk, 660014, Russian Federation
E-mail: amkukarcev@yandex.ru

Received on March 26, 2017
Accepted on May 10, 2017

The solution of the equation in which an element of the Jevons group acts over a Boolean function reduces to the calculation of factor actions of the canonical representation of this element. According to average estimates a computer

with a clock speed of 1 GHz processes (not including overhead) 1 MiB of data (equivalent to the Boolean function of 23 arguments) in approximately 30 minutes. This time is unacceptably large for many tasks of information processing.

In this paper, we propose a model of an equimorphic calculator to solve the equation. It is based on three algorithms for three types of factors of the Jevons group canonical representation. Algorithms include elementary logical operations performed on machine words: logical multiplication, logical addition, right and left logical shifts, and assignment. These operations are applied to partitioning of the machine words on the equivalent to Boolean functions data. As a result, the performance of computations increases in proportion to the size of the machine word in compare to the processing of single values of the function. Moreover the machine words can be processed in a parallel way. So we can improve the performance of calculations by increasing the number of computing cores. Eventually, the performance of computations is increased by hundreds and thousands of times in relation to algorithms for processing individual values of the function.

The algorithms are formulated for an abstract arithmetic logic device which corresponds to both IA-32 and IA-64/AMD64 architectures of computer processors and to other architectures. For the possibility of working with different capacities, the necessary recommendations and calculation formulas for forming constants for a particular architecture are given. The article contains proofs of the correctness of the algorithms. Also we estimate their complexity. The algorithms can be used for both solving the equations of action of the Jevons group over Boolean functions and implementing such actions themselves.

Keywords: *action on the set, the Jevons group, Boolean functions, equations of action on a set, equimorphic groups, abstract machines.*

Acknowledgements: *The reported study was funded by RFBR and the government of Krasnoyarskiy kray according to the research project № 17-47-240318.*

For citation:

Kukartsev A. M., Kuznetsov A. A. On the Equimorphic Computation of the Jevons Group Element Action over Boolean Functions*, Programmnaya Ingeneria,* 2017, vol. 8, no. 7, pp. 300—309.

DOI: 10.17587/prin.8.300-309

References

1. **Logachjov O. A., Sal'nikov A. A., Jashhenko V. V.** *Bulevy funkcii v teorii kodirovanija i kriptologii* (Boolean functions in coding theory and cryptology), Moscow, MCMNO, 2004, 470 p. (in Russian).

2. **Gluhov M. M., Remizov A. B., Shaposhnikov V. A.** *Obzor po teorii k-znachnyh funkcij. Chast' 1. Spravochnoe posobie / Red. N. R. Emel'janov. Zakaz № 163f* (Review on the Theory of k-valued functions. Part 1. A Reference Guide), Moscow, Tipografija v/ch 33965, 1988, 153 p. (in Russian).

3. **Golomb S. W.** On classification of Boolean functions, *IRE, Trans. circuit theory, Spec. Suppl.*, 1959, no. 6, pp. 176—186.

4. **Jakubajtis Je. A.** Subklassy i klassy bulevyh funkcij (Sub class and the class of Boolean functions), *Avtomatika i vychislitel'naja tehnika*, 1974, no. 1, pp. 1—8 (in Russian).

5. **Kukartsev A. M., Kuznetsov A. A.** O primenenii chastotnogo analiza dlja reshenija nekotoryh gruppovyh uravnenij indukcii dejstvija gruppy Dzhevonsa i ejo podgrupp na mnozhestve bulevyh funkcij (The use of frequency analysis to solve some group induction equations action Jevons group and its subgroups on the set of Boolean functions), *Diskretnye modeli v teorii upravljajushhih sistem: IX Mezhdunarodnaja konferencija,* Moscow, Podmoskov'e, 20—22 May 2015, Trudy, Moscow, 2015, pp. 136—138 (in Russian).

6. **Intel®** 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture, Order Number: 253665-060US September 2016, 2016, 482 p.

7. **Kukartsev A. M.** O chastotnyh svojstvah dejstvij gruppy Dzhevonsa na bulevyh funkcijah (On Frequency Characteristics Jevons Group Action on Boolean functions), *Programmnaya Ingeneria*, 2016, vol. 7, no. 11, pp. 515—521 (in Russian).

8. **Kukartsev A. M., Kuznetsov A. A.** O dejstvijah gruppy Dzhevonsa na mnozhestvah binarnyh vektorov i bulevyh funkcij dlja inzhenerno-tehnicheskih reshenij obrabotki informacii (About Actions of the Jevons Group on Sets of Binary Vectors and Boolean Functions for Engineering Solutions of Information Processing), *Programmnaya Ingeneria*, 2016, vol. 7, no. 1, pp. 29—36 (in Russian).

9. **Marchenko S. S.** *Zamknutye klassy bulevyh funkcij* (Post's lattice), Moscow, FIZMATLIT, 2000, 128 p. (in Russian).

10. **Suprunenko D. A.** *Gruppy podstanovok* (Groups of permutations), Minsk, Navuka i tjehnika, 1996, 366 p. (in Russian).

11. **Kargapolov M. I., Merzljakov Ju. I.** *Osnovy teorii grupp. 3-e izd., pererab. i dop.* (Basics of group theory), Moscow, Nauka, 1982, 288 p. (in Russian).

12. **Kukartsev A. M., Kuznetsov A. A.** O konstruktivnom predstavlenii gruppy Dzhevonsa dlja inzhenerno-tehnicheskih reshenij obrabotki informacii (Constructive representation of the Jevons group for engineering solutions of information processing), *Programmnaya Ingeneria*, 2015, no. 11, pp. 25—33 (in Russian).

УДК 004.82 DOI: 10.17587/prin.8.310-318

М. Г. Гриф, д-р техн. наук, проф., зав. кафедрой, e-mail: grifmg@mail.ru,
А. В. Лукоянычев, магистрант, e-mail: dizzystyle@yandex.com,
Новосибирский государственный технический университет

3D-анимация русского жестового языка на основе нотации Димскис

Представлены и проанализированы особенности русского жестового языка, трудности компьютерного сурдоперевода и современные нотации жестового языка.

Обосновано использование нотации Димскис для демонстрации русского жестового языка и использование графического пакета Unity3D для создания модели анимированного сурдопереводчика. Описаны возможности разработанного программного комплекса для создания справочника жестового языка и перспективы его дальнейшего использования.

Ключевые слова: *русский жестовый язык, мультимедийный словарь, нотации Димскис, Unity3D, анимированный персонаж — аватар*

Введение

В Российской Федерации действует целевая программа "Доступная среда", в которой предусматриваются всевозможные средства для социальной адаптации инвалидов по слуху. Основой адаптации слабослышащих людей в современном обществе является обучение их жестовому языку (ЖЯ).

На сегодняшний день наиболее оправдано и перспективно создание мультимедийных приложений на основе 3D-компьютерного персонажа (аватара), демонстрирующего элементы ЖЯ, поэтому создание информационных приложений, способных работать с ЖЯ, является одной из приоритетных задач при работе с глухими и слабослышащими людьми.

Особенности жестового языка

Жестовый язык — это язык межличностного общения неслышащих людей во всем мире, при котором способ общения строится не на звуковой, а на жестикулярно-мимической основе. В жестовых языках информация кодируется движениями рук, тела, лица, глаз и воспринимается зрительно. Этим определяются их основные отличия от звучащих языков. В ЖЯ большую роль играет пространство вокруг говорящего. Это выражается в пространственной организации самого жеста и объектов, о которых идет речь. Кроме того, в ЖЯ элементы жеста выполняются и воспринимаются одновременно, в отличие от звукового языка, где произносимые звуки достигают нашего уха последовательно.

Универсального ЖЯ, единого для глухих людей всего мира, не существует. Жестуно (*Gestuno*) или панъевропейский пиджинизированный ЖЯ являются искусственными и предназначены для облегчения общения глухих участников международных мероприятий [1]. Жестуно не является родным языком какой-либо группы слабослышащих людей. В мире зафиксировано более 120 национальных ЖЯ.

Русский жестовый язык (РЖЯ) также обладает малой степенью стандартизации. Существует несколько диалектов этого языка. Прежде всего, выделяют петербургский и московский диалекты РЖЯ (различия между ними могут доходить до 30...40 % жестов-слов [2]). Диалекты РЖЯ используются во многих районах РФ, а также в Белоруссии, Казахстане, Украине. Соответственно, возникает проблема выбора тех диалектов, на которые будет ориентирован перевод [3].

Жестовый язык объединяет в себе несколько разновидностей жестовой коммуникации. Выделяют "ручную азбуку" (дактильную азбуку), использующуюся для жестовой передачи букв алфавита, и непосредственно жестовую речь, ориентированную на передачу не букв, а слов, языковых конструкций и семантических концептов.

Дактильная азбука применяется во всех жестовых языках как средство передачи незнакомых слов и для передачи различных имен собственных, аббревиатур и редко встречающихся слов. Она тесно связана с вербальным языком и письменностью.

Основным же способом межчеловеческой коммуникации в среде глухих является собственно ЖЯ, в котором каждому смысловому понятию (или группе понятий) соответствует определенный уникальный жестовый эквивалент.

Система жестового общения глухих имеет сложную структуру, включает две разновидности жестовой речи — национальную и калькирующую [3].

Национальная жестовая речь — это общение с помощью средств ЖЯ, которое имеет самобытную лингвистическую систему, обладающую своеобраз-

ной лексикой и грамматикой. Калькирующая жестовая речь калькирует лингвистическую структуру словесного языка. Калькирующая жестовая речь — вторичная знаковая система, которая усваивается на базе и в процессе изучения глухим ребенком словесной речи. Жесты здесь являются эквивалентами слов, а порядок их следования такой же, как в обычном предложении [4]. В разных регионах страны существуют несколько различающихся, но использующих практически единую дактильную азбуку диалектов РЖЯ.

Особенностями русской дактильной азбуки, в отличие от других алфавитов ЖЯ мира, являются следующие положения:

— все буквы русской азбуки показываются одной рукой (в отличие, например, от британской, где пользуются двумя руками);

— ряд букв русской азбуки отображается посредством динамических жестов (в то время как жесты британской азбуки — статические);

— конфигурации кисти и пальцев рук в русском дактиле более сложные.

На настоящее время грамматика РЖЯ еще недостаточно изучена и формализована, чтобы вести разговоры об автоматическом сурдопереводе из произвольного русскоязычного текста на ЖЯ. Серьезные различия в семантико-синтаксической структуре письменного и жестового языков не позволяют выполнять однозначный машинный перевод русскоязычных текстов на РЖЯ. И поэтому действующих автоматических систем сурдоперевода на данный момент не существует [5]. Для создания такой полноценной модели необходимо проводить глубокий семантический анализ и разбор письменных фраз, а это пока возможно лишь на поверхностном уровне в силу несовершенства алгоритмов и баз знаний [6].

Однако калькирующая жестовая речь напрямую отражает разговорную звучащую речь, поэтому компьютерный синтез калькирующей жестовой речи намного проще. Примером компьютерной системы распознавания перевода разговорной речи (английской) в калькирующую жестовую речь с элементами ЖЯ может служить американская разработка iCommunicator [7], лексикон которой состоит более чем из девяти тысяч видеофрагментов жестов.

Автоматические системы сурдоперевода развиваются в двух направлениях: перевод разговорного языка в ЖЯ и перевод с ЖЯ в разговорный. Первое направление предназначено, в первую очередь, для обучения ЖЯ и для научных исследований. Второе направление имеет большую практическую составляющую, но также связано с научными лингвистическими исследованиями этой проблемы. В связи с малой лингвистической проработанностью данного направления и наличием множества диалектов ЖЯ основные усилия пока направлены на первое направление для реализации дактильной и калькирующей речи ЖЯ. В любом случае оба эти направления опираются на систему нотаций и словарь ЖЯ.

Особенность человеко-машинного взаимодействия состоит в том, что ЖЯ и жестовый словарь должны быть определенным образом записаны, чтобы компьютер мог обрабатывать и синтезировать жесты. Для описания жеста по его признакам существует несколько различных систем нотации (жестовой транскрипции), позволяющих зафиксировать представление жеста в записи.

Современные системы нотации жестового языка

Современные исследования жестовых языков начались в 1960-х гг. с трудов У. Стокоу (Stokoe) [8], в которых он впервые на материале американского жестового языка (*American sign language*, *ASL*, Амслен) описал письменную систему кодификации для многоуровневой записи жестов. Стокоу показал, что жестовые языки являются естественными человеческими языками, первым указал на сходство звучащих и жестовых языков, несмотря на различие в канале передачи информации.

Нотация Стокоу опирается на латинский алфавит. В качестве базовой смысловой единицы в ЖЯ служит жест (визуально-кинетический акт), в котором участвуют в первую очередь руки, а часто также мимика лица и артикуляция губ. Кинетическая природа жеста и его визуальное восприятие обусловливают особенности ЖЯ. К их числу относятся: возможность определенным образом расположить жест в пространстве, исполнить одновременно два жеста двумя руками и т. д. Лексические жестовые элементы непосредственно и в ЖЯ, и в дактильной азбуке формируются практически одинаково.

Такое описание, основанное на выделении единиц жеста, У. Стокоу назвал "хиремы" (от греческого χειρ, χειρός — "рука"). Особенностью структуры жеста является взаимосвязь между его компонентами: все они воплощаются в жесте одновременно. В системе нотации Стокоу используется 55 символов.

Каждый жест складывается из хирем. Хиремы подразделяются на три класса — параметра, определяющих структуру жеста.

- К первому классу (параметру) относятся **табы** (12 символов), указывающие *место исполнения жеста*. Внутри данного класса можно выделить три группы значений, т. е. жест может выполняться:

— в нейтральном жестовом пространстве;

— на уровне какой-либо части тела, но при этом рука, выполняющая жест, не касается тела;

— в контакте с какой-либо частью тела (рука, выполняющая жест, касается тела говорящего).

- Ко второму классу относятся **дезы** (19 символов), указывающие на *положение кисти руки*.
- К третьему классу относятся **сиги** (24 символа), описывающие *траекторию движения руки*. При этом учитывается как перемещение руки из одной точки пространства в другую, так и "мелкие" движения пальцев или кисти руки, в то время как положение руки в пространстве остается неизменным. Выстроенные

вертикально, они показывают отдельные движения, совершаемые синхронно, последовательные действия записываются слева направо.

Впоследствии был введен четвертый параметр структуры жеста — *относительная ориентация рук в пространстве* — друг к другу и к корпусу говорящего [9].

Стокоу определил порядок символов при записи жеста: сначала указывается место выполнения жеста, затем форма руки, и, наконец, характер движения руки. Он разработал для ASL систему записи жестов как последовательности таба, деза и сига (рис. 1). Дополнительно используемые символы, записываемые снизу и сверху основного символа, — "субскрипты" и "прескрипты", соответственно, показывают важные различия в активном элементе, т. е. в ориентации или положении руки.

Для обозначения места выполнения жеста и характера движения руки использовались иконические (условные) значки, например, символ **[ ]** обозначал туловище (пространство от плеч до бедер), символом > изображалось движение вправо. Для обозначения формы руки Стокоу использовал цифры (например, символ **5** обозначал раскрытую ладонь с растопыренными пальцами) и буквы английского алфавита в соответствии с американской дактильной азбукой. Стокоу использовал эту систему в словаре "A Dictionary of American Sign Language on Linguistic Principles" [11], в котором место жеста определялось по собственной форме жеста (отраженной в транскрипции), а не по переводу жеста на английский язык.

В качестве недостатков нотации Стокоу отмечается, что она алфавитная, каждая хирема, ориентация и позиция обозначаются буквой с диакритическими знаками. Кроме того, нотация не позволяет фиксировать выражение лица, которое является неотъемлемой составляющей многих жестов. Нотация Стокоу фонематична и позволяет записать только те значения параметров жеста, которые Стокоу считал смыслоразличительными для американского ЖЯ. Нотация более подходит для записи отдельных слов, а не предложений.

Таким образом, данная нотация, с одной стороны, не позволяет описать такие параметры выполнения жеста, как, например, резкость/плавность и амплитуда движения, напряженность/расслабленность руки, которые, как показали дальнейшие исследования жестовых языков, вносят существенный вклад в смысл жеста [10]. С другой стороны, данную нотацию нельзя

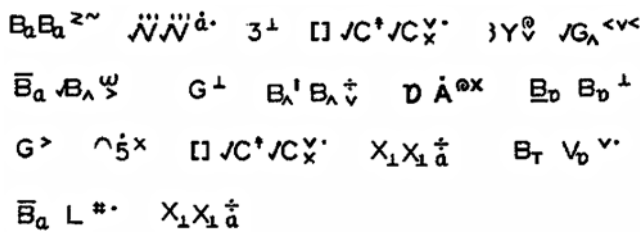

Рис. 1. Примеры обозначения жестов в нотации У. Стокоу [10]

без изменений применять и для транскрибирования других жестовых языков. Модифицированная версия нотации Стокоу используется, например, в словаре британского ЖЯ [12].

В современных исследованиях применяется не исходная версия данной транскрипционной системы, а ее различные модификации, поэтому, говоря о нотации Стокоу, подразумевают целое семейство родственных транскрипционных систем. Основное применение нотация Стокоу нашла среди лингвистов и других ученых, занимающихся изучением ЖЯ.

Предложенная запись плохо подходит для образовательных целей, так как совершенно не наглядна, поэтому разрабатываются другие системы нотации ЖЯ, использующие символы, более наглядно демонстрирующие положения рук и их перемещения.

Система транскрипции Berkeley (Беркли) (BTS) [13] была разработана для лингвистического анализа жестовых языков на уровне смысловых компонентов.

В основе BTS лежит принцип выделения в жесте "значимых элементов", ее можно сопоставить с глоссированием. Одни и те же элементы жестовой структуры могут в зависимости от смысла жеста получать разное содержательное наполнение. Авторы стремились учесть лингвистические и коммуникативные элементы.

Жест в транскрипции представляет собой последовательность элементов, снабженных индексами: сначала записывается форма пассивной руки, потом форма активной руки, движение руки, ориентация руки. Например, глагол "сидеть на" в транскрипции Беркли выглядит следующим образом:

—pm'PL_VL—pm'TL—gol'PL_VL_TOP—pst'STR,

где

- **pm'PL_VL** (plane showing vertical length) — пассивная рука держится вертикально, с плоской ладонью, пальцы вытянуты вперед;
- **pm'TL** (twolegged animate being) — активная рука в перевернутой позиции;
- **gol'PL_VL_TOP** (move to top of vertical plane) — активная рука движется к верхней части пассивной руки;
- **pst'STR** (posture straddle) — накрыть (оседлать) руку.

Этот глагол может относиться к целому ряду событий, таких как "ковбой верхом на коне" или "мальчик сидит на заборе". Приведенное выражение может быть представлено как глагол с четырьмя смысловыми компонентами ("морфемами"), что показывается четырьмя тире.

Лингвистический статус каждого компонента задается в нижнем регистре (**pm, gol, pst**). Буквы верхнего регистра указывают на смысловое содержание каждого компонента.

Система BTS была разработана для транскрибирования жестового дискурса, поэтому нотация включает в себя символы для обозначения дискурсивного поведения говорящего — различных выражений лица, интенсивности выполнения жестов,

пауз, ошибок в выполнении жестов и т. п. Система BTS связана с морфосинтаксическими, семантическими и прагматическими аспектами языка.

В академической среде чаще всего пользуются системой транскрипции жестовых языков HamNoSys, созданной в Гамбургском университете (основной разработчик — Т. Ханке) [14]. Она основана на нотации Стокоу, однако число символов увеличено до 200, чтобы передавать жесты любого ЖЯ. Фонологические особенности обычно передаются отдельными символами, а группы особенностей, дополняющих жест, обозначаются одним символом все в совокупности. Система HamNoSys ориентирована на очень подробное описание жеста.

В системе HamNoSys [15] задаются форма кисти, ее ориентация по двум параметрам (направление пальцев и разворот кисти), место, где располагается рука во время жеста, и характер движения. Жесты могут быть одноручными и двуручными. Для двуручных жестов в запись добавляются символы, позволяющие совместить действия обеих рук в рамках одной записи (рис. 2).

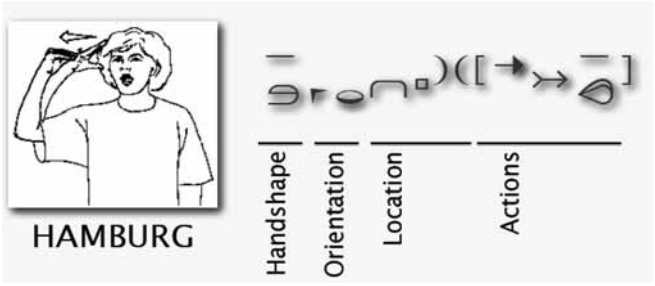

Рис. 2. Структура записи в нотации HamNoSys [14]

На рис. 3 показано несколько примеров кодирования конфигурации пальцев и кисти руки, а на рис. 4 — кодирование ориентации ладони и руки, положения и движения.

Для особого положения отдельных пальцев используется численное обозначение от единицы до пяти: 1 — большой палец, 5 — мизинец. Для скрещивания пальцев используются дополнительные обозначения, соответствующие последовательности: нижний палец, точка касания, верхний палец. Например,  — второй палец касается третьего на ногте.

В системе HamNoSys символами обозначаются не только место выполнения жеста, форма и траектория движения руки, ориентация руки, но и немануальные жесты (мимика и глаза говорящего). Набор символов постоянно расширяется и совершенствуется разработчиками, чтобы соответствовать новой лексике, постоянно появляющейся в ЖЯ.

Жесты в транскрипции HamNoSys довольно сложны для визуального восприятия, однако эта система записи подходит для компьютерной обработки. Транскрипция HamNoSys использовалась в многочисленных тематических словарях немецкого ЖЯ, разрабатываемых в Гамбурге, в словарях австралийского ЖЯ, новозеландского ЖЯ и др.

Одна из самых известных систем нотации, широко распространенная в Америке (в частности, для Амслена), называется SignWriting [16] и разработана В. Саттон в 1974 г. Система SignWriting — не фонемная орфография, и между жестами и записью нет 100 %-ного соответствия.

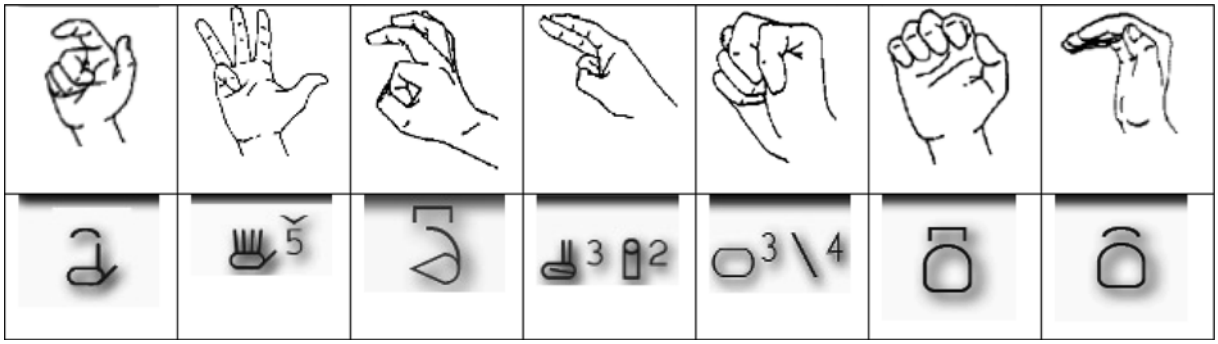

Рис. 3. Примеры записи конфигурации пальцев и кисти руки в системе HamNoSys [14]

				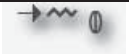	
Ладонь вверх от тела	От тела, вниз и вправо	На уровне плеч с правой стороны	Локоть с правой стороны	Волновая и зигзагообразная линия движения	Круговые движения и производные формы

Рис. 4. Примеры записи ориентации ладони и руки, положения и движения (действия) в системе HamNoSys [14]

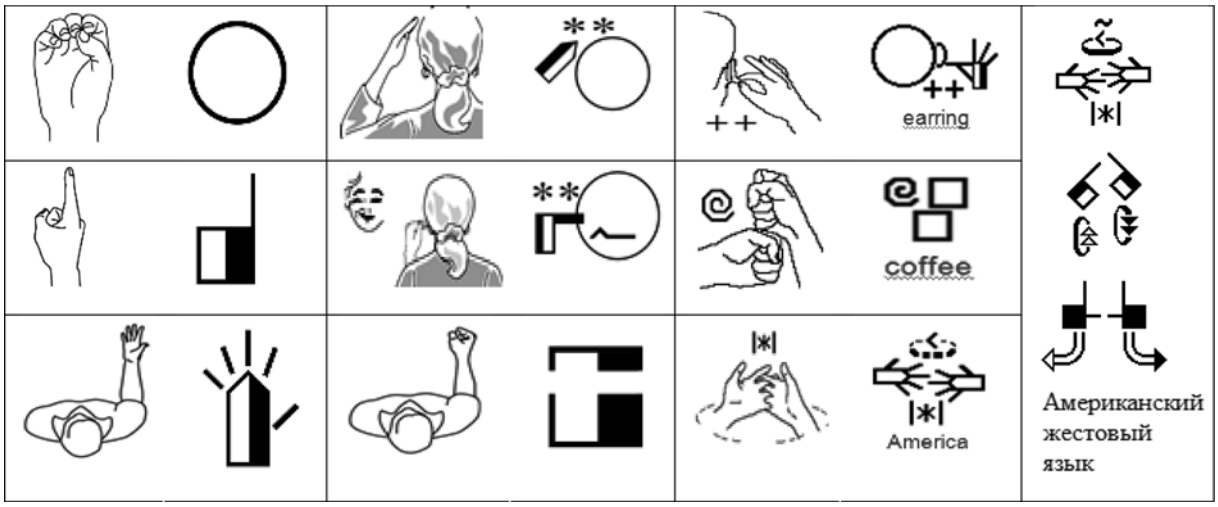

Рис. 5. Примеры записей в нотации SignWriting [17]

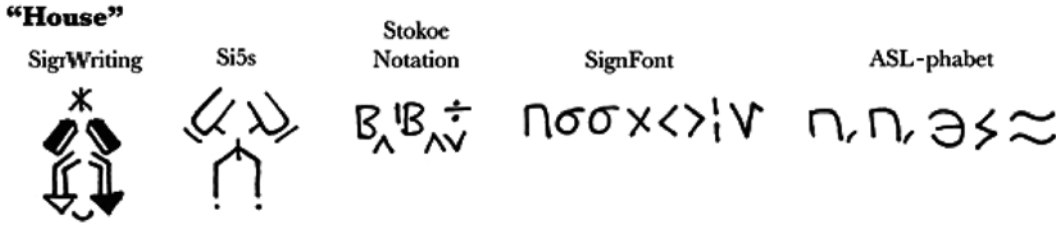

Рис. 6. Пример записи слова "Дом" в разных нотациях [18]

В системе SignWriting жест не записывается как линейная последовательность символов, а изображается пиктограммой, включающей иконические изображения рук, их движений (с помощью разнообразных стрелок), их расположения относительно друг друга и говорящего, при необходимости передается выражение лица или направление взгляда, сопутствующие жесту. Цепочка жестов записывается сверху вниз. Транскрипция позволяет точно и наглядно отображать жесты, при этом остается простой для восприятия (рис. 5).

Система SignWriting используется в некоторых школах для глухих и взрослыми носителями, а также как система нотации в лингвистических исследованиях и применяется для создания словарей жестовых языков.

В системе SignWriting имеется более 500 исходных форм, что затрудняет ее использование на практике.

Кроме перечисленных, существуют и другие менее известные системы записи ASL: Sign Script, Si5s, ASL-phabet. На рис. 6 приведена запись жеста "Дом" с использованием этих нотаций в сравнении с SignWriting и нотацией Стокоу. На рис. 7 приведена запись жеста "Дом" в нотации Димскис.

Рис. 7. Пример записи слова "Дом" в нотации Димскис [19]

Нотация Димскис

Для учета особенностей РЖЯ используется система нотации Л. Димскис [19]. В этой нотации выделяется более 30 конфигураций рук, около 50 характеристик места исполнения жеста и более 70 характеристик локализации. В нотации Димскис структура жеста состоит из отдельных элементов и имеет следующие постоянные характеристики:

1) форма руки (рук);

2) место расположения жеста (жестовое пространство);

3) характер движения.

Первый элемент — форма руки, он предполагает строго определенное положение ладоней и направление пальцев. Он состоит из трех частей: форма руки, положение ладони, направление пальцев.

Второй элемент структуры — место расположения жеста в пространстве. Жест может исполняться над головой, у лица, на уровне плеч, шеи и т. д. Локализация каждого жеста строго постоянна и изменение жестового пространства влияет на смысловое значение жеста в РЖЯ.

По характеру исполнения жесты подразделяются на одноручные, на двуручные с одинаковой формой рук, на двуручные с разной формой рук.

Особенностью нотации является использование иконографического изображения формы руки, это значительно уменьшает число символов для опи-

Рис. 8. Пример записи в нотации Димскис слова "Август" [19]

сания конфигурации пальцев. На рис. 8 приведен пример кодирования в нотации Димскис.

В этом примере:

— первый знак указывает на то, что жест выполняется правой рукой;

— второй знак иконографически указывает форму руки — все пальцы собраны в кулак, большой выпрямлен, прижат к согнутому указательному;

— третий символ схематически изображает разворот ладони — она направлена от себя, от корпуса говорящего;

— четвертый знак указывает на ориентацию ладони — пальцами вверх;

— пятый символ означает, что жест исполняется на уровне груди человека.

— в конце записи добавлен элемент, описывающий кинетику жеста: в конце исполнения жеста кисть руки делает обратимое горизонтальное движение.

В России на настоящее время не существует законченных систем транскрибирования и формализации РЖЯ [6]. Немногочисленные работы, которые посвящены компьютерному сурдопереводу, используют нотационную запись HamNoSys [20, 6]. Хотя и нотация Димскис имеет ряд преимуществ. Во-первых, она единственная, которая полностью создавалась для РЖЯ и, соответственно, учитывает все его особенности. Во-вторых, это единственная нотация, которая представляет конфигурацию руки (пальцев) в виде "фотографической" иконографической записи. На современном этапе развития компьютерных 3D-технологий создание такого изображения не представляет большой сложности. Но такая запись значительно уменьшает число символов для представления конфигурации руки и повышает наглядность. Базовый набор символов в нотации небольшой и интуитивно понятный, что позволяет упростить и облегчить создание записей в нотации Димскис. Отсутствие возможности описания немануальных действий (рта, глаз), взаимодействия с окружающей обстановкой свидетельствует о том, что, как любая развивающаяся система, система нотаций Димскис должна совершенствоваться и улучшаться. Применение нотации Димскис для демонстрации РЖЯ на сегодняшний день является обоснованным решением.

Управление 3D-анимационным персонажем

Для разработки и управления анимированным 3D-персонажем наиболее развитыми и популярными графическими пакетами (игровыми движками) являются Unity3D, UDK и CryENGINE. Эти пакеты являются самыми мощными игровыми движками, с характерными особенностями.

Для выбора движка должны учитываться следующие основные показатели: возможность качественного отображения и управления 3D-персонажем, кроссплатформенность приложения, возможность загрузки 3D-модели аватара из других графических редакторов, дополнительные возможности движков и особенности работы с ними.

Сравнение характеристик этих трех графических пакетов позволяет выделить их наиболее сильные стороны: Unity3D больше подходит для мобильных 3D-приложений; UDK, обладая уникальным скриптовым языком программирования, ориентирован на сложные действия и сюжеты; CryENGIN, адаптированный под платформы следующего поколения, имеет наилучшие графические возможности.

Для реализации анимированного персонажа наиболее подходит Unity3D как наиболее документируемый и хорошо поддерживаемый пакет, который легко интегрируется с практически любым 3D-редактором, и главное — это мультиплатформенный графический движок. Он дает возможность сборки приложения под несколько популярных операционных систем, как на мобильных устройствах, так и на персональных компьютерах.

Программный комплекс для создания и демонстрации РЖЯ

Целью работы, результаты которой представлены в данной статье, является разработка простого в использовании, расширяемого по возможностям, мультиплатформенного 3D-анимационного программного комплекса для создания и демонстрации РЖЯ.

Для создания словаря РЖЯ использовалась система нотаций Димскис и графический пакет Unity3D. С помощью Unity3D реализована 3D-модель сурдопереводчика (аватара).

В Unity3D сложный анимационный персонаж описывается и управляется по частям. Это предполагает древовидную структуру скелета аватара, при этом каждым подвижным элементом можно управлять отдельно.

Например, движение плеча (части руки до локтевого сустава) невозможно без движения ключицы (в плечевом суставе), таким образом, приходится создавать наборы простейших анимаций для создания движения элементов руки. Для описания иконографической формы руки требуется разработка достаточно сложного сценария, так как при этом описании необходимо управление пятью пальцами, которые состоят из запястья, пястья и фаланг пальцев. Для реализации данной концепции разработан программный комплекс, который имеет два режима: "Разработчик" и "Пользователь".

При запуске программы загружается файл описания сцены с аватаром, находящимся в исходном положении (рис. 9, *а*, см. третью сторону обложки). В режиме "Разработчик" доступны для отображения все созданные анимационные файлы. Разработчик имеет возможность создавать сценарии, используя всю иерархию библиотеки от простейших до составных анимаций. При этом возможно увеличение ава-

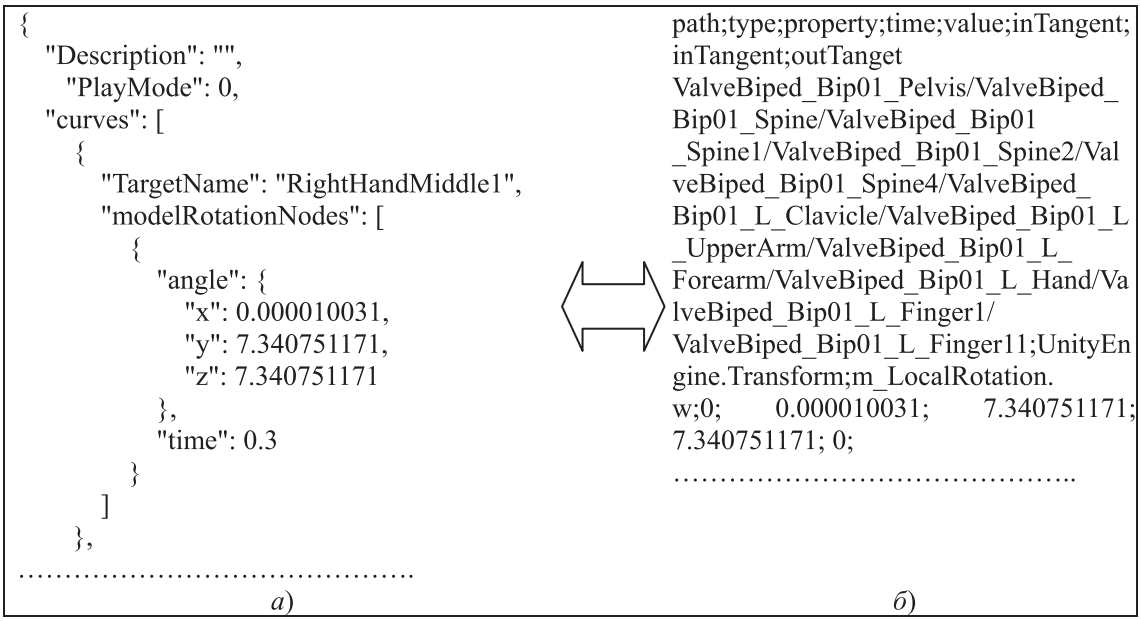

Рис. 10. Пример скриптового (*а*) и анимационного (*б*) файлов движения аватара

тара для точного установления координат отдельных элементов (рис. 9, *б*, см. третью сторону обложки), что требуется при создании формы руки. В данном режиме осуществляется регулирование скорости выполнения клипа, его просмотр и редактирование.

Разработано программное обеспечение, которое позволяет переводить созданный в Unity3D анимационный клип в скриптовый текстовый формат и наоборот — текстовый формат такого описания в анимационное движение. Создание скриптового языка позволило упростить понимание отображения движений аватара, оперативно корректировать файл и сократило объем файлов клипов, что особенно актуально для мобильной реализации. Пример данного скриптового файла и файла в формате Unity3D приведен на рис. 10.

В соответствии с анатомическим строением руки человека число подвижных суставов составляет 18 единиц. Unity3D позволяет воспроизводить движения суставов руки аватара одновременно. Параллельные процессы требуют синхронизации действий, для этого разработан параметр анимации "PlayMode". Он реализует семафорную систему синхронизации процессов. Значение "PlayMode": 1 обозначает, что требуется дождаться завершения всех активных действий, прежде чем запустить очередную анимацию. Для реализации кинетики жеста требуется, чтобы действия выполнялись в относительной системе координат, так как руки аватара необходимо вернуть в предыдущее состояние. Для этого используется параметр анимации "howToChange". При значении "howToChange": 1 координаты отсчитываются относительно предыдущего состояния.

Созданный клип записывается в библиотеку символов, при этом ставится в соответствие подходящий элемент нотации Димскис. На основе полученных клипов осуществляется разработка справочника РЖЯ (библиотека знаков) за счет сборки составных клипов. Этот уровень работы приложения доступен в режиме "Пользователь" (рис. 11, см. третью сторону обложки).

В режиме "Пользователь" набирается необходимая запись в нотации Димскис, в которую из библиотеки отбираются необходимые клипы для реализации слова или дактилемы. Сформированные записи заносятся в справочник РЖЯ и в дальнейшем могут использоваться для составления новых слов и более сложных сценариев. Использование справочника предполагается для демонстрации дактилем и слов РЖЯ, а также составных словосочетаний.

Кроме того, в нотацию Димскис добавлены два служебных символа. Первый — возврат в исходное состояние. Он автоматически добавляется в конце при демонстрации любых записей из словаря. Второй служебный символ — принудительная синхронизация (семафор). Этот символ необходим при составлении сложных сценариев и уроков. Оба этих символа доступны как в режиме "Разработчик", так и в режиме "Пользователь".

Заключение

Разработанный программный комплекс на основе нотации Димскис и Unity3D является достаточно наглядным и простым в использовании, что позволяет привлечь знакомое с РЖЯ заинтересованное сообщество к пополнению справочника через сеть Интернет и к дальнейшему составлению обучающих уроков. Кроме того, комплекс легко пополняется новыми символами, что дает возможность совершенствовать

как саму систему, так и нотации Димскис. Он ориентирован на использование как в сети Интернет, так и на мобильных устройствах и персональных компьютерах (рис. 12, см. третью сторону обложки).

Список литературы

1. **Комарова А. А.** Развитие исследований по истории глухих // Материалы Первого Моск. симпоз. по истории глухих. М.: Загрей, 1997. С. 16—25.
2. **Буркова С. И., Варинова О. А.** К вопросу о территориальном и социальном варьировании русского жестового языка // Русский жестовый язык: Первая лингвистическая конференция. М.: Буки Веди, 2012. С. 127—143.
3. **Гриф М. Г., Тимофеева М. К.** Проблема автоматизации сурдоперевода с позиции прикладной лингвистики // Сибирский филологический журнал. 2012. № 1. С. 211—219.
4. **Зайцева Г. Л.** Жестовая речь. Дактилология: Учебник для студентов высших учебных заведений. М.: Гуманит. изд. центр ВЛАДОС, 2000. 192 с.
5. **Воскресенский А. Л., Ильин С. Н., Milos Z.** О распознавании жестов языка глухих // Компьютерная лингвистика и интеллектуальные технологии: По материалам ежегодной Междунар. конф. "Диалог". Бекасово, 26—30 мая 2010 г. 2010. Вып. 9 (16). С. 76—81.
6. **Карпов А. А., Кагиров И. А.** Формализация лексикона системы компьютерного синтеза языка жестов // Труды СПИИРАН, 2011. № 1 (16). С. 123—140.
7. **iCommunicator.** URL: http://www.icommunicator.com (accessed 12.03.2017).
8. **Stokoe W.** Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf. New York: University of Buffalo, 1960. 78 p.
9. **Battinson R.** Lexical Borrowing in American Sign Language. Silver Spring: Linstok Press, 1978. 240 p.
10. **Martin J.** A linguistic comparison of two notation systems for signed languages: Stokoe Notation and Sutton SignWriting. Western Washington University. 2000. URL: http://www.signwriting.org/archive/docsl/sw0032-Stokoc-Sutton.pdf.
11. **Stokoe W., Casterline D., Croneberg C.** A dictionary of American sign language on linguistic principles. Silver Spring: Linstok Press, 1976. 346 p.
12. **Brien D.** Dictionary of British sign language. London: Faber & Faber Ltd, 1992. 1084 p.
13. **Hoiting N., Slobin D.** Transcription as a tool for understanding: The Berkeley transcription system for sign language research (BTS) // Directions in sign language acquisition. Amsterdam: John Benjamins, 2002. P. 55—75.
14. **Hanke T.** HamNoSys — Hamburg Notation System for Sign Languages. 2010. URL: https://www.sign-lang.uni-hamburg.de/dgs-korpus/files/inhalt_pdf/HamNoSys_06en.pdf.
15. **Prillwitz S., Leven R., Zienert H., Hanke T., Henning J.** HamNoSys. Version 2.0; Hamburg Notation System for sign languages: An introductory guide International Studies on Sign Language and Communication of the Deaf. Vol. 5. Hamburg: Signum Press, 1989. 45 p.
16. **Sutton V.** Lessons in SignWriting Textbook. SignWriting Press, 2014. 220 p.
17. **Sutton V.** SignWriting Basics Instruction Manual. SignWriting Press, 2014. 42 p.
18. **American Sign Language.** URL: https://en.wikipedia.org/wiki/American_Sign_Language.
19. **Димскис Л. С.** Изучаем жестовый язык: учебное пособие. М.: Академия, 2002. 128 с.
20. **Мануева Ю. С., Гриф М. Г., Козлов А. Н.** Построение системы компьютерного сурдоперевода русского языка // Труды СПИИРАН. 2014. № 6 (37). С. 170—187.

3D Animation of Russian Sign Language based on Dimskis Notation

M. G. Grif, grifmg@mail.ru, **A. V. Lukoyanychev,** dizzystyle@yandex.com,
Novosibirsk state technical University, 630073, Russian Federation

Corresponding author:
Grif Michael G., Professor, Novosibirsk state technical University, 630073, Russian Federation
E-mail: grifmg@mail.ru

Received on April 02, 2017
Accepted on May 10, 2017

The article is devoted to the features of the Russian sign language, the difficulties of computer translation. The analysis of modern notations of sign language and the possibility of their use for the development of the Russian sign language are presented. The use of Dimskis notation for demonstrating gestures is explained. The main characteristics of Dimskis notation and the features of its use in computer realization are considered. We consider the main 3D graphic packages for the implementation of the animation character. Using the modern cross-platform graphics package Unity 3D for the animated model of the sign language interpreter us allows to create an economical and efficient system. The possibilities of the software complex on the basis of Dimskis notation and an animated 3D character

using Unity 3D gestures for demonstration are described. The stages of creating the dictionary of the Russian sign language and compound phrases are considered. In conclusion, the authors present the main results of the work and the prospects for its further development. Using the Dimskis notation allows the users knowing the Russian sign language to supplement the glossary of terms to use the dictionary for demonstrating words and complex phrases. The proposed approach makes it possible to create "lessons" on specific topics for training.

Keywords: *Russian Sign Language, a multimedia dictionary, Dimskis notation, Unity3D, animated character, avatar*

For citation:

Grif M. G., Lukoyanychev A. V. 3D Animation of Russian Sign Language based on Dimskis Notation, *Programmnaya Ingeneria,* 2017, vol. 8, no. 7, pp. 310—318.

DOI: 10.17587/prin.8.310-318

References

1. **Komarova A. A.** Razvitie issledovaniy po istorii gluhih (Development of Deaf History Studies), *Materials of the 1 Moscow symposium on the history of the Deaf*, Moscow, 1997, pp. 16—25 (in Russian).

2. **Burkova S. I., Varinova O. A.** K voprosu o territorialnom i socialnom varirovanii russkogo zhestovogo yazyka **(**On the issue of territorial and social variation of the Russian sign language), *Russian Sign Language: The First Linguistic Conference*, Moscow, Buki Vedi, 2012, pp. 127—143 (in Russian).

3. **Grif M. G., Timofeeva M. K.** Problema avtomatizatsii surdoperevoda s pozitsii prikladnoi lingvistiki (The problem of automation of sign language from the perspective of applied linguistics), *Sibirskii filologicheskii zhurnal*, 2012, no. 1, pp. 211—219 (in Russian).

4. **Zayceva G. L.** *Zhestovaya rech. Daktilogiya* (Sign speech. Dactylology): Textbook for students of higher educational institutions, Moscow, Humanite. Ed. Center VLADOS, 2000, 192 p. (in Russian).

5. **Voskresensriy A. L., Ilin S. N., Milos Z.** O raspoznavanii zhestov yazuka gluhih (About recognition of sign language gestures), *Kompyuternaya lingvistika i intellektualnye tehnologii: po materialam ezhegodnoy mezhdunarodnoy konferencii "Dialog"*, 26—30 May 2010, Bekasovo, 2010, is. 9 (16), pp. 76—81 (in Russian).

6. **Karpov A. A., Kagirov I. A.** Formalizaciya leksikona sistemy kompyuternogo sinteza yazuka zhestov (The formalization of the lexicon of computer synthesis of sign language system), *Trudy SPIIRAN*, 2011, is. 1 (16), pp. 123—140 (in Russian).

7. **iCommunicator**, available at: http://www.icommunicator.com

8. **Stokoe W**. *Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf*, New York: University of Buffalo, 1960, 78 p.

9. **Battinson R**. *Lexical Borrowing in American Sign Language,* Silver Spring: Linstok Press, 1978, 240 p.

10. **Martin J.** *A linguistic comparison of two notation systems for signed languages: Stokoe Notation and Sutton SignWriting*, Western Washington University, 2000, available at: http://www.signwriting.org/archive/docsl/sw0032-Stokoc-Sutton.pdf

11. **Stokoe W., Casterline D., Croneberg C.** *A dictionary of American sign language on linguistic principles*, Silver Spring.: Linstok Press, 1976, 346 p.

12. **Brien D.** *Dictionary of British sign language*, London: Faber & Faber Ltd, 1992, 1084 p.

13. **Hoiting N., Slobin D.** Transcription as a tool for understanding: The Berkeley transcription system for sign language research (BTS), *Directions in sign language acquisition*, Amsterdam: John Benjamins, 2002, pp. 55—75.

14. **Hanke T.** *HamNoSys — Hamburg Notation System for Sign Languages*, 2010, available at: https://www.sign-lang.uni-hamburg.de/dgs-korpus/files/inhalt_pdf/HamNoSys_06en.pdf

15. **Prillwitz S., Leven R., Zienert H., Hanke T., Henning J.** *HamNoSys. Version 2.0; Hamburg Notation System for sign languages*: An introductory guide, International Studies on Sign Language and Communication of the Deaf, vol. 5, Hamburg: Signum Press, 1989, 45 p.

16. **Sutton V.** *Lessons in SignWriting Textbook,* SignWriting Press, 2014, 220 p.

17. **Sutton V.** *SignWriting Basics Instruction Manual*, SignWriting Press, 2014, 42 p.

18. **American Sign Language,** available at: https://en.wikipedia.org/wiki/American_Sign_Language.

19. **Dimskis L. S.** *Izuchaem zhestovyy yazyk:* uchebnoe posobie (The study of sign language: study guide), Moscow, Academy, 2002, 128 p. (in Russian).

20. **Manueva J. S., Grif M. G., Kozlov A. N.** Postroenie sistemy kompyuternogo surdoperevoda russkogo yazyka **(**Computer Sign Language Interpretation System Development of Russian Language), *Trudy SPIIRAN*, 2014, no. 6 (37), pp. 170—187 (in Russian).

УДК 004.82 DOI: 10.17587/prin.8.319-327

К. И. Костенко, канд. физ.-мат. наук, зав. каф., e-mail: kostenko@kubsu.ru,
Кубанский государственный университет, г. Краснодар

О синтезе реализаций когнитивных целей для задач управления содержанием областей знаний

Определена унифицированная система структурных компонентов для моделирования процессов развития содержания областей знаний, основанного на представлении содержания понятий структурами отношений с другими понятиями. Приведена система классов объектов и отношений между классами, представляемая когнитивной картой и аккумулирующая элементарные и простые знания области деятельности. Рассмотрены представления профессиональных задач и схемы синтеза сложных знаний иерархической структуры, реализующие окрестности объектов разной глубины и обеспечивающие реализацию таких задач.

Ключевые слова: *когнитивная карта, когнитивная цель, синтез знания, область знаний, представление знания, управление контентом*

Введение

Рассматривается задача управления процессами развития содержания многообразий знаний для произвольных областей деятельности. Она состоит в построении связных семантических представлений, аккумулирующих описания свойств и соотношений эмпирических и теоретических знаний. Для реализации таких представлений используют разные способы порождения и извлечения знаний из разных источников. Технологии построения баз знаний для произвольных областей знаний составляют взаимодействующие процессы декомпозиции содержания таких областей в многообразие формализованных знаний простой структуры, а также анализа и синтеза структурированных знаний в связные семантические представления, составляющие основы решения отдельных профессиональных задач. Операциями анализа и синтеза моделируются когнитивные процессы, аналогичные процессам человеческого мышления. Анализ знаний заключается в нахождении значений атрибутов систем знаний с использованием алгебраических и логических инструментов. Процесс синтеза распадается на этапы, связанные с поиском и интеграцией подходящих элементарных и простых знаний в структуры сложных знаний [1]. Синтезируемые структуры являются результатами процессов интеграции знаний, связанных с конкретными когнитивными целями [2]. Достижение таких целей реализуется с использованием преобразований трансформации сложных знаний. Всякая когнитивная цель определяет шаблон синтезируемого знания. Многообразия когнитивных целей отдельных интеллектуальных систем и классы решаемых в них задач определяют классификацию таких систем.

Унифицированная семантическая структура области знаний

Пример унифицированной системы универсальных типов компонентов интеллектуальных систем, соответствующих сформулированным целям моделирования содержания областей знаний, приведен на рис. 1. К базовым инвариантам компонентов такой системы отнесем *сущности*, *инструменты*, *источники* и *содержание области знаний*. Сущности составляют класс объектов, которые являются фокусными для семейства решаемых профессиональных задач. К инструментам относятся средства, используемые для нахождения решений таких задач. Область инструментов составляют классы объектов, определяющих схемы вычислений и анализа значений атрибутов элементов разных классов, а также синтеза сложных структур знаний, применяемых для реализации когнитивных целей, связанных с отдельными профессиональными задачами.

Источники образуют систематизированное описание знаний об объектах, которые содержат знания разных типов, составляющие модель области знаний. Примерами таких источников являются научные публикации, учебники, элементы эмпирических данных и фактов. Ссылки на источники являются одним из способов классификации и группирования знаний. Они позволяют проводить изучение процессов развития областей знаний.

В компоненте содержания области знаний представлено многообразие элементарных и простых знаний с использованием форматов подходящих формализмов представления знаний [2]. Такие знания являются результатом декомпозиции содержания, представленного в источниках, до фрагментов,

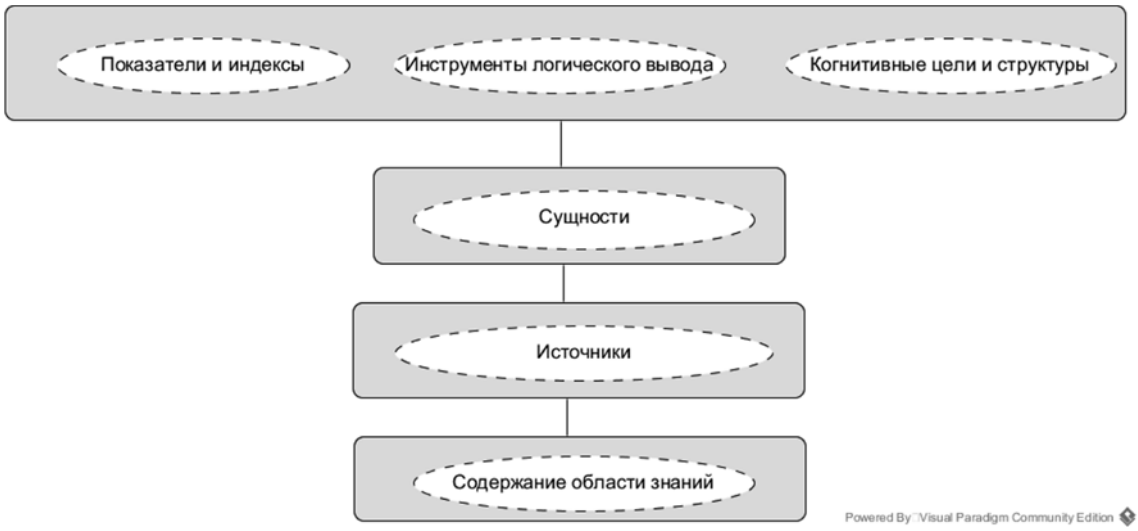

Рис. 1. Унифицированные компоненты интеллектуальной системы

дальнейшая атомизация которых не существенна для достижения целей интеллектуальной системы.

Приведенная система компонентов допускает развитие в общий для содержания разных областей деятельности прототип интеллектуальных систем. Его можно реализовать как фрагмент онтологии, составленной классами объектов разных типов (элементарных знаний) и связей между элементами классов (простые знания). Связями реализуется интеграция элементов классов модели в интегрированную семантическую сетевую структуру. Функциональный и логический аспекты унифицированной модели области знаний составляют многообразия морфизмов и предикатов. Они определяют алгебраические и логические операции над элементами классов всякой такой модели. Прототипы конкретных моделей абстрактных и прикладных систем для различных областей знаний образуют расширения общих конструкций, получаемые с помощью операций детализации и конкретизации.

Когнитивная карта системы управления развитием содержания областей знаний

Рассмотрим подробнее составляющие концепции интеллектуальной системы для баз элементарных и простых знаний, связанной с достижением когнитивных целей управления содержанием областей знаний. Для этого детализируем унифицированные компоненты, представленные на рис. 1. Элементарные и простые знания такой системы связаны с процессами описания и связывания новых знаний. Такие знания формируются как результат деятельности научных организаций, коллективов и отдельных исследователей. Для формализации представления содержания выбранной области используем семантические карты [3]. Пример такой карты для информационной структуры интеллектуальной системы рассматриваемого вида деятельности приведен на рис. 2. Вершины карты соответствуют именованным классам объектов, соответствующих элементарным знаниям, задаваемым с помощью имен. Связи объектов классов представляются на карте одной или несколькими именованными дугами, связывающими пары классов. Уровень сущностей информационной системы реализован классами *задачи*, *области знаний*, *организации* и *авторы*. Источниками знаний об элементах перечисленных классов являются *опубликованные научные работы*, *монографии*, *учебники*, *программы* и другие типы ресурсов. Источники содержат результаты интеллектуальной деятельности конкретных авторов, профессиональных и творческих групп. Каждому источнику соответствует семейство интеллектуальных объектов, составляющих такие результаты.

Компонент инструментов приведенной когнитивной карты содержит классы количественных и качественных показателей (атрибутов и сравнений), сущностей из разных классов, алгоритмов и формул нахождения и сравнения значений атрибутов, а также схем работы с произвольными элементами когнитивной карты, обеспечивающей конструирование семантических структур, необходимых для достижения когнитивных целей.

Примерами общеупотребительных атрибутов профессиональной деятельности ученых являются всевозможные индексы, вычисляемые по эмпирическим данным с помощью специальных формул и алгоритмов. Методы нахождения значений указанных атрибутов представлены элементами классов алгоритмов, формул, когнитивных целей и внешних источников. На рис. 3 приведен фрагмент уровня инструментов, в котором представлены знания о методах вычисления значений атрибутов для элементов классов сущностей.

Способы нахождения значений таких атрибутов представлены элементами классов алгоритмов, формул, когнитивных структур и внешних источников.

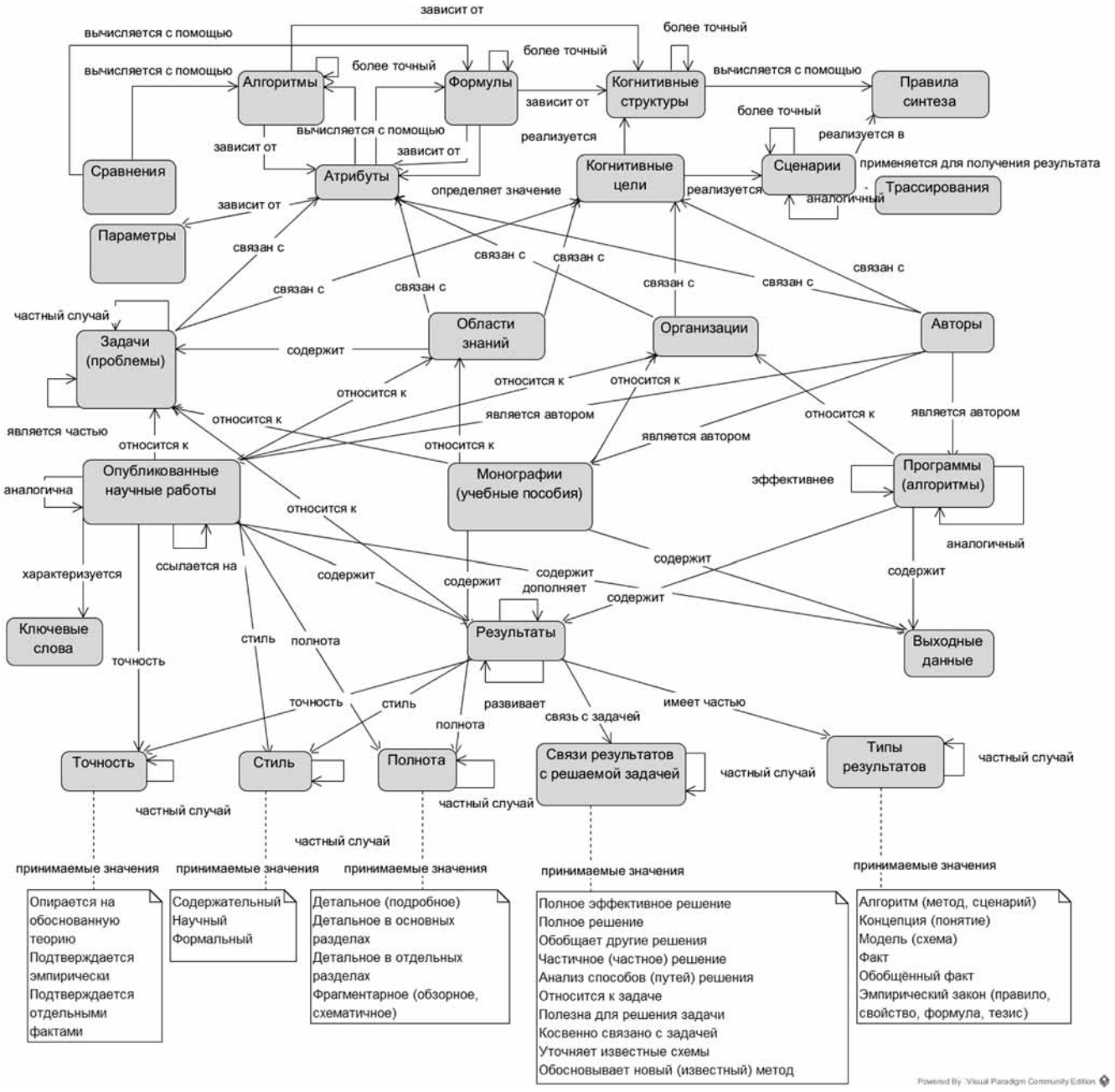

Рис. 2. Семантическая карта управления развитием областей знаний

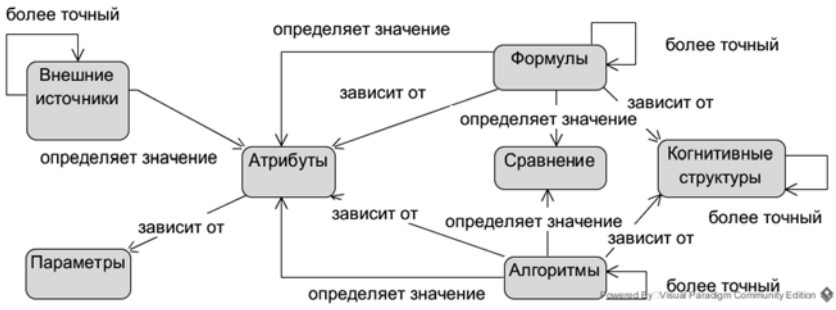

Рис. 3. Фрагмент уровня инструментов когнитивной карты

Основу данного фрагмента составляют класс атрибутов и классы объектов нахождения значений атрибутов, связываемые отношениями *определяет значение* и *зависит от*. Указанные отношения определяют структуру системы знаний, соответствующую формату функциональных сетей [3]. Дополнительное отношение *более точный* позволяет рассматривать разные источники получения значений атрибутов сущностей, обеспечивающих разную точность получаемых

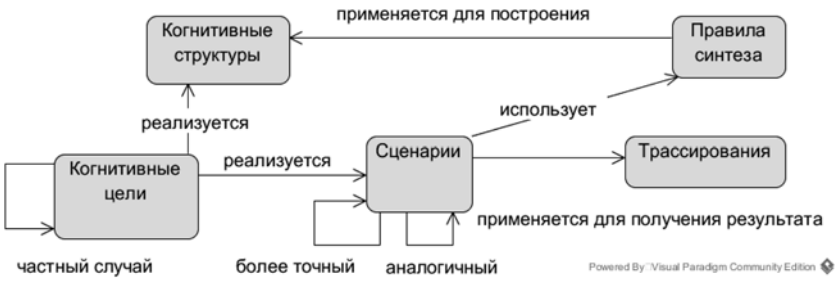

Рис. 4. Фрагмент области инструментов, относящийся к синтезу

с их помощью значений. При этом класс *внешние источники* определяет семейство объектов, интерфейсы с которыми позволяют моделировать операции запроса значений определенных атрибутов, если такие объекты в состоянии их предоставить. Класс когнитивных целей составляют профессиональные задачи, реализация которых связана с составлением связных семантических представлений (когнитивных структур), рассматриваемых как достижения таких целей [1]. Когнитивные структуры составляют класс, элементами которого являются реализации отдельных когнитивных целей. Эти структуры синтезируются из элементов других классов и отношений между такими элементами. Процесс синтеза задается с помощью правил специального языка [4].

С классом когнитивных целей связан фрагмент области инструментов, элементы которых уточняют способы достижения целей. Пример такого фрагмента приведен на рис. 4.

Многообразие проблем в произвольной области профессиональной деятельности составляет причину проведения разнообразных исследований в ней. Это многообразие упорядочено отношением *является*, а его пополнение отражает процессы развития системы задач, исследуемых в области знаний. Для этого отношения соотношение *a является b* означает, что *a* является частным случаем *b*. Использование иерархии задач для последнего отношения связано с конструированием окрестностей отдельных задач. С помощью окрестностей интегрируются результаты исследования отдельных задач. Это позволяет анализировать систему результатов исследований, связанных с отдельными задачами области знаний. Схемы формирования окрестностей и сравнений элементов в них связаны с целями построения и последующей обработки окрестностей. На рис. 5 приведен пример фрагмента иерархии задач области знаний, относящийся к задаче изучения и применения сравнений формализмов представления знаний [3].

Задачи области знаний составляют класс, элементы которого задаются своими именами. Пример унифицированной структуры описания отдельной задачи приведен на рис. 6. Описание составляют атрибуты когнитивной цели, идентификатора объекта исследования, исследуемого аспекта объектов, требования к реализации когнитивной цели, варианта применения результата, потребности решения

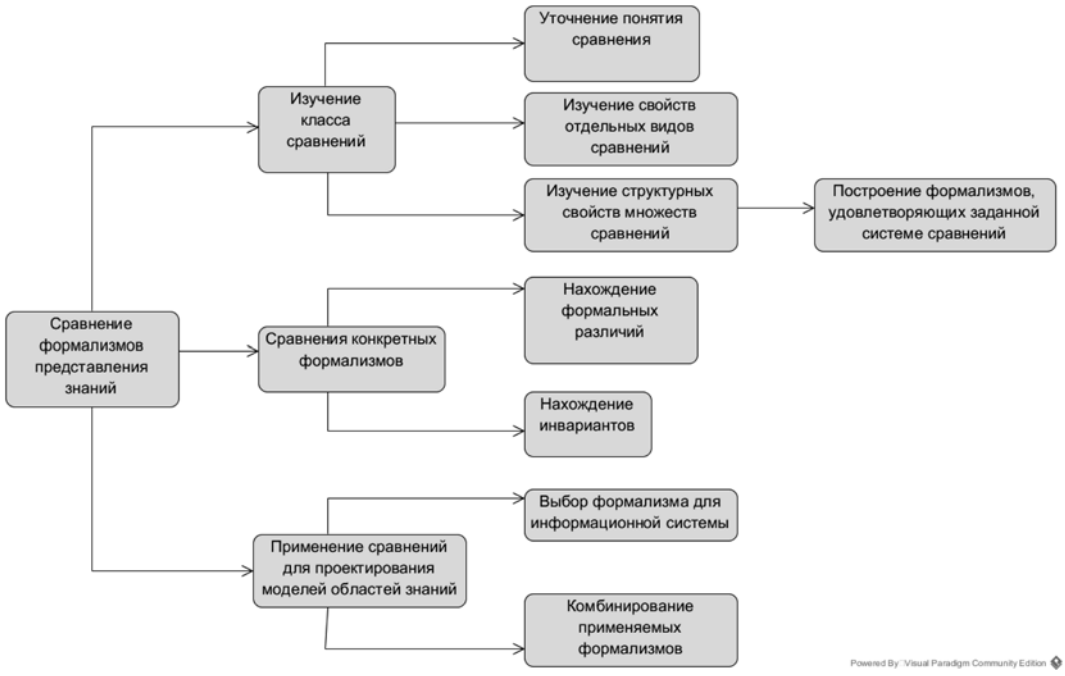

Рис. 5. Фрагмент иерархии проблем области знаний

Рис. 6. Компоненты описания задачи отдельной области знаний

задачи. Полное представление задачи в информационной среде получается с помощью элементов подходящих классов когнитивной карты и отношений между такими элементами. При этом элементами перечисленных классов могут быть стандартные значения, между которыми выполняются подходящие отношения. Примеры возможных значений рассматриваемых атрибутов указаны на рис. 6.

Элементами класса областей знаний являются имена областей деятельности, связанных с созданием и систематизацией конкретных многообразий знаний, объединяемых общностью назначения и использования. Области составляют иерархию в отношении *являться*. Организации составляют группы людей, согласованно ведущих совместную деятельность, связанную с достижением общих целей. Наконец, авторы — это объекты, связанные с получением результатов исследования задач области знаний. Опубликованные результаты деятельности авторов регистрируются в модели области знаний, представленной когнитивной картой.

Компонент источников знаний рис. 2 составлен классами опубликованных научных работ, учебников и монографий, а также программ (алгоритмов). Элементы этих классов связаны отношениями, позволяющими отследить такие общие для наукометрии атрибуты, как авторство публикаций, издания, учреждения, в которых работают авторы, цитирования (ссылки), области знаний, к которым относятся работы, являющиеся источниками знаний.

Класс результатов принадлежит компоненту содержания области знаний. Этот класс составляют объекты, идентифицирующие отдельные результаты. Содержание объектов задается окрестностями, определяемыми семантическими отношениями с элементами классов, представленных на когнитивной карте. Всякий результат характеризуется набором значений атрибутов, позволяющих оценивать содержание и уровень. К ним относятся роли и фильтры, определяющие назначение и достигаемое в результатах качество, связанное с решением профессиональных задач [3].

Роли результатов представлены классом *тип результата*. Качественные свойства результатов различных авторов научных исследований относятся к фильтрам знаний. Семейство фильтров для приведенной на рис. 2 когнитивной структуры составляют классы *точность*, *стиль*, *полнота*, *связь с решаемой задачей*. Каждый из приведенных классов составляет унифицированное множество значений соответствующего атрибуту результата. При этом атрибут качества *точность* отражает уровень обоснованности, *стиль* — оценку способа изложения содержания, а *полнота* — завершенность изложения содержания работы или отдельного результата. Еще одна качественная характеристика результатов научных исследований представлена классом *связь результата с профессиональной проблемой*. Она характеризует результат с точки зрения завершенности исследования задачи. Элементами класса *тип результата* являются значения, определяющие роли результатов в связных семантических представлениях, интегрирующих разнообразные результаты, относящиеся к заданной профессиональной проблеме. Приведенные классы содержат значения качественных атрибутов для

результатов решения задач, применимые для разных областей профессиональной деятельности. На рис. 2 приведены фрагменты указанных классов, составленные общеупотребительными значениями соответствующих характеристик результатов.

Классификатор когнитивных целей

Понятие когнитивной цели относится к разделу инструментов рассматриваемой модели интеллектуальных систем. Целями являются объекты, обозначающие отдельные задачи работы со знаниями. Описание всякой цели согласовано с используемым формализмом представления знаний и имеет вид нагруженного бинарного дерева [4]. Для формализма абстрактного пространства знаний висячие вершины такого дерева размечены элементами классов или символами неизвестных, а внутренние вершины — семантическими отношениями. Реализациями целей являются структуры сложных знаний, задаваемые нагруженными бинарными деревьями. Процессами построения таких структур реализуются сценарии синтеза. Сценарии составляют специальный класс компонента *когнитивные цели и структуры* (см. рис. 1). Для сравнения сценариев применяются отношения точности и эффективности. Со сценариями связаны правила синтеза семантических структур сложных знаний, а также методы извлечения реализаций целей из таких структур (трассирования). Такие методы основаны на монотонных соответствиях вершин структур и их фрагментов, для которых разметки сопоставляемых вершин оказываются сравнимыми [3]. Синтезируемые структуры являются начальными данными формул и алгоритмов области инструментов.

Пример описания простой цели приведен на рис. 7, где A — имя сущности, ρ — семантическое отношение, а X — обозначение фрагмента нагруженного бинарного дерева (конфигурации), являющегося реализацией когнитивной цели, связанной отношением ρ с сущностью A.

Классификация когнитивных целей, рассматриваемой в работе модели области знаний, связана с задачами управления профессиональной научно-исследовательской деятельностью. Для элементов классов сущностей семантической карты, приведенной на рис. 2, актуальны задачи, связанные с построением и сравнением окрестностей, удовлетворяющих специальным требованиям.

Рис. 7. Описание простой цели

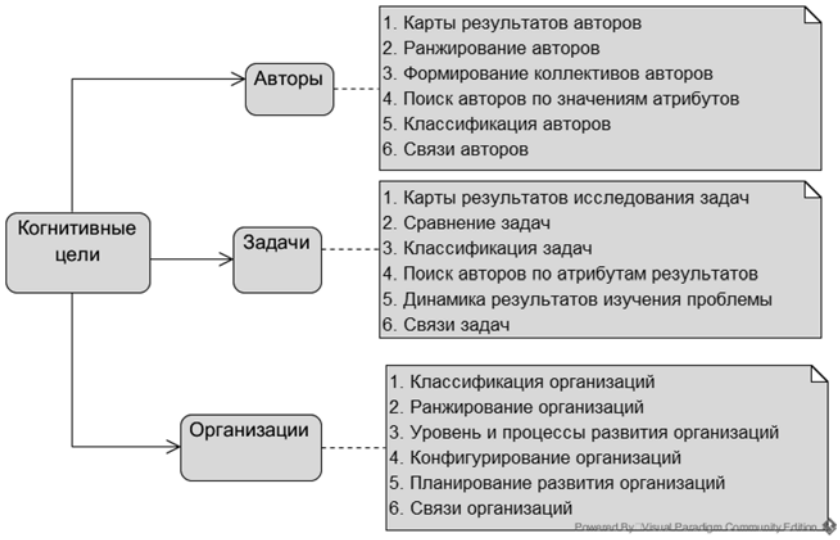

Рис. 8. Иерархия классов целей для сущностей области знаний

Многообразие когнитивных целей составляет иерархию классов целей для отношения вложения классов. Пример фрагмента такой иерархии, относящегося к классам авторов, проблем и организаций, приведен на рис. 8.

Приведенные на рис. 8 списки конкретных целей связаны с задачами сравнения и анализа результатов деятельности организаций отдельных специалистов.

Моделирование окрестностей авторов и профессиональных задач

Рассмотрим примеры когнитивных целей, относящихся к элементам классов *авторы* и *задачи*. Рассмотрим цель нахождения структурно-семантического представления многообразия результатов профессиональной деятельности отдельного автора. Эта цель реализуется с помощью иерархической когнитивной структуры, интегрирующей сведения об опубликованных работах автора, содержащихся в них результатах, а также отношениях между отдельными результатами. Указанная структура может быть расширена включением в нее ссылок на результаты других авторов, представленные в интеллектуальной системе и обладающие необходимыми свойствами. Для описания целей и схем конструирования семантических структур сложных знаний о многообразиях результатов отдельных авторов можно использовать язык конструкций описаний задач и правил синтеза сложных знаний, согласованный с форматами абстрактного пространства знаний [4]. Процесс син-

Рис. 9. Когнитивная структура семейства публикаций автора

теза может быть разбит на несколько этапов, каждый из которых связан с включением в составляемую иерархическую структуру элементов, связываемых конкретным отношением. Такая структура может включать вспомогательные элементы, избыточные для представления реализации рассматриваемой когнитивной цели. Например, это могут быть дополнительные структурные элементы графа развития идеи, отражающего структурные, причинно-следственные и временные зависимости результатов [6]. Рассмотрим последовательность расширений структур сложных знаний, реализующих этапы процесса синтеза. На первом этапе конструируется последовательная серия, составленная публикациями автора. Общий вид такой серии приведен на рис. 9.

Приведенная структура составляет окрестность автора радиуса один для отношения *являться автором*. Использованное для конструирования окрестности отношение параллельной серии позволяет интегрировать семейство объектов, связываемых отношением, приписанным корню создаваемой структуры нагруженного бинарного дерева с объектами в висячих вершинах такой структуры.

Приведенная структура интегрирует не всю информацию об авторе, к которой относятся также сведения о содержании опубликованных научных работ автора и связях работ и составляющих их результатов. Расширение структуры, приведенной на рис. 9, связано с включением в нее серий результатов отдельных работ, извлекаемых из фрагмента когнитивной карты для отношения *содержит*, связывающего класс *опубликованные научные работы*, с классом *результаты*. Здесь также используются иерархические структуры, генерируемые отдельно для каждой опубликованной работы и встраиваемые

Рис. 10. Простая структура для многообразия результатов автора

в структуру, приведенную на рис. 9. На рис. 10 представлен общий вид расширенной структуры. Порядок следования публикаций и содержания публикаций из серий результатов рассматриваемого примера может быть произвольным.

Формируемая семантическая структура составляет окрестность автора радиуса два. Она получается заменой вершин отдельных публикаций на структуры из содержащихся в них результатов, связанных с публикациями отношением *содержит результат*. Расширенная семантическая структура интегрирует знания, позволяющие реализовать более сложную классификацию отдельных авторов, чем рассмотренная окрестность радиуса один. Для этого применяют разнообразные количественные и качественные показатели структур, заданные в области инструментов.

Последующие этапы процесса синтеза сложных знаний состоят в построении окрестностей отдельных результатов, расширяющих приведенную на рис. 10 структуру, с использованием отношений между результатами, представленных в когнитивной карте. Для всякого результата формируется несколько последовательных серий результатов, связанных с ним такими отношениями, как *развивает*, *дополняет*, *конкретизирует, использует*. Последовательные серии группируются в параллельные серии [4], составленные из серий результатов, связанных с заданным результатом конкретными отношениями. Сформированная указанным образом иерархическая структура образует окрестность автора радиуса три. Она обеспечивает совместное представление представительного разнообразия сведений, отражающих структурно-семантические свойства процессов профессиональной деятельности специалиста. Синтезированная когнитивная структура полезна для анализа и сравнения процессов профессиональной деятельности отдельных авторов и групп специалистов. Подобным образом можно интегрировать сведения об объектах других классов сущностей. Например, для элементов класса задач полезны цели, связанные с синтезом когнитивных структур, позволяющих моделировать сравнение, оценивание, ранжирование процессов исследования задач, реализуемое на основе окрестности задачи, составленной результатами разных авторов.

На рис. 11 приведен пример иерархической структуры результатов, отражающей развитие системы знаний о некоторой задаче, представленной в работах одного или нескольких авторов. Эта структура является реализацией шаблона описания структуры сложного знания, интегрирующего результаты разностороннего исследования заданной профессиональной задачи. Система знаний, связанных с задачей, составлена как последовательная серия иерархических структур, интегрирующих содержа-

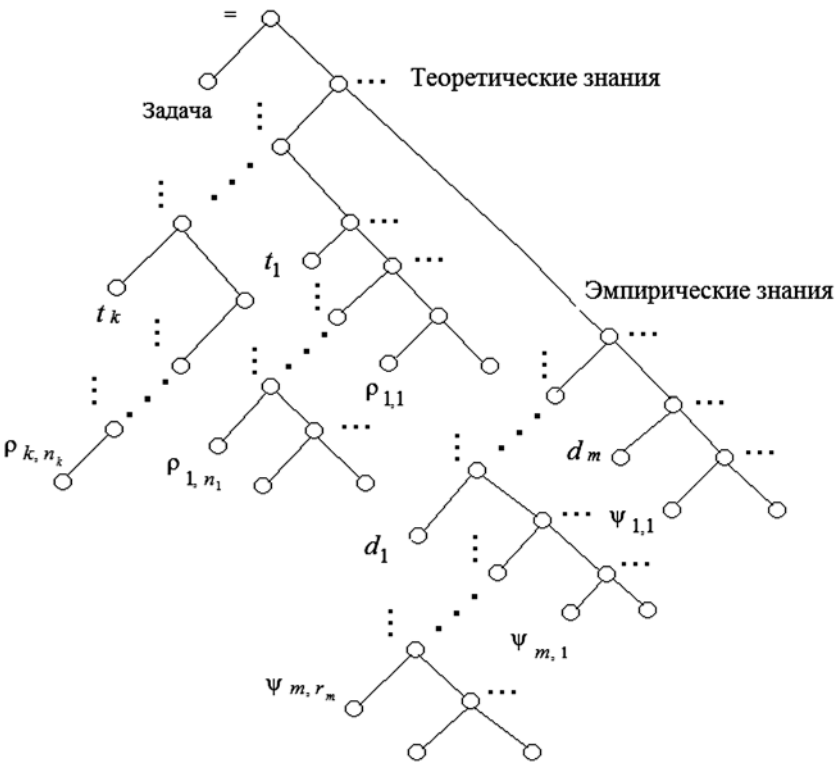

Рис. 11. Инвариантная структура многообразия результатов исследования профессиональной проблемы

ние теоретических (эмпирических) знаний, связанных с этой проблемой.

Заданная структура представляется как параллельная серия основных теоретических (эмпирических) результатов $t_1,..., t_k$ ($d_1,..., d_m$). Отбор объектов, составляющих эти серии, осуществляется с использованием подходящих критериев. Для каждого отобранного объекта t_i (d_j) формируется параллельная серия. Ее составляют семантические отношения этого объекта с другими объектами результатов $\rho_{i,1}, ..., \rho_{i,n_i}$ $\left(\psi_{j,1}, ..., \psi_{j,r_j}\right)$, применяемые в когнитивной карте. Знания, связанные с отобранными объектами разными отношениями, группируются в серии, составляющие окрестности указанных объектов. Всякий элемент серии является основой формирования структуры, развертываемой как семейство окрестностей элемента для разных отношений этого элемента с другими объектами.

Заключение

Построение сложных знаний связано с интеграцией элементарных и простых знаний, составляющих элементы онтологий областей знаний, в иерархические семантические представления, достаточные для реализации когнитивных целей, связанных с решением профессиональных задач в таких областях. Для этого применяются процессы анализа и синтеза, ассоциируемые с процессами мышления. Результатом анализа является когнитивная карта области деятельности, определяющая онтологию этой области. Для моделирования содержания областей знаний используется концепция когнитивной карты [5]. В настоящей работе применено уточнение этой концепции, позволяющее задавать совместные описания классов объектов и классов отношений между объектами онтологии. Процессы синтеза реализуются с использованием схем и правил составления сложных знаний, интегрирующих элементарные и простые знания в связные семантические представления [1, 4]. Синтезированные сложные знания являются основой для решения разнообразных профессиональных задач.

Работа выполнена при поддержке РФФИ, проект № 16-01-00214.

Список литературы

1. **Костенко К. И., Лебедева А. П., Левицкий Б. Е.** Анализ и синтез когнитивных структур при моделировании содержания областей знаний // Современные информационные технологии и ИТ-образование. 2016. Т. 12, № 2. С. 50—55.

2. **Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R.** Taxonomy of educational objectives: The classification Taxonomy of educational goals. Handbook 1: Cognitive domain / Eds. by B. S. Bloom. New York: David McKay, 1956.

3. **Костенко К. И.** Формализмы представления знаний и модели интеллектуальных систем. Краснодар: Изд-во Куб. гос. ун-та, 2015. 300 с.

4. **Костенко К. И.** Моделирование оператора вывода для иерархических формализмов знаний // Программная инженерия. 2016. Т. 7, № 9. С. 424—431.

5. **Willage J., Salustri F. A., Neumann W. P.** Cognitive mapping: Revealing the links between human factors and strategic goals in organizations // Int. Journal Ind. Ergonomics. 2013. Vol. 42, No. 1. P. 304—313.

6. **Налимов В. В., Мульченко З. М.** Наукометрия. Изучения развития науки как информационного процесса. М.: Наука, 1969. 192 с.

The Synthesis of Cognitive Goals Implementation for Tasks of Subject Domains Content Management

K. I. Kostenko, kostenko@kubsu.ru, Kuban State University, Krasnodar, 350040, Russian Federation

Corresponding author:

Kostenko Konstantin I., Assistant Professor, Kuban State University, Krasnodar, 350040, Russian Federation
E-mail: kostenko@kubsu.ru

Received on April 12, 2017
Accepted on May 02, 2017

The unified structure defined for components of system that simulates the development processes of knowledge areas content. This structure based on representation of the separate concept properties by means of semantic networks structures that consist of relations between concepts. The system of such components includes the tasks entities, tools, knowledge sources and elementary knowledge of domain content. Such a system extends opportunities of analysis, estimation and management in the area of creating new knowledge and applying it. Detailed structure of classes that realize separate components and relations between classes proposed for professional activity domain that deals with analysis of the semantic representation for set of objects considered as foundation of integrated knowledge area formal description. The domain content representation based on classes of domain exploration results and results attributes. These separate attributes form the special set that includes classes of meanings for completeness, style, level, purpose and truth. The tools component integrates knowledge of objects that realize operation of professional tasks solving processes. This component consists of such classes as parameters, formulas, algorithms, cognitive goals, cognitive structures, scenarios and cognitive structures synthesis rules. Main classes of entities component represent knowledge areas, knowledge areas tasks, authors and organizations. Classes of objects and relations between classes are basic elements of the system represented by the cognitive map. The sources component accumulates knowledge that relates to the announced and published results of authors scientific activities. Such a map represents the knowledge area ontology and accumulates the set of elementary and simple knowledge of simulated professional activity domain. The representations of professional tasks and the scheme of synthesis the complex knowledge as hierarchical structures are considered. Such knowledge structures accumulate objects dependencies and represent the objects neighborhoods of a given depth that provide separate professional tasks realization.

Keywords: *cognitive map, cognitive goal, knowledge synthesis, knowledge area, knowledge representation, content management*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project nos. 16-01-00214*

For citation:

Kostenko K. I. The Synthesis of Cognitive Goals Implementation for Tasks of Subject Domains Content Management, *Programmnaya Ingeneria,* 2017, vol. 8, no. 7, pp. 319—327.

DOI: 10.17587/prin.8.319-327

References

1. **Kostenko K. I., Lebedeva A. P., Levitskij B. E.** Analiz i sintez kognitivnyh structur pri modelirovanii soderzhanija oblastej znanij (Analysis and synthesis of knowledge structures by knowledge areas content simulation), *Sovremennye informacionnye tehnologii i IT-obrazovanie*, 2016, vol. 12, no. 2, pp. 50—55 (in Russian).

2. **Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R.** *Taxonomy of educational objectives: The classification Taxonomy of educational goals. Handbook 1: Cognitive domain* / Eds. by B. S. Bloom, New York: David McKay, 1956.

3. **Kostenko K. I.** *Formalizmy predstavlenija znanij i modeli intellektualnyh sistem* (Knowledge representation formalisms and intelligent systems models) Krasnodar, Izd-vo Kub. Gos. Un-ta, 2015, 300 p. (in Russian).

4. **Kostenko K. I.** Modelirovanie operatora vyvoda dlja ierarhicheskih formalizmov znanij (Simulation of inference operator for hierarchical knowledge representation formalisms), *Programmnaya Ingeneria*, 2016, vol. 7, no. 9, pp. 424—431 (in Russian).

5. **Willage J., Salustri F. A., Neumann W. P.** Cognitive mapping: Revealing the links between human factors and strategic goals in organizations, *Int. Journal Ind. Ergonomics,* 2013, vol. 42, no. 1, pp. 304—313.

6. **Nalimov V. V., Mulchenko Z. M.** *Naukometria. Izuchenie razvitija nauki kak informacionnogo processa* (Naukometriya. Studying of development of science as information process). Moscow, Nauka, 1969, 192 p. (in Russian).

УДК 004.89 DOI: 10.17587/prin.8.328-336

Е. И. Бурлаева, аспирант, e-mail: ekaterina0853@mail.ru, Донецкий национальный технический университет

Обзор методов классификации текстовых документов на основе подхода машинного обучения

Одной из технологий обработки текстовой информации является автоматическая классификация текстовых документов. Важным этапом при решении задачи классификации текста является выбор метода машинного обучения, который будет применяться к векторному представлению документа. В данной статье приведен анализ различных методов машинного обучения, которые используются для многоклассовой классификации текстовых документов. На основе опубликованных результатов проведен анализ алгоритмов классификации, в результате которого сделан вывод о необходимости повышения качества и скорости классификации текста за счет комбинирования преимуществ этих методов машинного обучения.

Ключевые слова: *автоматическая классификация документов, машинное обучение, метод опорных векторов (SVM), латентно-семантический анализ (LSA), деревья решений, наивный Байесовский классификатор*

Введение

Темпы роста числа документов в электронном виде и их доступность в сети Интернет приводят к тому, что подавляющая часть информации хранится на компьютерах в виде электронных текстовых документов. В большинстве организаций значительная часть полезных знаний содержится в документальных базах данных. Такая ситуация обусловливает повышенный интерес к области *Text mining* — методам автоматического извлечения и обработки знаний из текстовых документов.

Получение знаний в автоматическом режиме затрудняется слабой упорядоченностью текстов на естественном языке. Такие знания могут быть с легкостью извлечены экспертом, но с учетом огромного числа электронных документов их эффективная обработка человеком становится весьма затратной как по времени, так и по используемым ресурсам. Более точное и быстрое извлечение знаний имеет своей конечной целью информационную поддержку эксперта или автоматизированной системы при принятии решения в поставленной задаче (вопросе). В документах, созданных специалистами, могут быть описаны подходы к решению различных проблем, рекомендации к подбору параметров и прочие знания, полезные в различных областях деятельности организации.

Таким образом, основной функцией систем извлечения знаний является информационный поиск полезных сведений в документальных базах. Однако наряду с этой задачей должны решаться и промежуточные задачи автоматической классификации, кластеризации и аннотирования документов. Задача классификации текстов на естественном языке (ЕЯ-текстов) имеет практическое применение в следующих областях:

- сужение области поиска в поисковых системах;
- фильтрация спама;
- составление тематических каталогов;
- контекстная реклама;
- системы документооборота.

Востребованность в эффективном решении задачи автоматической классификации ЕЯ-текстов привела к бурному развитию новых методов и повышению эффективности уже существующих подходов.

Целью данной статьи является анализ эффективности применения классических методов машинного обучения в задачах классификации ЕЯ-текстов.

Далее, на основе анализа публикаций в статье описаны различные алгоритмы машинного обучения, используемые для классификации текстов, их преимущества и недостатки, рассмотрены вопросы эффективности их использования в задачах классификации текстовых документов.

Начиная с начала 1990-х гг. постоянно растет число статистических методов классификации и машинного обучения, которые стали применять к классификации текста. К таким методам можно отнести: многомерные модели регрессии (Friedman 1991; Yang, Chute, 1994; Schütze и др., 1995); классификатор ближайшего соседа (Yang, 1994); вероятностные байесовские модели, деревья решений (Lewis, Ringuette 1994); нейронные сети (Bishop 1995; Schütze и др., 1995) [1], и символическое правило обучения (Apte и др., 1994; Cohen, Singer, 1996). В конце 1990-х Thorten Joachims (1998 г.) исследовал использование

опорных векторов (SVM) для классификации текста с многообещающими результатами [2].

Рассмотрим эффективность различных алгоритмов (метод опорных векторов, наивный байесовский классификатор, латентно-семантический анализ деревьев принятия решений, а также общий латентно-семантический анализ) с позиций скорости обучения, скорости и точности классификации.

1. Постановка задачи классификации ЕЯ-текстов

Большинство известных методов автоматической классификации текстов основаны на предположении, что тексты каждой тематической рубрики содержат отличительные признаки (слова или словосочетания) и, соответственно, на наличии или отсутствии таких признаков в тексте, свидетельствующих о принадлежности или отсутствия принадлежности исследуемого текста той или иной рубрике. Задача методов классификации состоит в том, чтобы наилучшим образом выбрать такие отличительные признаки и сформулировать правила, на основе которых будет принято решение об отнесении текста к рубрике. Актуальность задачи интеллектуальной обработки документов, в частности, состоит в преодолении проблемы отнесения текста к рубрике. Современные классификаторы документов должны решать задачу, связанную с управлением текстовыми документами для более точной рубрикации [3]. Формально задачу классификации можно определить следующим образом.

Имеется множество объектов $T = \{t_i\}$, не обязательно конечное, а также множество $C = \{c_i\}$, $i = 1,..., N_c$, состоящее из N_c классов объектов. Каждый класс c_i представлен некоторым описанием F_i, имеющим некоторую внутреннюю структуру. Процедура классификации f объектов $t \in T$ заключается в выполнении преобразований над ними, после которых либо делается вывод о соответствии t одной из структур F_i, что означает отнесение t к классу c_i, либо вывод о невозможности классификации t.

Применительно к нашей задаче, элементами множества T являются электронные версии текстовых документов. Плоские рубрикаторы состоят из двух уровней, на первом уровне размещается корневая, а на втором уровне — дочерние к корневой рубрики. Общая модель плоского текстового рубрикатора может быть представлена алгебраической системой следующего вида:

$$R = \langle T, C, F, R_c, f \rangle,$$

где T — множество текстов, подлежащих рубрицированию; C — множество классов-рубрик; F — множество описаний; R_c — отношение на $C \times F$; f — операция рубрицирования вида $T \to C$.

Отношение R_c имеет следующее свойство:

$$\forall c_i \in C \; \exists F_i \in F : (c_i, F_i) \in R_c,$$

т. е. классу соответствует единственное описание. Обратное требование необязательно. Отображение f не имеет никаких ограничений, так что возможны ситуации, когда:

$$\exists t \in T : f(t) = C_t \subset C \wedge |C_t| > 1,$$

т. е. некоторый текст может быть отнесен к нескольким классам одновременно.

Кроме сформулированной задачи классификации определяется задача обучения рубрикатора, под которой подразумевается частичное или полное формирование C, F, R_c и f на основе некоторых априорных данных [3, 4].

Исходя из представления модели классификатора, классификаторы могут быть разделены в зависимости от способа представления описаний классов, а также от организации процедуры классификации.

2. Обзор классических алгоритмов автоматической классификации ЕЯ-текста

Процедура автоматической классификации текстов обычно включает две основные части [5]: представление текстов в виде векторов признаков и построение классификатора на созданном массиве векторов. Необходимость представления текстов в виде векторов признаков определяется тем обстоятельством, что рассматриваемые методы классификации требуют, чтобы классифицируемые объекты были представлены в виде последовательностей чисел одинакового размера и одинакового формата. Стандартное представление документа, используемое в текстовой классификации, является векторной моделью.

Можно использовать модель *tf—idf* или более современные аналоги технологий векторного представления текста, такие как Word2vec от Google или fastText от Facebook.

Работа технологии Word2vec осуществляется с помощью принятия большого текстового корпуса в качестве входных данных и сопоставления каждого слова вектору с выдачей координат слов на выходе. Сначала технология создает словарь, "обучаясь" на входных текстовых данных, а затем вычисляет векторное представление слов. Векторное представление основывается на контекстной близости: слова, встречающиеся в тексте рядом с одинаковыми словами (а следовательно, имеющие схожий смысл), в векторном представлении будут иметь близкие координаты векторов-слов. Полученные векторы-слова могут быть использованы для обработки естественного языка и машинного обучения [6].

Компания Facebook объявила об открытии исходных текстов библиотеки fastText, предоставляющей средства для классификации текста с использованием методов машинного обучения. Код написан на языке C++ и открыт под лицензией BSD. Библиотека

позволяет организовать автоматическое назначение категорий для произвольного текста на основании предварительно проведенного обучения по наборам текстов с уже известными категориями. Например, алгоритм fastText может оценить, является ли письмо спамом, или определить, к какой категории относится статья (научная, спорт, финансы, развлечения и т. п.), после обучения по типовым базам спама и тематических статей. Из достоинств fastText можно отметить поддержку различных языков и очень высокую скорость обучения [7].

Для увеличения производительности работы с большим числом категорий в fastText применяется иерархический классификатор, организующий хранение категорий в древовидной структуре вместо обычно применяемых плоских моделей. При этом дерево строится с учетом популярности категорий, что позволяет повысить скорость доступа к часто используемым элементам.

Рассмотрим некоторые из методов автоматической классификации текста, выделим их преимущества и недостатки, что может послужить отправной точкой для разработки более эффективных подходов.

2.1. Векторное представление текста

Векторная модель (*Vector Space Model*, *VSM*) [8] представления текстов является одним из первых способов, применяемых для решения задач тематической классификации текстов. Изначально эта модель применялась в 1996—1997 гг. в задачах определения тем (*Topic Detection and Tracking*) путем извлечения событий из потока информации [9, 10]. Представление текста в данном случае происходит с помощью векторов из одного общего для всей коллекции векторного пространства, в котором каждому слову ставится в соответствие вес в зависимости от выбранной весовой функции [11].

Для полного определения векторной модели необходимо указать, каким именно образом будет определяться вес слова в документе. Для этого используются различные методы: статистический подход (булевский вес [12], *tf—idf* [10], логарифм числа вхождения слова в текст [13] и пр.), место появления слова, оформление слова и др. [13]. При таком представлении текстов можно, например, находить расстояние между точками и решать задачу подобия документов — чем ближе расположены точки, тем больше похожи рассматриваемые тексты [10].

В большинстве прикладных программ, использующих естественно-языковые тексты, применяется векторная модель [14, 15].

Наиболее распространенным является преобразование *tf—idf* и различные его модификации.

Аббревиатура *tf* (*term frequency* — частота слова) означает отношение числа вхождения некоторого слова к общему числу слов документа. Таким образом, оценивается важность терма в пределах отдельного документа. Сочетание *idf* (*inverse document frequency* — обратная частота документа) подразумевает инверсию частоты, с которой некоторый терм встречается в документах коллекции. Учитывается тот факт, что если терм встречается во многих документах множества, он не может являться существенным критерием принадлежности документа рубрике и наоборот.

В преобразовании *tf—idf* существует два интуитивных правила:

- чем чаще терм g_j встречается в документе t_i (т. е. чем больше соответствующая ему частота *term frequency*), тем более значим он в нем;
- чем в большем числе документов терм g_j встречается (т. е. чем меньше соответствующая ему частота *inverse document frequency*), тем менее отличительным он является.

Существует много вариантов *tf—idf*. Приведем один из них:

$$\omega_{ij} = \frac{tf_{ij} \cdot idf_j}{\sqrt{\sum (tf_{ij} \cdot idf_i)^2}},$$

где ω_{ij} — вес i-го терма в документе t_i; idf_j — частота встречаемости i-го терма в рассматриваемом документе; $idf_j = \log N/n$ — логарифм отношения числа документов в коллекции к числу документов, в которых встречается i-й терм. Веса, вычисленные по этой формуле, нормализованы таким образом, что сумма квадратов весов каждого документа равна единице [16].

Модель *tf—idf* [16—19] позволяет устранить сложности присутствия в тексте общеупотребимых слов путем домножения частот их появления на функцию, монотонно убывающую по общей встречаемости слова в языке.

В некоторых случаях для вычисления веса слова в тексте привлекается также дополнительная информация [20]. Например, можно учитывать информацию о структуре текста и словам, встреченным в заголовке, присваивать больший вес [21].

2.2. Метод опорных векторов (SVM)

Алгоритм классификации SVM, предложенный Вапником [22], является контролируемым алгоритмом обучения и принадлежит к группе методов детерминистского подхода. Это двоичный линейный классификатор, который отделяет позитивный и негативный примеры в тестовом наборе. Метод реализуется путем поиска оптимальной гиперплоскости, отделяющей положительные примеры от отрицательных примеров, гарантируя таким образом, что граница между ближайшими позитивами и негативами является максимальной.

Метод SVM базируется на следующем постулате [23]: наилучшей разделяющей плоскостью является та, которая максимально далеко отстоит от ближайших до нее точек обоих классов. Таким образом, задача метода SVM состоит в том, чтобы найти разделяющую полосу с максимальной шириной, поскольку, чем шире полоса, тем увереннее можно классифицировать объекты. Как следствие, в методе SVM считается, что самая широкая полоса является наилучшей. Границами полосы (рис. 1) являются две параллельные гиперплоскости с направляющим вектором **w**. Точки, ближайшие к разделяющей

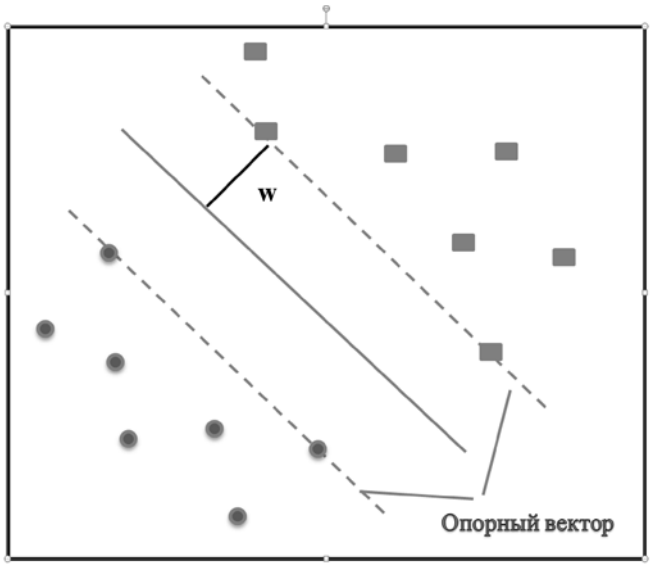

Рис. 1. Пример результата работы метода SVM

гиперплоскости, расположены точно на границах полосы. При этом сама разделяющая гиперплоскость проходит ровно посередине полосы (рис. 1).

В 1992 г. Bernhard E. Boser, Isabelle M. Guyon и Vladimir N. Vapnik предложили способ создания нелинейного классификатора, в основе которого лежит переход от скалярных произведений к произвольным ядрам, так называемый *kernel trick* (предложенный впервые М. А. Айзерманом, Э. М. Браверманом и Л. В. Розоноэром для метода потенциальных функций), позволяющий строить нелинейные разделители [24]. Результирующий алгоритм крайне похож на алгоритм линейной классификации, с той лишь разницей, что каждое скалярное произведение в приведенных выше формулах заменяется нелинейной функцией ядра (скалярным произведением в пространстве с большей размерностью). В этом пространстве уже может существовать оптимальная разделяющая гиперплоскость. Так как размерность получаемого пространства может быть больше размерности исходного, то преобразование будет нелинейным, а значит функция, соответствующая в исходном пространстве оптимальной разделяющей гиперплоскости, будет также нелинейной.

Если исходное пространство имеет достаточно высокую размерность, то можно надеяться, что в нем выборка окажется линейно разделимой.

Использование метода SVM для классификации текстов обладает следующими преимуществами:

- на тестах с массивами документов его реализация превосходит другие методы;
- при выборе ядра, задаваемого различными функциями, становится возможной эмуляция различных подходов;
- итоговое правило выбирается не с помощью некоторых эвристик, а путем оптимизации некоторой целевой функции.

К недостаткам SVM можно отнести следующие:

- малое число параметров для настройки, а именно, после того как фиксируется функция ядра, единственным параметром, который варьируется, остается коэффициент ошибки;
- нет четких критериев выбора функции ядра, есть лишь рекомендации, позволяющие сделать предположение об эффективности использования той или иной функции;
- довольно медленное обучение системы классификации.

2.3. Латентно-семантический анализ (LSA)

Латентно-семантический анализ (*LSA*, *Latent Semantic Analysis*) — это метод для извлечения контекстно-зависимых значений слов с помощью статистической обработки больших наборов текстовых данных [25]. Метод LSA был запатентован в 1988 г. и относится к классификаторам, основанным на функциях подобия. В области информационного поиска этот подход называют латентно-семантическим индексированием (*LSI*, *Latent Semantic Indexing*). Метод LSA также работает с векторным представлением типа "мешок слов" текстовых единиц. Его суть состоит в том, что неважен порядок слов в документе, в каких морфологических формах они представлены, а важно только число вхождений конкретных слов. Предположим, что каждую тему можно охарактеризовать определенным набором слов и частотой их появления. Если в тексте конкретный набор слов употребляется с определенными частотами, то текст принадлежит к определенной теме. Текстовый корпус представляется в виде числовой матрицы, строки которой соответствуют словам, а столбцы — текстовым единицам.

Объединение слов в темы и представление текстовых единиц в пространстве тем осуществляется путем применения к данной матрице одного из матричных разложений. Наиболее популярными являются сингулярное разложение [25] и факторизация неотрицательных матриц [26].

Каждое слово и текст представляются с помощью векторов в общем пространстве размерности k — пространстве гипотез. Сходство между любой комбинацией слов и/или текстов легко вычисляется с помощью скалярного произведения векторов. Как правило, выбор k зависит от поставленной задачи и подбирается эмпирически. Если выбранное значение k слишком велико, то метод теряет свою эффективность и приближается по характеристикам к стандартным векторным методам. Слишком маленькое значение k не позволяет улавливать различия между похожими словами или текстами [23].

К достоинствам метода LSA можно отнести следующие:

- метод является наилучшим для выявления латентных зависимостей внутри множества документов;
- используются значения матрицы близости, основанной на частотных характеристиках документов и лексических единиц;
- частично снимаются вопросы, связанные с полисемией и омонимией.

К недостаткам LSA относятся перечисленные далее.

- Существенным недостатком метода является значительное снижение скорости вычисления при увеличении объема входных данных. Как показано в работе [27], скорость вычисления соответствует порядку N^{2k}, где $N = N_{\text{doc}} + N_{\text{term}}$ — сумма числа документов и термов; k — размерность пространства факторов.
- Вероятностная модель, которую использует метод, не соответствует реальности. Предполагается, что слова и документы имеют нормальное распределение, хотя ближе к реальности распределение Пуассона. В связи с этим обстоятельством лучше подходит вероятностный латентно-семантический анализ, основанный на мультиномиальном распределении.

Вероятностный латентно-семантический анализ (ВЛСА), также известный как вероятностное латентно-семантическое индексирование (ВЛСИ), особенно в области информационного поиска, — это статистический метод анализа корреляции двух типов данных. Данный метод является развитием латентно-семантического анализа. Подход на основе ВЛСА применяется в таких областях, как информационный поиск, обработка естественного языка, машинное обучение и в смежных областях. Данный метод впервые был опубликован в 1999 г. T. Hofmann [28, 29].

По сравнению с обычным латентно-семантическим анализом, который основан на методах линейной алгебры и является способом снижения размерности матрицы, вероятностный латентно-семантический анализ основан на смешанном разложении, которое, в свою очередь, берет свое начало из модели скрытых классов. Данный подход более доказателен, поскольку имеет прочную основу в области статистики.

2.4. Вероятностные модели на основе наивного байесовского классификатора

Наивный байесовский классификатор строится с помощью обучающих данных. Он нацелен на то, чтобы оценить вероятность каждой категории, учитывая особенность документа как значения нового экземпляра [30].

Наивный байесовский алгоритм (НБА) — это алгоритм классификации, основанный на теореме Байеса с допущением о независимости признаков. Байесовский классификатор использует теорему Байеса об апостериорной вероятности:

$$P\left(C = c_k \middle| \mathbf{x}\right) = \frac{P\left(\mathbf{x} \middle| C = c_k\right) P\left(C = c_k\right)}{P\left(\mathbf{x}\right)},$$

где $P(c|x)$ — апостериорная вероятность данного класса c (т. е. данного значения целевой переменной) при данном значении признака x; $P(c)$ — априорная вероятность данного класса; $P(x|c)$ — правдоподобие, т. е. вероятность данного значения признака при данном классе; $P(x)$ — априорная вероятность данного значения признака.

Наивный байесовский классификатор хорошо известен и является широко используемым на практике вероятностным классификатором. Наивным этот классификатор называется, поскольку предполагает, что все атрибуты (т. е. признаки) примеров являются независимыми друг от друга с учетом контекста класса. Несмотря на то что предположение об условной независимости, как правило, не соответствует действительности с позиции появления слова в документах, наивный байесовский классификатор показывает достаточно хорошие результаты при несоблюдении условия статистической независимости [31].

Отмечают следующие достоинства байесовского классификатора.

- Классификация, в том числе многоклассовая, выполняется легко и быстро.
- Когда допущение о независимости выполняется, НБА превосходит другие алгоритмы, такие как логистическая регрессия (*logistic regression*), и при этом требует меньший объем обучающих данных.
- Подход с использованием НБА лучше работает с категорийными признаками, чем с непрерывными. Для непрерывных признаков предполагается нормальное распределение, что является достаточно сильным допущением.

К недостаткам классификатора можно отнести следующие.

- Если в тестовом наборе данных присутствует некоторое значение категориального признака, которое не встречалось в обучающем наборе данных, то модель присвоит нулевую вероятность этому значению и не сможет сделать прогноз. Это явление известно под названием "нулевая частота" (*zero frequency*). Эту задачу можно решить с помощью сглаживания. Одним из самых простых методов является сглаживание по Лапласу (*Laplace smoothing*).
- Хотя НБА является хорошим классификатором, значения спрогнозированных вероятностей не всегда являются достаточно точными. Поэтому не следует слишком полагаться на результаты, полученные методом НБА.
- Еще одним ограничением НБА является допущение о независимости признаков. В реальности наборы полностью независимых признаков встречаются крайне редко.

2.5. Дерево принятия решений

Дерево принятия решений представляет собой простой классификатор и широко используется в задачах классификации текстов.

"Деревья решений" (для задачи классификации) — это метод, позволяющий предсказывать принадлежность наблюдений или объектов к тому или иному классу категориальной зависимой переменной в соответствии со значениями одной или нескольких предикторных переменных [32].

Итак, дерево решений, подобно его прототипу из живой природы, состоит из ветвей и листьев. Ветви (ребра графа) хранят в себе значения атрибутов, от которых зависит целевая функция; на листьях же записывается значение целевой функции. Существуют также и другие узлы — родительские и потомки, по которым происходит разветвление (рис. 2).

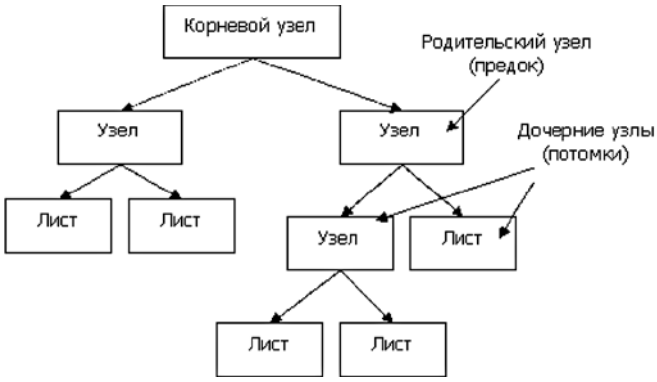

Рис. 2. Пример дерева решений

Один из способов автоматического построения деревьев решений заключается в последовательном разбиении множества обучающих документов на классы, до тех пор, пока в классе не останется документов, определенных только в одну из категорий. На каждом этапе в качестве узла дерева выбирается терм, содержащий множество всех возможных значений, и определяется условие для ветвей, затем множество документов разбивается на два класса, каждый из которых имеет свои условия.

Обычно построенное дерево решений является сильно детализированным (эффект переобучения с помощью адаптации модели к частным случаям, нетипичным примерам, шумам в данных и т. д.), поэтому применяются различные алгоритмы усечения дерева. Широкое применение получили алгоритмы ID3 [33] и C4.5 [34].

Преимущества деревьев решений заключаются в следующем:

- просты в понимании и интерпретации, не требуют подготовки данных;
- позволяют оценить модель с помощью статистических тестов;
- быстро обучаются.

Недостатки деревьев решений:

- задача получения оптимального дерева решений бывает NP-полной;
- могут появиться слишком сложные конструкции, которые при этом недостаточно полно представляют данные;
- существуют концепты, которые сложно понять из модели, так как модель описывает их сложным путем;
- для данных, которые включают категориальные переменные с большим набором уровней, больший информационный вес присваивается тем атрибутам, которые имеют большее число уровней.

3. Сравнение методов машинного обучения

Большое число исследований эффективности методов автоматической рубрикации проводится на популярной коллекции финансовых сообщений информационного агентства Рейтер — Reuters-21578 [35], которая была специально создана для тестирования методов автоматической рубрикации текстов. Для этой коллекции характерны следующие особенности:

- тексты сообщений небольшие по размеру и принадлежат узкой предметной области финансовых и биржевых новостей:
- рубрикатор, включающий 135 рубрик, относительно прост, без иерархии, причем обычно [36] для тестирования используются лишь десять наиболее частотных рубрик (присвоение рубрик проводилось с контролем качества работы экспертов).

В частности, 40 % из имеющихся 21 578 документов не рекомендуются к использованию в силу того, что присвоение рубрик к ним признано некачественным. Оставшиеся 12 902 документа помечены как "качественно отрубрицированные".

Для десяти наиболее частотных рубрик коллекции Reuters-21578 результаты применения машинного обучения весьма высоки — в среднем около 84 %. Сравнительные исследования эффективности методов машинного обучения на коллекции Reuters-21578 [37—39] показали, что наиболее эффективным методом является метод опорных векторов SVM (по сравнению с методами Байеса, ближайших соседей, Rocchio, деревьев решений C4.5, нейронных сетей, байесовских сетей [40]).

Анализ опубликованных работ позволяет отметить следующее:

- большинство методов тестируется на коллекции Reuters-21578, состоящей из коротких сообщений с очень простым рубрикатором;
- для тестирования выбираются только рубрики с большим числом примеров.

Существуют востребованные практикой задачи, в которых применение описанных в литературе методов машинного обучения неэффективно [40, 42]. В реальных задачах по-прежнему часто используют ручной труд экспертов, а также системы рубрикации, основанные на вручную задаваемых правилах [30, 33].

На основании представленного выше краткого анализа можно сделать выводы о том, что наиболее быстрым является метод SVM, а наиболее точным — метод LSA.

Использование векторно-пространственной модели представления текста не лишено недостатков. К факторам, усложняющим или делающим невозможным применение методов машинного обучения для автоматической рубрикации текстов, можно отнести перечисленные далее [39].

- При отсутствии простейшей дополнительной обработки, такой как морфологический анализ, существенно снижается качество классификатора. Причина в том, что разные формы одного слова считаются разными терминами, а морфологический анализ — весьма нетривиальная задача, требующая для ее решения привлечения лингвистов.
- Размерность векторов признаков зависит от общего числа терминов в обучающей выборке текстов, что в реальных задачах приводит к необходимости разрабатывать альтернативные структуры данных, отличные от векторов.

• Словарь терминов может не охватывать всех документов, подлежащих классификации, так что анализируемые документы могут содержать значимые термины, не вошедшие в обучающую выборку. Это обстоятельство отрицательно сказывается на адекватности модели.

Создание достаточно большой, последовательно отрубрицированной текстовой коллекции, является сложной для решения организационно-технической задачей.

Заключение

В статье были рассмотрены вопросы и сложности, возникающие при решении задач автоматической классификации текстовых документов. Отмечены преимущества и недостатки основных методов классификации текста. Результаты представленного краткого обзора позволяют сделать следующие выводы.

Для улучшения характеристик классификатора при классификации текста необходимо последовательное объединение нескольких алгоритмов классификации. При формировании последовательности необходимо оптимизировать критерии качественного обучения каждого метода классификации. Для увеличения точности классификации при объединении методов и взаимодействии между собой необходимо их оптимальное сочетание, например, бустингом [42] или случайным лесом [43]. При этом предварительно следует сократить размерности обучающей выборки с помощью представленного выше метода ВЛСА.

Список литературы

1. **Dumais S. T., Platt J. C., Hecherman D., Sahami M.** Inductive Learning Algorithms and Representations for Text Categorization // Proceedings of CIKM 1998, Bethesda, US. ACM Press, New York, 1998. P. 148—155.

2. **Joachims T.** Text Categorization with Support Vector Machines: Learning with Many Relevant Features // European Conference on Machine Learning (ECML), 1998. P. 137—142.

3. **Золотухин О. В.** Классификация политематических текстовых документов с использованием нечетких нейросетевых технологий // Системы обработки информации. 2012. № 9 (107). С. 101—105. URL: http://www.hups.mil.gov.ua/periodic-app/article/10215/soi_2012_9_26.pdf.

4. **Андреев А. М., Березкин Д. В., Морозов В. В., Симаков К. В.** Автоматическая классификация текстовых документов с использованием нейросетевых алгоритмов и семантического анализа // Труды 5-й Всероссийской научной конференции "Электронные библиотеки: перспективные методы и технологии, электронные коллекции" — RCDL2003, Санкт-Петербург, Россия, 2003. URL: http://rcdl.ru/doc/2003/B1.pdf.

5. **Киселев М. В.** Оптимизация процедуры автоматического пополнения веб-каталога // Интернет-математика 2005. Автоматическая обработка веб-данных. М.: Яндекс, 2005. С. 342—363. URL: http://elar.urfu.ru/bitstream/10995/1417/1/IMAT_2005_18.pdf.

6. **Mikolov T., Chen K., Corrado G., Dean J.** Efficient Estimation of Word Representations in Vector Space// Proceedings of Workshop at ICLR, 2013. P. 1—12.

7. **Библиотека** исходного текста fastText. URL: https://github.com/facebookresearch/fastText

8. **Salton G., Wong A., Yang C. S.** A Vector Space Model for Automatic Indexing // Communications of the ACM. 1975. Vol. 18, No 11. P. 613—620.

9. **Allan J., Carbonell J., Doddington G. et al.** Topic Detection and Tracking Pilot Study. Final Report // Proceedings of the Broadcast News Transcription and Understanding Workshop (Supported by DARPA). 1998. P. 1—16.

10. **Коршунов А., Гомзин А.** Тематическое моделирование текстов на естественном языке // Труды Института Системного Программирования РАН, 2012. Т. 23. С. 216—243.

11. **Leskovec J., Rajaraman A., Ullman J.** Mining of Massive Datasets. Cambridge University Press, 2014. 513 p.

12. **Губин М. В.** Модели и методы представления текстового документа в системах информационного поиска: Автореф. дис. кан. физ.-мат. наук. Санкт-Петербург, 2005. 14 с.

13. **Сэлтон Г.** Автоматическая обработка, хранение и поиск информации. М.: Сов. Радио, 1973. 560 с.

14. **Salton G., Buckley C.** Term-weighting approaches in automatic text retrieval // Information Processing & Management. 1988. Vol. 24, No. 5. P. 513—523.

15. **Епрев А. С.** Автоматическая классификация текстовых документов // Математические структуры и моделирование 2010. № 21. С. 65—81.

16. **Jones S. K.** A statistical interpretation of term specificity and its application in retrieval // Journal of documentation. 1972. Vol. 28, No. 1. P. 11—21.

17. **Salton G.** Developments in automatic text retrieval // *Science.* 1991. Vol. 253, No. 5023. P. 974—980.

18. **Manning Ch. D., Raghavan P., Schutze H.** Scoring, term weighting and the vector space model // Introduction to Information Retrieval. 2008. Vol. 100. pp. 504.

19. **Загорулько Ю. А., Кононенко И. С., Костов Ю. В., Сидорова Е. В.** Классификация деловых писем в системе документооборота // Материалы междунар. науч.-техн. конф. «Информационные системы и технологии» (ИСТ'2003). Новосибирск: Изд-во НГТУ, 2003, Т. 3. С. 141—145.

20. **Norkin V., Keyzer M.** On stochastic optimization and statistical learning in reproducing kernel Hilbert spaces by Support Vector Machines (SVM) // Informatica. 2009. Vol. 20, No. 2. P. 273—292.

21. **Beuster G.** MIC — A System for Classification of Structured and Unstructured Texts. Diploma Thesis. University Koblenz, 2001.

22. **Chapelle O., Vapnik V., Bousquet O., Mukherjee S.** Choosing Multiple Parameters for Support Vector Machines // Machine Learning. 2002. Vol. 46, No. 1. P. 131—159.

23. **Landauer T. K., Foltz P., Laham D.** An Introduction to Latent Semantic Analysis // Discourse Processes. 1998. Vol. 25. P. 259—284. URL: http://lsa.colorado.edu/papers/dp1.LSAintro.pdf.

24. **Вьюгин В.** Математические основы машинного обучения и прогнозирования. МЦМНО, 2014. 304 с.

25. **Kim H., Park H.** Nonnegative Matrix Factorization Based on Alternating Nonnegativity Constrained Least Squares and Active Set Method// SIAM Journal on Matrix Analysis and Applications, 2008. Vol. 30, No. 2. P. 713—730. URL: http://www.cc.gatech.edu/~hpark/papers/simax-nmf.pdf (дата обращения 23.04.2017).

26. **Воеводин В. В., Кузнецов Ю. А.** Матрицы и вычисления. М.: Наука, 1984. 320 с.

27. **Deerwester S., Dumais S. T., Furnas G. W. et al.** Indexing by Latent Semantic Analysis // Journal of the American Society for Information Science. 1990. Vol. 41, No. 6. P. 391—407.

28. **Hofmann T.** Probabilistic Latent Semantic Indexing// Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in en:Information Retrieval (SIGIR-99), 1999. P. 50—57.

29. **Hofmann T.** Probabilistic Latent Semantic Analysis// In Proc. of Uncertainty in Artificial Intelligence, UAI'99, Stockholm. 1999. P. 289—296.

30. **Rish I.** An empirical study of the naive Bayes classifier // IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence. 2001. URL: http://www.research.ibm.com/people/r/rish/papers/RC22230.pdf.

31. **Chavan G. S., Manjare S., Hegde P., Sankh A.** A Survey of Various Machine Learning Techniques for Text Classification // International Journal of Engineering Trends and Technology (IJETT). 2014. Vol. 15, No 6. P. 288—292.

32. **Breiman L., Friedman J. H., Olshen R. A., Stone C. J.** Classification and regression trees. Monterey, CA: Wadsworth & Brooks. Cole Advanced Books & Software, 1984.

33. **Quinlan J.** Induction of decision trees // Machine Learning. 1998. Vol. 1, No. 1. P. 81—106.

34. **Quinlan J.** C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993. 302 p.

35. **Агеев М. С., Журавлев С. В., Ламбурт В. Г.** Подготовка Web-версий традиционных изданий // Открытые Системы. 2000. № 12. С. 31—35.

36. **Агеев М. С., Добров Б. В., Макаров-Землянский Н. В.** Метод машинного обучения, основанный на моделировании логики рубрикатора // RCDL'2003 Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Пятая всероссийская науч. конф. Санкт-Петербург, 2003. С. 150—158.

37. **Ageev M., Dobrov B., Loukachevitch N.** Text Categorization Tasks for Large Hierarchial Systems of Categories // SIGIR 2002 Workshop on Operational Text Classification Systems / Eds. F. Sebastiani, S. Dumas, D. D. Lewis, T. Montgomery, I. Moulinier Univ. of Tampere, 2002. P. 49—52.
38. **Ageev M., Dobrov B., Makarov-Zemlyanskii N.** On-line Thematic and Metadata Analysis of Document Collection // New Trends in Intelligent Information Processing and Web Mining'2004: Proceedings of the International Conference. Springer, Advanced in Soft Computing. Zakopane, Poland, May 2004. P. 279—286.
39. **Dumais S., Lewis D., Sebastiani F.** Report on the Workshop on Operational Text Classification Systems (OTC-02) // SIGIR-2002 — Tampere, Finland, 2002. URL: http://sigir.org/files/forum/F2002/sebastiani.pdf
40. **Rose T., Stevenson M., Whitehead M.** The Reuters Corpus Volume 1 — from Yesterday News to tomorrow's Language // In Proceedings of the Third International Conference on Language Resources and Evaluation, Las Palmas de Gran Canaria, 29—31 May 2002. P. 29—31.
41. **Wasson M.** Classification Technology at LexisNexis // SIGIR 2001 Workshop on Operational Text Classification. URL: http://www.daviddlewis.com/events/otc2001/presentations/otc01-wasson-paper.txt
42. **Valiant L. G.** A theory of the learnable// Communications of the ACM. 1984. Vol. 27, No. 11. P. 1134—1142.
43. **Breiman L.** Random forests // Machine Learning. 2001. Vol. 45, No. 1. P. 5—32.

Review of Methods for Classifying Text Documents Based on the Machine Learning Approach

E. I. Burlayeva, ekaterina0853@mail.ru, Donetsk National Technical University, Pokrovsk, Donetsk region, 85300, Ukraine

Corresponding author:

Burlayeva Ekaterina I., Graduate Student, Donetsk National Technical University, Pokrovsk, Donetsk region, 85300, Ukraine
E-mail: ekaterina0853@mail.ru

Received on March 23, 2017
Accepted on May 04, 2017

Increased interest in creating effective tools for working with textual information based on automatic text processing is due to a sharp increase in textual information in electronic form leading to the need to automate various types of activities. Automatic text classifiers can be useful in almost any system where text documents are used to represent information. The use of classifiers makes it possible to reduce the labor costs for finding the necessary information represented by electronic texts. In order to understand the variety of documents, certain rules for their compilation, forms and methods of working with documents have been developed. One of the technologies for processing textual information is the automatic classification of text documents, which consists of assigning a document to one of several categories based on the content of the document. An important step in solving the task of classifying text is the choice of the machine learning method that will be applied to the vector representation of the document. This article presents a comparative analysis of various methods of machine learning, which are used for multi-class classification of text documents. Comparing and studying the four classification algorithms, namely, support vector method (SVM), latent-semantic analysis (LSA), naive Bayes and decision tree. There is a need to improve the quality and speed of the text classification, by combining the advantages of known text classification algorithms.

Keywords: *automatic classification of documents, machine learning, support vector method (SVM), latent semantic analysis (LSA), decision trees, naive Bayesian classifier*

For citation:

Burlayeva E. I. Review of Methods for Classifying Text Documents Based on the Machine Learning Approach, *Programmnaya Ingeneria,* 2017, vol. 8, no. 7, pp. 328—336.

DOI: 10.17587/prin.8.328-336

References

1. **Dumais S. T., Platt J. C., Hecherman D., Sahami M.** Inductive Learning Algorithms and Representations for Text Categorization, *Proceedings of CIKM* 1998, Bethesda, US. ACM Press, New York, 1998, pp. 148—155.
2. **Joachims T.** Text Categorization with Support Vector Machines: Learning with Many Relevant Features, *European Conference on Machine Learning* (ECML), 1998, pp 137—142.
3. **Zolotuhin O. V.** Klassifikacija politematicheskih tekstovyh dokumentov s ispol'zovaniem nechetkih nejrosetevyh tehnologij (Polythematic classification of text documents using fuzzy neural network technology), *Sistemy obrabotki informacii*, 2012, no. 9 (107). pp. 101—105, available at: http://www.hups.mil.gov.ua/periodic-app/article/10215/soi_2012_9_26.pdf (in Russian).
4. **Andreev A. M., Berezkin D. V., Morozov V. V., Simakov K. V.** Avtomaticheskaja klassifikacija tekstovyh dokumentov s ispol'zovaniem nejrosetevyh algoritmov i semanticheskogo analiza (Automatic classification of text documents using neural network algorithms and semantic analysis), *Trudy 5 Vserossijskoj nauchnoj konferencii "Jelektronnye biblioteki: perspektivnye metody i tehnologii, jelektronnye kollekcii" — RCDL2003*, Saint-Petersburg, 2003, available at: http://rcdl.ru/doc/2003/B1.pdf (in Russian).
5. **Kiselev M. V.** Optimizacija procedury avtomaticheskogo popolnenija veb-kataloga (Optimization of the procedure of automatic replenishment web directory), *Internet-matematika* 2005, *Avtomaticheskaja obrabotka veb-dannyh*, Moscow Yandex, 2005,

pp. 342—363, available at: http://elar.urfu.ru/bitstream/10995/1417/1/IMAT_2005_18.pdf (in Russian).

6. **Mikolov T., Chen K., Corrado G., Dean J.** Efficient Estimation of Word Representations in Vector Space, *Proceedings of Workshop at ICLR*, 2013, pp. 1—12.

7. **Library** for fastText, available at: https://github.com/facebookresearch/fastText.

8. **Salton G., Wong A., Yang C. S.** A Vector Space Model for Automatic Indexing, *Communications of the ACM*, 1975, vol. 18, no. 11, pp. 613—620.

9. **Allan J., Carbonell J., Doddington G., Yamron J., Yang Y.** Topic Detection and Tracking Pilot Study. Final Report, *Proceedings of the Broadcast News Transcription and Understanding Workshop* (Supported by DARPA), 1998, pp. 1—16.

10. **Korshunov A., Gomzin A.** Tematicheskoe modelirovanie tekstov na estestvennom jazyke (Topic modeling of natural language texts), *Trudy Instituta Sistemnogo Programmirovanija RAN*, 2012, vol. 23, pp. 216—243 (in Russian).

11. **Leskovec J., Rajaraman A., Ullman J.** *Mining of Massive Datasets.* Cambridge University Press, 2014, 513 p.

12. **Gubin M. V.** Modeli i metody predstavlenija tekstovogo dokumenta v sistemah informacionnogo poiska (Models and methods for representation of text document in information retrieval systems): autoreferat k.f.-m.n. Saint-Petersburg, 2005, 14 p.

13. **Salton G.** *Avtomaticheskaja obrabotka, hranenie i poisk informacii* (Automatic processing, storage and retrieval of information), Moscow: *Sov. Radio*, 1973, 560 p.

14. **Salton G., Buckley C.** Term-weighting approaches in automatic text retrieval, *Information Processing & Management*. 1988, vol. 24, no. 5, pp. 513—523.

15. **Eprev A. S.** Avtomaticheskaja klassifikacija tekstovyh dokumentov (Automatic classification of text documents), *Matematicheskie struktury i modelirovanie*, 2010, no. 21, pp. 65—81 (in Russian).

16. **Jones S. K.** A statistical interpretation of term specificity and its application in retrieval, *Journal of documentation*, 1972, vol. 28, no. 1, pp. 11—21.

17. **Salton G.** Developments in automatic text retrieval, *Science*, 1991, vol. 253, no. 5023, pp. 974—980.

18. **Manning Ch. D., Raghavan P., Schutze H.** Scoring, term weighting and the vector space model, *Introduction to Information Retrieval,* 2008, vol. 100, pp. 504.

19. **Zagorul'ko Ju. A., Kononenko I. S., Kostov Ju. V., Sidorova E. V.** Klassifikacija delovyh pisem v sisteme dokumentooborota (Classification of business letters in the document management system), *Materialy mezhdunarodnoj nauchno-tehnicheskoj konferencii "Informacionnye sistemy i tehnologii"* (IST'2003). Novosibirsk, Izdatel'stvo NGTU, 2003, vol. 3, pp. 141—145 (in Russian).

20. **Norkin V., Keyzer M.** On stochastic optimization and statistical learning in reproducing kernel Hilbert spaces by Support Vector Machines (SVM), *Informatica*, 2009, vol. 20, pp. 273—292.

21. **Beuster G.** MIC — A System for Classification of Structured and Unstructured Texts. Diploma Thesis. University Koblenz, 2001.

22. **Chapelle O., Vapnik V., Bousquet O., Mukherjee S.** Choosing Multiple Parameters for Support Vector Machines, *Machine Learning*, 2002, vol. 46, no. 1, pp. 131—159.

23. **Landauer T. K., Foltz P., Laham D.** An Introduction to Latent Semantic Analysis, *Discourse Processes*, 1998, vol. 25, pp. 259—284, available at: http://lsa.colorado.edu/papers/dp1.LSAintro.pdf

24. **V'jugin V.** *Matematicheskie osnovy mashinnogo obuchenija i prognozirovanija* (Mathematical foundations of machine learning and prediction), MCMNO, 2014, 304 p. (in Russian).

25. **Kim H., Park H.** Nonnegative Matrix Factorization Based on Alternating Nonnegativity Constrained Least Squares and Active Set Method, *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 2, pp. 713—730, 2008. available at: http://www.cc.gatech.edu/~hpark/papers/simax-nmf.pdf

26. **Voevodin V. V., Kuznecov Ju. A.** *Matricy i vychislenija* (Matrix and calculations), Moscow, Nauka, 1984, 320 p. (in Russian).

27. **Deerwester S., Dumais S. T., Furnas G. W., Landauer T. K., Harshman R.** Indexing by Latent Semantic Analysis, *Journal of the American Society for Information Science*, 1990, vol. 41, no. 6, pp. 391—407.

28. **Hofmann T.** Probabilistic Latent Semantic Indexing, *Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in en:Information Retrieval (SIGIR-99)*, 1999, pp. 50—57.

29. **Hofmann T.** Probabilistic latent Semantic Snalysis, *In Proc. of Uncertainty in Artificial Intelligence,* UAI'99, Stockholm, 1999, pp. 289—296.

30. **Rish I.** An empirical study of the naive Bayes classifier, *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, 2001, available at: http://www.research.ibm.com/people/r/rish/papers/RC22230.pdf

31. **Chavan G. S., Manjare S., Hegde P., Sankh A.** A Survey of Various Machine Learning Techniques for Text Classification, *International Journal of Engineering Trends and Technology (IJETT),* 2014, vol. 15, no. 6, pp. 288—292.

32. **Breiman L., Friedman J. H., Olshen R. A., Stone C. J.** *Classification and regression trees*, Monterey, CA: Wadsworth & Brooks, Cole Advanced Books & Software, 1984.

33. **Quinlan J.** Induction of decision trees, *Machine Learning*, 1998, vol. 1, no. 1, pp. 81—106.

34. **Quinlan J.** *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993, 302 p.

35. **Ageev M. S., Zhuravlev S. V., Lamburt V. G.** Podgotovka Web-versij tradicionnyh izdanij (Web-versions of traditional media), *Otkrytye Sistemy*, 2000, no. 12, pp. 31—35 (in Russian).

36. **Ageev M. S., Dobrov B. V., Makarov-Zemljanskij N. V.** Metod mashinnogo obuchenija, osnovannyj na modelirovanii logiki rubrikatora (A machine learning technique based on modeling the logic of categories), *RCDL'2003 Jelektronnye biblioteki: perspektivnye metody i tehnologii, jelektronnye kollekcii: Pjataja vserossijskaja nauch. konf.* Saint-Petersburg, 2003, pp. 150—158. (in Russian).

37. **Ageev M., Dobrov B., Loukachevitch N.** Text Categorization Tasks for Large Hierarchial Systems of Categories, *SIGIR 2002 Workshop on Operational Text Classification Systems* / Eds. F. Sebastiani, S. Dumas, D. D. Lewis, T. Montgomery, I. Moulinier, Univ. of Tampere, 2002, pp. 49—52.

38. **Ageev M., Dobrov B., Makarov-Zemlyanskii N.** On-line Thematic and Metadata Analysis of Document Collection, *New Trends in Intelligent Information Processing and Web Mining'2004: Proceedings of the International Conference*, Springer, Advanced in Soft Computing. Zakopane, Poland, May 2004, pp. 279—286.

39. **Dumais S., Lewis D., Sebastiani F.** Report on the Workshop on Operational Text Classification Systems, *SIGIR-02, Tampere, Finland,* 2002, available at: http://www.sigir.org/forum/F2002/sebastiani.pdf.

40. **Rose T., Stevenson M., Whitehead M.** The Reuters Corpus Volume 1 — from Yesterday News to tomorrow's Language, *In Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas de Gran Canaria, May 2002, pp. 29—31.

41. **Wasson M.** Classification Technology at LexisNexis, *SIGIR 2001 Workshop on Operational Text Classification*, available at: http://www.daviddlewis.com/events/otc2001/presentations/otc01-wasson-paper.txt

42. **Valiant L. G.** A theory of the learnable, *Communications of the ACM*, 1984, vol. 27, no. 11, pp. 1134—1142.

43. **Breiman L.** Random forests, *Machine Learning*, 2001, vol. 45, no. 1, pp. 5—32.

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Е. В. Комиссарова*

Сдано в набор 11.05.2017 г. Подписано в печать 19.06.2017 г. Формат 60×88 1/8. Заказ РІ717
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**