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. 7
N 5
2016

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 004.41 DOI: 10.17587/prin.7.195-201

А. А. Ковалевский, аспирант, e-mail: morskoyzmey@gmail.com, **А. Н. Пустыгин,** канд. техн. наук, доц., e-mail: p2008an@rambler.ru, Челябинский государственный университет

Выделение признаков неделимых частей исходного текста с помощью универсального промежуточного и эквивалентного представлений

Предложен способ выделения признаков исходного текста, основанный на преобразовании его в промежуточное и далее эквивалентное представления, в целях последующей классификации частей текста по разработанной системе признаков. Предложен анализ программных проектов с открытым исходным текстом, основанный на задании признаков искомого текста без привязки к именам идентификаторов. Представлены статистические данные результатов кластеризации по группам признаков четырех проектов с открытым исходным кодом, имеющих попарно функционально схожее назначение.

Ключевые слова: *универсальное промежуточное представление, эквивалентное представление, статический анализ, открытый исходный код, классификация исходного текста, признаки исходного текста, поток управления*

Введение

Одним из подходов, упрощающих статический анализ и построение различных эквивалентных представлений исходных текстов на разных языках, является использование универсального промежуточного представления (УПП) [1]. При генерации УПП используется абстрактное синтаксическое дерево (AST) исходного кода, получаемое после обработки директив препроцессора и инстанцирования шаблонов. Затем УПП используется для генерации эквивалентного представления (ЭП), обеспечивающего решение задачи анализа.

Целью работы, результаты которой представлены в статье, были реализация прототипа утилиты для выделения набора признаков неделимых частей исходного текста программных проектов и демонстрация возможностей использования предложенного подхода в задачах анализа исходного текста.

Для заданной формулировки поставленной задачи прямых аналогов найти не удалось. Наиболее близким по тематике является метод *Support Vector Machines* (SVM), применяемый для классификации текстов, в том числе архивов исходного текста [2]. В статье [2] предлагается способ классификации исходных текстов на 11 категорий по обучающим наборам следующих признаков трех типов и их комбинаций: отдельных слов, лексических фраз и биграмм. В каждой категории были выделены 100 характерных признаков. В качестве обучающей выборки использовались базы открытого программного обеспечения (ПО). Такой подход с определенной точностью позволяет выполнять классификацию программного проекта в целом, но вряд ли применим для поставленной задачи классификации небольших отдельных частей исходного текста программ типа метод или блок. Тем более, если задача решается в целях определения недокументированных возможностей программного кода, которые могут быть легко скрыты от чисто текстового анализа.

Определения

Определение *1.* *Квант — неделимая часть исходного текста текущего уровня абстракции (например, класс для диаграммы классов, метод для графа потока, блок для тела функции).*

Определение 2. *Признак — именованный набор свойств, характеризующих квант исходного текста.*

Определение 3. *Шаблон признака — запись совокупности условий, выделяющих свойства признака кванта, которые накладываются на узел AST из УПП. При наложении условия на квант ЭП определяется степень соответствия кванта этому шаблону:*

- *полное соответствие, если все условия шаблона выполняются;*
- *неполное соответствие, если некоторые из условий шаблона не выполняются;*
- *несоответствие шаблону, если ни одно из условий шаблона не выполнено.*

Определение 4. *Дайджест — набор признаков, соответствующих кванту исходного текста.*

Определение 5. *Строгий дайджест — набор признаков, шаблоны которых полностью соответствуют кванту.*

Определение 6. *Полный дайджест — объединение строгого дайджеста и набора признаков, шаблоны которых соответствуют кванту не полностью.*

Определение 7. *Глоссарий — набор введенных признаков.*

Каждый квант (класс, метод) в ЭП ставится в соответствие набору условий шаблона каждого признака из глоссария. Каждому кванту присваивается набор признаков (дайджест), соответствующих ему полностью или частично.

Определение 8. *Группы квантов исходного текста формируются по строгому дайджесту: если строгие дайджесты квантов совпадают, то они включаются в одну группу. Внутри каждой группы выделяются подгруппы квантов, совпадающих по полному дайджесту. Можно проследить включения каждого из признаков в сформированные группы.*

Определение 9. *Кластер — объединение групп вокруг непустого ядра. Ядро, в свою очередь, это группа квантов, дайджест которой входит в дайджесты всех групп, включенных в кластер.*

Определение 10. *Совокупность кластеров — разбиение второго уровня. Фактически разбиение второго уровня является совокупностью кластерных ядер.*

Описание наборов данных предложенного прототипа

Алгоритм получения набора данных состоит из двух этапов преобразований. Сначала из УПП путем атрибутирования метками потока данных (операции над данными) и потока управления (вызовы, передача управления) получается эквивалентное представление UIRDCF. На втором этапе последовательно просматривается глоссарий и для каждой его записи выполняется модификация каждого кванта UIRDCF, если он соответствует шаблону признака из глоссария. Полученное эквивалентное представление назовем UDCFM.

Для тестирования использовались проекты с открытым исходным кодом pugixml [3], jsoncpp [4], lz4 [5] и zlib [6].

Генерация UDCFM выполняется с помощью инструмента MultiCrossAnalyzer [7] с подключением специального конфигурационного файла [8].

Была предложена тестовая система признаков (глоссарий), сформированная из эвристических соображений, соответствующих алгоритмическому смыслу описываемого кванта [9].

Глоссарий признаков описывается в текстовом файле в формате XML и разделен на две категории квантов: метод и класс. Каждая категория содержит набор шаблонов, соответствующих искомым признакам квантов, в виде блока условий:

```
<type name = "<имя признака>" title = "<описание признака>" [параметры признака] >
        <node name = "<имя тега>" [параметры условия] >
                <attr name = "<имя атрибута>" [параметры атрибута] />

        </node>

</type>
```

Каждый узел условия конкретизирует свойства искомого признака кванта.

Пример. В категории "метод" для признака "получатель поля" (getter) заданы следующие условия шаблона:

```
<node name = "MethodDecl">
        <attr name = "const" vb = "true" />
        <attr name = "argc" vi = "0"/>
</node>
<node name = "FieldRef" tags = "return" parent = "this" />
```

Что соответствует:

- метод является константным,
- метод не имеет аргументов,
- поле объекта this напрямую возвращается оператором return.

Атрибут tags второго условия определяет набор меток из эквивалентного представления UDCFM, которым должен обладать узел УПП.

Формат шаблонов позволяет записывать условия, требующие семантического анализа, который дополнительно выполняется утилитой на первом этапе обработки запроса. Результат этого анализа записывается в метаданные узлов дерева разбора, к которым относятся: принадлежность типу, принадлежность классу определения, узел дерева точки объявления. К метаданным добавляется информация UDCFM, которая включает в себя итеративно дополняемый список уже вычисленных признаков квантов на базе анализа предыдущих шаблонов.

Предложенная система признаков разбивает достаточно крупный проект на некоторое число уникальных групп квантов, т. е. групп методов или классов. Например, для проекта pugixml с 621 методом число групп методов составляет 200, а для проекта jsoncpp с 532 методами — 101 группу методов.

Примеры выделенных признаков квантов исходного текста

Для шаблона признака "функция-конвертер" (func-converter) в тестовом проекте pugixml утилита показывает методы, имеющие в названии составляющую "convert" (sic!) или характерный предлог "to" (табл. 1), что является симптоматичным совпадением, так как имена шаблонов не привязаны к именам идентификаторов. Признак "функция-конвертер" полностью соответствует 39 методам проекта и входит в 16 групп строгих дайджестов.

Таблица 1

Примеры методов признака "функция-конвертер" (файл pugixml.cpp)

Строка	Имя метода
6273	convert_number_to_boolean
6026	tolower_ascii
6332	convert_number_to_string
1386	convert_buffer

При поиске шаблона признака "проверяющая функция" (func-checker) в тестовом проекте pugixml находится функция "check_string_to_number_ format" (строка 6395, файл pugixml.cpp) с ключевым словом "check" в имени.

При поиске шаблона признака "проверяющий метод" (checker) в тестовом проекте jsoncpp находятся соответствующие методы, в названиях которых присутствует глагол "is" (табл. 2).

Таблица 2

Примеры методов с признаком "проверяющий метод" (файл json_value.cpp)

Строка	Имя метода
248	Value::CZString::isStaticString() const
1255	Value::isNull() const
1262	Value::isBool() const
1269	Value::isInt() const
1276	Value::isUInt() const
1292	Value::isDouble() const
1306	Value::isString() const

Например, метод Value::isBool в строке 1262 файла json_value.cpp сравнивает значение поля "type_" с константой "booleanValue" и возвращает булевый результат сравнения (табл. 3).

При поиске шаблона "сравнивающий метод" (comparator) в тестовом проекте pugixml находятся

Таблица 3

Исходный код метода признака "проверяющий метод"

Строка	Исходный текст
1261	bool
1262	Value::isBool() const
1263	{
1264	return type_ == booleanValue;
1265	}

методы, выполняющие функцию перегруженных операторов сравнения (табл. 4).

Например, метод "operator==" в строке 3738 файла pugixml.cpp сравнивает поле "_attr" передаваемого по ссылке параметра "r" объектного признака "xml_ attribute" и поле "_attr" данного объекта (табл. 5).

Использование системы выделенных признаков для задач поиска

В задаче, когда не вполне формализуются условия поиска, по дайджесту можно отсеивать целые группы квантов, заведомо не соответствующих поисковым критериям.

Например:

- при поиске методов, не имеющих определения (тела), можно исключить все группы квантов с признаком "имеет определение" (defined) (исключить 607 вхождений из 621 метода проекта pugixml);
- при поиске методов и функций, содержащих системные вызовы, можно исключить те, что не содержат вызовов вообще (!caller) (исключить 213 вхождений из 621 метода проекта pugixml), или те, что не содержат вызовов функций (!func-caller) (исключить 359 вхождений из 621 метода проекта pugixml);
- при поиске методов, модифицирующих поля объекта своего класса, можно отсеять все константные методы (исключить 160 вхождений из 621 метода проекта pugixml).

Частичные соответствия признаку позволяют выделять участки исходного текста, потенциально содержащие искомый код, если в глоссарии нет полностью соответствующего условиям поиска шаблона.

Поиск квантов по заданному набору признаков (дайджесту) делает возможным новый способ анализа программных проектов по исходному тексту с использованием УПП и его эквивалентного представления. Такой поиск позволяет фильтровать исходный текст, не используя имен идентификаторов и задавая только признаки искомого множества квантов исходного текста (в противовес текстовому поиску).

Приведем примеры предложенного анализа исходного текста с использованием эквивалентного представления UDCFM.

Для проекта pugixml были сформированы вопросы для поиска, представленные далее.

Таблица 4

Примеры методов с признаком "сравнивающий метод" (файл pugixml.cpp)

Строка	Имя метода
3738	bool xml_attribute::operator==(const xml_attribute& r) const
3743	bool xml_attribute::operator!=(const xml_attribute& r) const
3748	bool xml_attribute::operator<(const xml_attribute& r) const
3753	bool xml_attribute::operator>(const xml_attribute& r) const
3758	bool xml_attribute::operator<=(const xml_attribute& r) const
3763	bool xml_attribute::operator>=(const xml_attribute& r) const
3974	bool xml_node::operator==(const xml_node& r) const
3979	bool xml_node::operator!=(const xml_node& r) const
3984	bool xml_node::operator<(const xml_node& r) const
3989	bool xml_node::operator>(const xml_node& r) const
3994	bool xml_node::operator<=(const xml_node& r) const
3999	bool xml_node::operator>=(const xml_node& r) const

Таблица 5

Исходный код метода с признаком "сравнивающий метод"

Строка	Исходный текст
3738	bool xml_attribute::operator==(const xml_attribute& r) const
3739	{
3740	return (_attr == r._attr);
3741	}

1. Найти константные методы, обрабатывающие аргументы в цикле.

Синтаксис запроса: const, method, cycled-arg-processor.

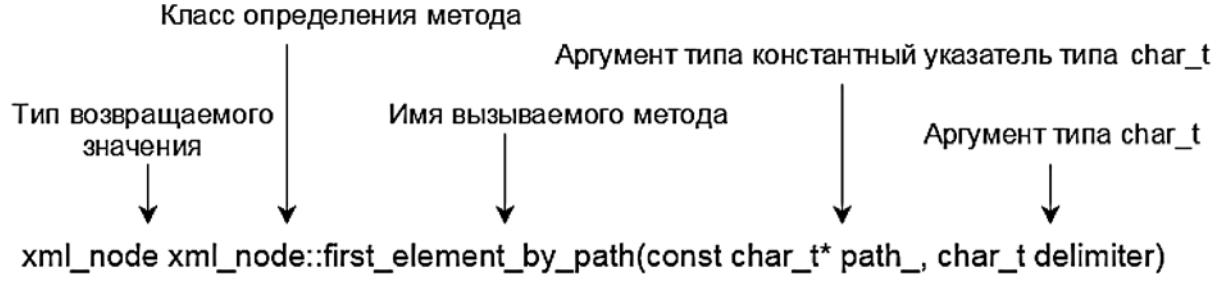
Результат: строка 4550 в файле pugixml.cpp (см. рисунок).

2. Найти определения функций, содержащих несколько прямых рекурсивных вызовов.

Синтаксис запроса: function, defined, many-recursive

Результат для файла pugixml.cpp представлен в табл. 6.

3. Найти методы, которые устанавливают значения нескольких полей, копируя значения аргументов или одного аргумента.



Результат выполнения запроса

Таблица 6

Примеры функций, содержащих несколько прямых рекурсивных вызовов (файл pugixml.cpp)

Строка	Имя метода
3079	void node_output(xml_buffered_writer& writer, const xml_node& node, const char_t* indent, unsigned int flags, unsigned int depth)
6551	const char_t* namespace_uri(const xpath_node& node)
6682	xpath_variable* new_xpath_variable(xpath_value_type type, const char_t* name)
6709	void delete_xpath_variable(xpath_value_type type, xpath_variable* var)
5619	template <typename I, typename Pred> void sort(I begin, I end, const Pred& pred)

Таблица 7

Примеры методов, устанавливающих значения нескольких полей, копируя значения аргумента(ов) (файл pugixml.cpp)

Строка	Заголовок определения метода
421	void* xml_allocator::allocate_memory_oob(size_t size, xml_memory_page*&out_page)
1583	void push(char_t*& s, size_t count)
2853	void write(const char_t* data, size_t length)
2899	void write(char_t d0)
2907	void write(char_t d0, char_t d1)
2916	void write(char_t d0, char_t d1, char_t d2)
2926	void write(char_t d0, char_t d1, char_t d2, char_t d3)
2937	void write(char_t d0, char_t d1, char_t d2, char_t d3, char_t d4)
2949	void write(char_t d0, char_t d1, char_t d2, char_t d3, char_t d4, char_t d5)
5682	void* allocate_nothrow(size_t size)

Синтаксис запроса: common-setter, defined, method.

Результат для файла pugixml.cpp представлен в табл. 7.

4. Найти методы, которые вызывают методы своего класса, при этом не вызывают методы других классов или функций.

Синтаксис запроса: in-caller, !out-caller, !func-caller.

Результат для файла pugixml.cpp представлен в табл. 8.

Объем исходного текста проекта pugixml составляет 300 Кбайт, при этом заранее неизвестны имена искомых квантов. Таким образом, поиск по указанным признакам без использования прототипа утилиты представляет собой полный просмотр исходного текста вручную во всех файлах по всем 11 500 строкам.

Таблица 8

Примеры методов, вызывающих методы своего класса и не вызывающих методы других классов или функций (файл pugixml.cpp)

Строка	Заголовок определения метода
4383	xml_node xml_node::append_child(const char_t* name_)
4392	xml_node xml_node::prepend_child(const char_t* name_)
4401	xml_node xml_node::insert_child_after(const char_t* name_, const xml_node& node)
4410	xml_node xml_node::insert_child_before(const char_t* name_, const xml_node& node)
...	и еще 61 метод

Выделение характерных групп признаков квантов для проектов разной природы

Разбиение второго уровня выделяет наборы признаков, характерные для определенных категорий квантов (кластеров), и может применяться для сокращения числа сущностей анализируемого проекта.

Алгоритм разбиения построен на полном переборе вариантов кластеризации с эвристически подобранными параметрами:

- кластеры обеспечивают покрытие 90 % множества всех групп проекта;
- число поэлементных пересечений для всех ядер в сумме не превышает пяти.

Для проекта pugixml были получены следующие варианты *кластеризации*.

1. Кластер "Статические методы с циклической обработкой аргументов" (arg-processor, cycled-arg-processor, cycled, static, defined, method), который объединяет 4 различные группы квантов из 200, в которых участвуют 7 из 621 вхождений.

2. Кластер "Методы, содержащие рекурсивный вызов" (recursive, in-caller, caller, defined, method), который объединяет 9 различных групп квантов, в которых участвуют 29 вхождений.

3. Кластер "Константный метод, возвращающий значение поля объекта данного класса" (getter, const, defined, method), который объединяет 6 различных групп квантов, в которых участвуют 45 вхождений.

Для проектов процедурной парадигмы характерны более крупные кластеры групп. Например, для проекта lz4 было получено всего четыре перечисленных далее кластера, объединяющих 95 % групп (18 из 19).

1. Кластер "Функция, содержащая вызовы других функций" (func-caller, caller, defined, function). Число групп, входящих в кластер: 13 из 19. Число вхождений методов в кластер: 162 из 176 (92 %).

2. Кластер "Функция-конвертер, обрабатывающая переданные аргументы" (func-converter, arg-processor, defined, function). Число групп, входящих в кластер: 6. Число вхождений методов в кластер: 47 из 176 (27 %).

3. Кластер "Функция-обработчик аргументов, содержащая оператор switch" (arg-processor, selector, defined, function). Число групп, входящих в кластер: 3 из 19. Число вхождений методов в кластер: 15 из 176 (9 %).

4. Кластер "Функция-конвертер, обрабатывающая переданные аргументы в цикле, содержащая вложенные циклы и оператор switch, вызывающая другие функции" (func-converter, arg-processor, cycled-arg-processor, complex-cycled, cycled, selector, func-caller, caller, defined, function). Число групп, входящих в кластер: 1 из 19. Число вхождений методов в кластер: 12 из 176 (7 %).

Заключение

На основе УПП [1] и дополненного эквивалентного представления UIRDCF [7] была введена тестовая система признаков, условий и правил их выделения [9] и разработан прототип утилиты, который выделяет признаки квантов исходного текста согласно применению набора шаблонов (глоссария) над эквивалентным представлением UDCFM, группирует кванты, строит разбиения по заданным параметрам, тем самым выделяя нетривиальные множества признаков, а также позволяет выполнять поиск квантов по заданным признакам.

Предложенный подход выделения признаков показал свою применимость для задач поиска квантов исходного текста по сложным или плохо формулируемым условиям. Целью такого поиска может быть поиск недокументированных возможностей и потенциально ошибочных участков кода при введении соответствующих признаков и категорий для искомых частей исходного текста (классов, методов, блоков).

Список литературы

1. **Ошнуров Н. А., Пустыгин А. Н., Ковалевский А. А.** Построение универсального промежуточного представления исходных текстов на языках C++ и C# // Доклады Томского гос. ун-та систем управления и радиоэлектроники. 2014. Т. 33, № 3. С. 135—139.
2. **What's** the Code? Automatic Classification of Source Code Archives. URL: https://clgiles.ist.psu.edu/papers/KDD-2002-whatsthecode.pdf
3. **Pugixml** v1.2 — Light-weight, simple and fast XML parser for C++ with XPath support. URL: http://pugixml.org/2012/05/01/pugixml-1.2-release.html
4. **Jsoncpp** v0.5.0 C++ JSON parser. URL: http://sourceforge.net/projects/jsoncpp/files/jsoncpp/0.5.0/.
5. **LZ4** r131 — Extremely fast compression. URL: https://github.com/Cyan4973/lz4.
6. **Zlib** 1.2.8 — A Massively Spiffy Yet Delicately Unobtrusive Compression Library. URL: http://www.zlib.net/
7. **Ковалевский А. А., Пустыгин А. Н., Ошнуров Н. А.** Построение эквивалентного представления исходных текстов программ в форме, пригодной для выполнения анализов потока данных в потоке управления // Известия Юго-Западного гос. ун-та, Серия управление, вычислительная техника, информатика. Медицинское приборостроение. 2015. № 1 (14). С. 28—34.
8. **Конфигурационный** файл для построения UDCFM. URL: https://yadi.sk/d/NZufKwwtkTZtN
9. **Глоссарий** признаков квантов исходного текста. URL: https://yadi.sk/d/fSLygTHMkTa36

Source Code Indivisible Parts Symptoms Extraction Based on Universal Intermediate and Further Equivalent Representations

A. A. Kovalevskiy, morskoyzmey@gmail.com, **A. N. Pustygin**, p2008an@ya.ru, Chelyabinsk State University, Chelyabinsk, 454001, Russian Federation

Corresponding author:

Kovalevskiy Aleksey A., Postgraduate Student, Chelyabinsk State University, Chelyabinsk, 454001, Russian Federation,
e-mail: morskoyzmey@gmail.com

Received on December 24, 2015
Accepted on February 15, 2016

In this article a source code symptoms extraction method is proposed, based on source code transformation to universal intermediate and then to an equivalent representation. The goal of the transformation is the subsequent classification of text parts in accordance with the developed types system. The analysis of open source projects is based on symptoms of the parts without binding to identifier names. For four open source projects there are statistical results to a groups symptoms clustering accordingly with a having functionally in pairs similar appointment submitted. The aim of this study was as an implementation of the source code selection given set of functional attributes parts as well as demonstration of the using possibilities of the proposed approach to the source text analysis. The test system types (glossary) was created according to heuristic reasons to described quantum algorithmic meaning.

Keywords: *universal intermediate representation, equivalent representation, statical analysis, open source software, source code classification, source code indivisible symptoms*

For citation:

Kovalevskiy A. A., Pustygin A. N. Source Code Indivisible Parts Symptoms Extraction Based on Universal Intermediate and Further Equivalent Representations, *Programmnaya Ingeneria*, 2016, vol. 7, no. 5, pp. 195—201.

DOI: 10.17587/prin.7.195-201

References

1. **Oshnurov N. A., Pustygin A. N., Kovalevskiy A. A.** Postroenie universal'nogo promezhutochnogo predstavlenija ishodnyh tekstov na jazykah C++ i C# (Universal intermediate representation construction of source code in C++ and C# languages). *Doklady Tomskogo Gos. Un-ta Sistem Upravlenija i Radiojelektroniki*, 2014, no. 3, pp. 135—139, available at: http://www.tusur.ru/filearchive/reports-magazine/2014-33-3/23.pdf (in Russian).

2. **What's** the Code? Automatic Classification of Source Code Archives, available at: https://clgiles.ist.psu.edu/papers/KDD-2002-whatsthecode.pdf

3. **Pugixml** v1.2 — Light-weight, simple and fast XML parser for C++ with XPath support, available at: http://pugixml.org/2012/05/01/pugixml-1.2-release.html.

4. **Jsoncpp** v0.5.0 C++ JSON parser, available at: http://sourceforge.net/projects/jsoncpp/files/jsoncpp/0.5.0/

5. **LZ4** r131 — Extremely fast compression, available at: https://github.com/Cyan4973/lz4

6. **Zlib** 1.2.8 — A Massively Spiffy Yet Delicately Unobtrusive Compression Library, available at: http://www.zlib.net/

7. **Kovalevskiy A. A., Pustygin A. N., Oshnurov N. A.** Postroenie jekvivalentnogo predstavlenija ishodnyh tekstov programm v forme, prigodnoj dlja vypolnenija analizov potoka dannyh v potoke upravlenija (Construction of equivalent representations of applications source code in form of data propagation in control flow graph), *Izvestija Jugo-Zapadnogo Gos. Un-ta. Serija: Upravlenie, Vychislitel'naja Tehnika, Informatika. Medicinskoe Priborostroenie*, 2015, no. 1 (14), pp. 28—34 (in Russian).

8. **Building** UDCFM configuration file, available at: https://yadi.sk/d/NZufKwwtkTZtN

9. **Source** code parts types glossary, available at: https://yadi.sk/d/fSLygTHMkTa36

УДК 519.683.2 DOI: 10.17587/prin.7.202-210

М. С. Чушкин, аспирант, e-mail: chushkinm@rambler.ru, Институт систем информатики им. А. П. Ершова, г. Новосибирск

Система дедуктивной верификации предикатных программ

Представлен новый метод генерации формул тотальной корректности рекурсивных программ без циклов и указателей. Реализована система дедуктивной верификации, генерирующая формулы корректности. Разработана система правил для упрощения процесса генерации формул корректности. Истинность полученных формул проверена с помощью SMT-решателя CVC3 и в системе PVS.

Ключевые слова: *предикатное программирование, тотальная корректность программы, дедуктивная верификация, система автоматического доказательства PVS, SMT-решатель CVC3*

Введение

Язык предикатного программирования P определяет класс программ-функций, не взаимодействующих с внешним окружением [1]. Бесконечное исполнение таких программ бессмысленно, поэтому корректная программа на языке P всегда должна завершать свое исполнение. В языке P запрещены циклы и указатели.

На базе аппарата формальной операционной семантики разработан метод дедуктивной верификации предикатных программ. Определена формула тотальной корректности предикатной программы относительно своей спецификации, представленной предусловием и постусловием. Разработана система правил [2], позволяющая декомпозировать формулу тотальной корректности на набор простых и коротких формул, упрощая тем самым последующее доказательство. Корректность правил доказана в системе PVS [3].

Система дедуктивной верификации включает: генератор формул корректности, транслятор формул во внутреннее представление SMT-решателя CVC3, транслятор формул на язык спецификаций системы PVS. Генератор формул корректности строит формулы корректности программы, используя систему правил.

В процессе трансляции программы для каждого вхождения переменной (или выражения) реализуется проверка соответствия типа переменной (или выражения) типу позиции вхождения. Контроль соответствия типов не всегда можно провести статически при трансляции. При наличии ошибки несоответствия типов результат исполнения программы может оказаться неверным даже в том случае, когда доказана тотальная корректность программы. По этой причине для гарантии корректности программы необходимо проводить полный контроль типов.

В трансляторе с языка P реализована статическая проверка совместимости типов. Условия совместимости типов, которые транслятор не может проверить статически, добавляются к формулам корректности программы.

Все формулы проходят проверку в SMT-решателе CVC3. Формулы, истинность которых не удалось проверить с помощью решателя, оформляются в виде теории для системы PVS.

Система верификации успешно прошла апробацию в рамках университетского курса "Формальные методы в описании зыков и систем программирования".

Обзор работ

Методы верификации. Дедуктивная верификация программных систем в основном базируется на логике Хоара [4], определяющей набор правил вывода (аксиом) для троек Хоара. Последовательное применение правил позволяет получить набор формул — условий частичной корректности императивной программы.

Доказательство завершения программы не реализуется в классической логике Хоара. Позже метод был расширен. Для доказательства завершения исполнения циклов и рекурсивных вызовов применяется аппарат вполне упорядоченных множеств [5]. Каждой рекурсивной функции и циклу приписывается переменная из вполне упорядоченного множества, значение которой должно строго убывать с каждой итерацией или вызовом.

Метод дедуктивной верификации предикатных программ отличается от метода Хоара. Он реализован для невзаимодействующих программ, испол-

нение которых должно всегда завершаться. Метод реализует доказательство тотальной корректности программы, а не частичной, как в методе Хоара.

В предикатном программировании нет циклов, вместо них используются рекурсивные предикаты. Верифицировать рекурсивный предикат проще, чем цикл. Построение инварианта цикла — нетривиальная задача. В предикатном программировании роль инварианта выполняет предусловие рекурсивного предиката, которое существенно проще инварианта цикла.

В методе Хоара для доказательства тотальности рекурсивных программ применяется аппарат вполне упорядоченных множеств. В предлагаемом автором подходе используется функция меры, строго убывающая на аргументах рекурсивных вызовов. Доказательство с использованием функции меры удобнее, чем доказательство с использованием переменных на упорядоченных множествах.

В лаборатории системного программирования Института систем информатики им. А. П. Ершова был разработан простой универсальный механизм обобщения правил для рекурсивных вызовов. Правила для исключения квантора существования значительно упрощают процесс построения формул корректности.

Слабейшее предусловие Дейкстры [6] определяет условие тотальной корректности программы. Слабейшее предусловие строится применением к программе преобразователя предикатов `wp(S, R)` — отображения из множества программ и постусловий во множество слабейших предусловий. Его применение к некоторой программе `S` приводит к разделению этой программы на команды и последующему применению преобразователя предикатов к ним. В результате этого процесса получается формула — слабейшее предусловие.

Классический метод Хоара неприменим для программ с указателями. Метод separation logic [7] расширяет семантику императивного языка метода Хоара, описывая статическую и динамическую память. Язык расширяется командами для выделения и высвобождения динамической памяти. В работе [7] была предложена концепция разделенной памяти, в которой память, выделенная одной структуре данных, не может пересекаться с памятью, выделенной для другой структуры данных. Одна из ключевых идей метода — это расширение стандартной логики Хоара оператором *, где `P * Q` означает, что динамическая память может быть разделена на две непересекающиеся части, для каждой из которых истинно одно из условий `P` или `Q`.

Поведение программы, оперирующей указателями, анализировать трудно. Методы анализа программ, в том числе и метод separation logic, базируются на различных моделях организации динамической памяти. В предикатном программировании нет указателей, вместо них используются алгебраические типы данных. Это существенно упрощает процесс доказательства корректности программы.

Доказательство условий совместимости типов. Язык предикатного программирования P не единственный, где условия семантической корректности формализуются в виде логических формул. Аналогичный метод используется в системе PVS, где семантический контроль нетривиальных конструкций осуществляется через доказательство соответствующих формул корректности (TCCs — *type correctness conditions*).

Семантический анализатор автоматически строит условие совместимости типов. В дальнейшем система пытается автоматически их доказать. Если автоматически это сделать не удается, то пользователь должен доказать эти условия средствами системы PVS [8].

Предикатное программирование

Предикатная программа является предикатом в форме вычислимого оператора. Предикатная программа не взаимодействует с внешним окружением программы. Точнее, перед началом программы возможен ввод данных и лишь по ее завершению — вывод результатов. Программа обязана всегда завершаться, поскольку бесконечно работающая и невзаимодействующая программа бесполезна. Следовательно, программа определяет функцию, вычисляющую по набору аргументов некоторый набор результатов.

Базисные конструкции языка предикатного программирования P. Программа на языке предикатного программирования P состоит из набора определений предикатов. Определение предиката имеет следующий вид:

```
<имя предиката>
(<список аргументов>: <список результатов>)
pre <предусловие>
{
<оператор>
}
post <постусловие>
measure <мера>
```

Необязательные конструкции предусловие и постусловие являются формулами на языке исчисления предикатов; они используются для улучшения понимания программ и для дедуктивной верификации [9—12]. Для любых значений аргументов, удовлетворяющих предусловию, значения результатов, в случае завершения оператора, должны удовлетворять постусловию. Мера — выражение натурального типа от аргументов предиката. Значение меры должно строго убывать для рекурсивных вызовов.

Ниже представлены основные конструкции языка P: оператор присваивания, блок (оператор суперпозиции), параллельный оператор, условный оператор, вызов программы и описание переменных, используемое для аргументов, результатов и локальных переменных.

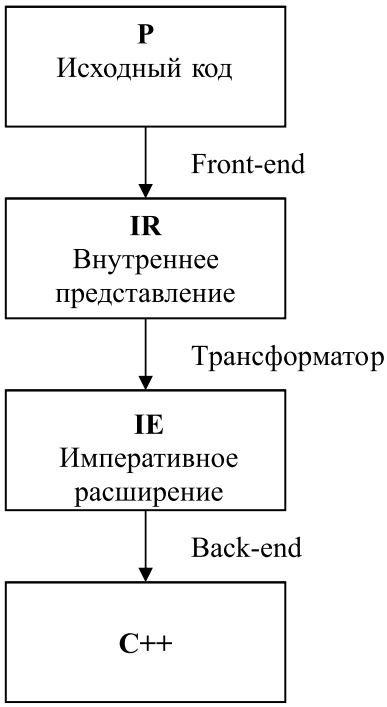


Рис. 1. Система предикатного программирования

```
<переменная> = <выражение>
{ <оператор 1>; <оператор 2> }
<оператор 1> || <оператор 2>
if (<логическое выражение>) <оператор 1>
else <оператор 2>
<имя предиката> (<аргументы>: <результаты>)
<тип> <список имен переменных>
```

В языке P запрещены такие языковые конструкции, как циклы и указатели, серьезно усложняющие анализ программы. Полное описание языка P дано в работе [1].

Императивное расширение языка P. Императивное расширение языка P содержит дополнительные языковые конструкции: прерывание исполнения цикла (`break`), циклы `for` и `while`, метку и оператор безусловного перехода на метку. Семантика циклов `for` и `while` соответствует семантике языка C++.

Эти конструкции возникают в программе в результате проведения трансформаций предикатной программы. Использование этих конструкций в исходной программе недопустимо.

Система предикатного программирования. Система предикатного программирования осуществляет трансляцию программы с языка P на язык C++ (рис. 1). Синтаксический анализатор выполняет разбор кода программы на языке P и формирует ее образ во внутреннем представлении.

Метод дедуктивной верификации предикатных программ

Семантика языка предикатного программирования P. *Предикатная программа* `H(x: y)` с аргументами `x` и результатами `y` есть предикат в форме вычислимого оператора. *Операционную семантику* программы `H(x: y)` определим в виде предиката:

$\mathcal{R}$`(H)(x, y)` ≅ для значения набора `x` исполнение программы `H` всегда завершается и существует исполнение программы, при котором результатом вычисления является значение набора `y`.

Минимальный полный *базис предикатных программ* определен в виде языка P_0. В таблице представлен полный базис вычислимых предикатов и соответствующих им операторов.

В таблице `x`, `y` и `z` — разные непересекающиеся наборы переменных. Набор `x` может быть пустым, наборы `y` и `z` не пусты; `B` и `C` — имена предикатов, `A` и `D` — имена переменных предикатного типа. Набор $x^{\sim}$ составлен из набора переменных `x` с возможным добавлением имен предикатных программ.

Для языка P_0 построена формальная операционная семантика и доказано тождество $\mathcal{R}$`(H) = H` [13]. На базе языка P_0 последовательным расширением и сохранением тождества $\mathcal{R}$`(H) = H` построен язык предикатного программирования P [14]:

$$P_0 \to P_1 \to P_2 \to P_3 \to P.$$

Вычислимые предикаты и их программная форма

Предикат	Программная форма
H(x: y) ≅ ∃z. B(x: z) & C(z: y)	H(x: y) { B(x: z); C(z: y) }
H(x: y, z) ≅ B(x: y) & C(x: z)	H(x: y, z) { B(x: y) || C(x: z) }
H(x: y) ≅ (e ⇒ B(x: y)) & (¬e ⇒ C(x: y))	H(x: y) { **if** (e) B(x: y) **else** C(x: y) }
H(x: y) ≅ B($x^{\sim}$: y)	H(x: y) { B($x^{\sim}$: y) }
H(A, x: y) ≅ A(x: y)	H(A, x: y) { A(x: y) }
H(x: D) ≅ ∀y,z. D(y: z) ≡ B(x, y: z)	H(x: D) { D(y: z) {B(x, y: z)} }
H(A, x: D) ≅ ∀y,z. D(y: z) ≡ A(x, y: z)	H(A, x: D) { D(y: z) {A(x, y: z)} }

Корректность программы. Рассмотрим определение предиката следующего вида:

```
A(X x:Y y)
pre P(x)
{ S(x: y) }                    (1)
post Q(x, y)
measure m(x)
```

Тотальная (или полная) корректность программы (1) определяется истинностью следующей формулы:

$$\mathtt{Corr(S,\ P,\ Q)(x)} \cong \forall \mathtt{x}\ \mathtt{P(x)} \Rightarrow [\forall \mathtt{y}\ \mathcal{R}\mathtt{(S)(x,\ y)} \Rightarrow \mathtt{Q(x,\ y)}] \ \&\ \exists \mathtt{y}\ \mathcal{R}\mathtt{(S)(x,\ y)} \quad (2)$$

Здесь $\forall \mathtt{y}(\mathcal{R}\mathtt{(S)(x,\ y)} \Rightarrow \mathtt{Q(x,\ y)})$ является условием частичной корректности, т. е. условием того, что спецификация истинна при завершении программы. Подформула $\exists \mathtt{y}\ \mathtt{R(S)(x,\ y)}$ определяет условие завершения исполнения оператора S(x: y).

Далее термин "корректность" будем использовать в смысле тотальной корректности.

Система правил вывода формул корректности. Используя формулу тотальной корректности, можно автоматически построить формулу корректности для некоторого оператора S(x: y). Итоговая формула корректности будет длинной и сложной даже для коротких программ; она будет намного длиннее программы S(x: y). Специализация формулы тотальной корректности для разных видов операторов позволяет провести декомпозицию сложной формулы корректности к набору более коротких и простых формул.

Определяется система правил вывода условий корректности для различных операторов. Доказательства правил приведены в работах [2, 14]. В дополнение к этому формальные доказательства корректности правил проведены в системе автоматического доказательства PVS; см. доказательства корректности правил в формате PVS [3].

Предположим, что наборы переменных x, y, z и v не пересекаются, а множества x и v могут быть пустыми. В таком случае, корректны следующие правила:

$$\mathbf{QP:}\ \frac{\mathtt{Corr}\left(\mathtt{B},\mathtt{P},\mathtt{Q_B}\right)\left(\mathtt{x}\right);\mathtt{Corr}\left(\mathtt{C},\mathtt{P},\mathtt{Q_C}\right)\left(\mathtt{x}\right);}{\mathtt{Corr}\left(\mathtt{B}\left(\mathtt{x:y}\right)||\mathtt{C}\left(\mathtt{x:z}\right),\mathtt{P},\mathtt{Q_B}\left(\mathtt{x,y}\right)\&\mathtt{Q_C}\left(\mathtt{x,z}\right)\right)\left(\mathtt{x}\right)},$$

$$\mathtt{P}\left(\mathtt{x}\right) \rightarrow \exists \mathtt{z}, \mathtt{v}\mathtt{R}\left(\mathtt{B}\right)\left(\mathtt{x},\left(\mathtt{z},\mathtt{v}\right)\right);$$

$$\mathbf{QS:}\ \frac{\forall \mathtt{z}\mathtt{Corr}\left(\mathtt{C},\left(\mathtt{P},\left(\mathtt{x}\right)\&\exists \mathtt{z}, \mathtt{v}\mathtt{R}\left(\mathtt{B}\right)\left(\mathtt{x}\left(\mathtt{z},\mathtt{v}\right)\right)\right),\mathtt{Q}\right)\left(\mathtt{x},\mathtt{z}\right);}{\mathtt{Corr}\left(\mathtt{B}\left(\mathtt{x:z,v}\right);\mathtt{C}\left(\mathtt{x,z:y}\right),\mathtt{P},\mathtt{Q}\right)\left(\mathtt{x}\right)},$$

$$\mathtt{Corr*}\left(\mathtt{B},\mathtt{PB},\mathtt{QB}\right)\left(\mathtt{x}\right);\mathtt{P}\left(\mathtt{x}\right) \rightarrow \mathtt{P_B^*}\left(\mathtt{x}\right);$$

$$\mathbf{QSB:}\ \frac{\forall \mathtt{z}\mathtt{Corr}\left(\mathtt{C},\lambda \mathtt{x},\mathtt{z}.\left(\mathtt{P}\left(\mathtt{x}\right)\&\mathtt{Q_B}\left(\mathtt{x,z}\right)\right),\mathtt{Q}\left(\mathtt{x,z}\right)\right);}{\mathtt{Corr}\left(\mathtt{B}\left(\mathtt{x:z,v}\right);\mathtt{C}\left(\mathtt{x,z:y}\right),\mathtt{P},\mathtt{Q}\right)\left(\mathtt{x}\right)}$$

$$\mathbf{QC:}\ \frac{\begin{array}{l}\mathtt{Corr}\left(\mathtt{B},\lambda \mathtt{x}.\mathtt{P}\left(\mathtt{x}\right)\&\mathtt{E}\left(\mathtt{x}\right),\mathtt{Q}\right)\left(\mathtt{x}\right);\\ \mathtt{Corr}\left(\mathtt{C},\lambda \mathtt{x}.\mathtt{P}\left(\mathtt{x}\right)\&\neg\mathtt{E}\left(\mathtt{x}\right),\mathtt{Q}\right)\left(\mathtt{x}\right);\end{array}}{\mathtt{Corr}\left(\mathbf{if}\left(\mathtt{E}\left(\mathtt{x}\right)\right)\mathtt{B}\left(\mathtt{x:y}\right)\mathbf{else}\mathtt{C}\left(\mathtt{x:y}\right)\right)\left(\mathtt{x}\right)}.$$

Используя эти правила, доказательства корректности оператора суперпозиции, параллельного оператора и условного оператора можно свести к доказательству корректности их подоператоров B и C.

В правилах, описанных ниже, если подоператор B является рекурсивным вызовом, то посылка Corr*(B, …...) опускается, а $\mathtt{P_B^*(x)}$ заменяется на $\mathtt{P_B(x)}$ & m(x) < m(z), где z обозначает аргументы предиката B. Если же подоператор B не является рекурсивным вызовом, то Corr* и P* обозначают просто Corr и P. Вхождения Corr* и P* для подоператора C трактуются аналогично:

$$\mathbf{RP:}\ \frac{\begin{array}{l}\mathtt{Corr*}\left(\mathtt{B},\mathtt{P_B},\mathtt{Q_B}\right)\left(\mathtt{x}\right);\mathtt{Corr*}\left(\mathtt{C},\mathtt{P_C},\mathtt{Q_C}\right)\left(\mathtt{x}\right);\\ \forall \mathtt{y},\mathtt{z}\left(\mathtt{Q_B}\left(\mathtt{x,y}\right)\ \&\ \mathtt{Q_C}\left(\mathtt{x,y}\right) \rightarrow \mathtt{Q}\left(\mathtt{x,y,z}\right)\right);\\ \mathtt{P}\left(\mathtt{x}\right) \rightarrow \mathtt{P_B^*}\left(\mathtt{x}\right)\ \&\ \mathtt{P_C^*}\left(\mathtt{x}\right);\end{array}}{\mathtt{Corr}\left(\mathtt{B}\left(\mathtt{x:y}\right)\ ||\ \mathtt{C}\left(\mathtt{x:z}\right),\mathtt{P},\mathtt{Q}\right)\left(\mathtt{x}\right)},$$

$$\mathtt{Corr*}\left(\mathtt{B},\mathtt{P_B},\mathtt{Q_B}\right)\left(\mathtt{x}\right);\forall \mathtt{z}\mathtt{Corr*}\left(\mathtt{C},\mathtt{P},\mathtt{Q_C}\right)\left(\mathtt{x,z}\right);$$

$$\forall \mathtt{z},\mathtt{v},\mathtt{y}\left(\mathtt{P}\left(\mathtt{x}\right)\ \&\ \mathtt{Q_B}\left(\mathtt{x,z,v}\right)\ \&\ \mathtt{Q_C}\left(\mathtt{x,z,y}\right) \rightarrow \mathtt{Q}\left(\mathtt{x,z}\right)\right)$$

$$\forall \mathtt{z},\mathtt{v}\left(\mathtt{P}\left(\mathtt{x}\right)\ \&\ \mathtt{Q_B}\left(\mathtt{x,z,v}\right) \rightarrow \mathtt{P_C^*}\left(\mathtt{x,z}\right)\right);$$

$$\mathbf{RS:}\ \frac{\mathtt{P}\left(\mathtt{x}\right) \rightarrow \mathtt{P_B^*}\left(\mathtt{x}\right);}{\mathtt{Corr}\left(\mathtt{B}\left(\mathtt{x:z,v}\right);\mathtt{C}\left(\mathtt{x,z:y}\right),\mathtt{P},\mathtt{Q}\right)\left(\mathtt{x}\right)},$$

$$\mathbf{RC:}\ \frac{\begin{array}{l}\mathtt{Corr*}\left(\mathtt{B},\mathtt{P_B},\mathtt{Q_B}\right)\left(\mathtt{x}\right);\mathtt{Corr*}\left(\mathtt{C},\mathtt{P_C},\mathtt{Q_C}\right)\left(\mathtt{x}\right);\\ \mathtt{P}\left(\mathtt{x}\right)\ \&\ \mathtt{E} \rightarrow \mathtt{P_B^*}\left(\mathtt{x}\right);\mathtt{P}\left(\mathtt{x}\right)\neg\mathtt{E} \rightarrow \mathtt{P_C^*}\left(\mathtt{x}\right);\\ \forall \mathtt{y}\left(\mathtt{P}\left(\mathtt{x}\right)\ \&\ \mathtt{E}\ \&\ \mathtt{Q_B}\left(\mathtt{x,y}\right) \rightarrow \mathtt{Q}\left(\mathtt{x,y}\right)\right);\\ \forall \mathtt{y}\left(\mathtt{P}\left(\mathtt{x}\right)\ \&\ \neg\mathtt{E}\ \&\ \mathtt{Q_C}\left(\mathtt{x,y}\right) \rightarrow \mathtt{Q}\left(\mathtt{x,y}\right)\right);\end{array}}{\mathtt{Corr}\left(\mathbf{if}\left(\mathtt{E}\left(\mathtt{x}\right)\right)\mathtt{B}\left(\mathtt{x:y}\right)\mathbf{else}\mathtt{C}\left(\mathtt{x:y}\right),\mathtt{P},\mathtt{Q}\right)\left(\mathtt{x}\right)}.$$

Описанное ниже правило **RB** — это специализация правила **RS**. Вызов предиката B(x: z) может быть записан как z = B(x). Таким образом, можно использовать конструкцию C(B(x): y) как эквивалент оператора суперпозиции B(x, z); C(z: y).

$$\mathbf{RB:}\ \frac{\begin{array}{l}\forall \mathtt{z}\mathtt{Corr*}\left(\mathtt{C},\mathtt{P_C},\mathtt{Q_C}\right)\left(\mathtt{z}\right);\\ \mathtt{P}\left(\mathtt{x}\right) \rightarrow \mathtt{P_B}\left(\mathtt{x}\right)\ \&\ \mathtt{P_C^*}\left(\mathtt{B}\left(\mathtt{x}\right)\right);\\ \forall \mathtt{y}\left(\mathtt{P}\left(\mathtt{x}\right)\ \&\ \mathtt{Q_C}\left(\mathtt{B}\left(\mathtt{x}\right),\mathtt{y}\right) \rightarrow \mathtt{Q}\left(\mathtt{x,y}\right)\right);\\ \forall \mathtt{x},\mathtt{z}_1,\mathtt{z}_2\mathtt{P_B}\left(\mathtt{x}\right)\ \&\ \mathtt{R}\left(\mathtt{B}\right)\left(\mathtt{x},\mathtt{z}_1\right)\ \&\ \mathtt{R}\left(\mathtt{B}\right)\left(\mathtt{x},\mathtt{z}_2\right) \rightarrow \mathtt{z}_1 = \mathtt{z}_2;\end{array}}{\mathtt{Corr}\left(\mathtt{C}\left(\mathtt{B}\left(\mathtt{x}\right):\mathtt{y}\right),\mathtt{P},\mathtt{Q}\right)\left(\mathtt{x}\right)}.$$

Представленные правила принципиально упрощают формулы корректности рекурсивных программ.

Последовательное применение описанных выше правил к исходной формуле тотальной корректности декомпозирует ее на множество формул вида W(x, y) ⇒ ⇒ R(S)(x, y), где W(x, y) — произвольная посылка. Ниже даны правила доказательства формулы для различных видов операторов в позиции оператора S(x: y):

$$\mathbf{FP:}\ \frac{\begin{array}{l}\mathtt{W}\left(\mathtt{x,y,z}\right) \rightarrow \mathtt{R}\left(\mathtt{B}\right)\left(\mathtt{x,y}\right);\\ \mathtt{W}\left(\mathtt{x,y,z}\right) \rightarrow \mathtt{R}\left(\mathtt{C}\right)\left(\mathtt{x,z}\right);\end{array}}{\mathtt{W}\left(\mathtt{x,y,z}\right) \rightarrow \mathtt{R}\left(\mathtt{B}\right)\ ||\ \mathtt{C}\left(\mathtt{x},\left(\mathtt{y,z}\right)\right)},$$

$$\mathtt{W(x,y)} \rightarrow \exists \mathtt{z R(B)(x,z)};$$

$$\textbf{FS:}\ \frac{\mathtt{W(x,y)\ \&R(B)(x,z)} \rightarrow \mathtt{R(C)(z,y)};}{\mathtt{W(x,y)} \rightarrow \mathtt{R(B;C)(x,y)}},$$

$$\mathtt{W(x,y)\ \&\ E} \rightarrow \mathtt{R(B)(x,y)};$$

$$\textbf{FC:}\ \frac{\mathtt{W(x,y)\ \&}\neg\mathtt{E} \rightarrow \mathtt{R(C)(x,y)};}{\mathtt{W(x,y)} \rightarrow \mathtt{R(}\textbf{if}\ \mathtt{(E)\ B\ }\textbf{else}\ \mathtt{C)(x,y)}}.$$

Приведенные правила достаточно просты. Их можно применять многократно для декомпозиции вхождений R(S)(x, y) при доказательстве формул вида W(x, y) → R(S)(x, y). Таких формул большинство среди посылок в приведенных выше правилах. Однако правило **FS** использует посылки двух других видов: W(x, y) → ∃y R(S)(x, y) и W(x, y) & R(S)(x, y) → → H(x, y), где W(x, y) и H(x, y) — произвольные формулы. Ниже приведены правила для декомпозиции вхождений R(S)(x, y) в этих новых видах формул.

$$\mathtt{W(x)} \rightarrow \exists \mathtt{y R(B)(x,y)};$$

$$\textbf{EP:}\ \frac{\mathtt{W(x)} \rightarrow \exists \mathtt{z R(C)(x,z)};}{\mathtt{W(x)} \rightarrow \exists \mathtt{y R(B \parallel C)(x,(y,z))}},$$

$$\mathtt{W(x)} \rightarrow \exists \mathtt{z R(B)(x,z)};$$

$$\textbf{ES:}\ \frac{\mathtt{W(x)\ \&\ R(B)(x,z)} \rightarrow \exists \mathtt{y R(C)(z,y)};}{\mathtt{W(x)} \rightarrow \exists \mathtt{y R(B;C)(x,y)}},$$

$$\mathtt{W(x)\ \&\ E} \rightarrow \exists \mathtt{y R(B)(x,y)};$$

$$\textbf{EC}:\ \frac{\mathtt{W(x)\ \&}\neg\mathtt{E} \rightarrow \exists \mathtt{y R(C)(x,y)};}{\mathtt{W(x)} \rightarrow \exists \mathtt{y R(}\textbf{if}\ \mathtt{(E)\ B\ }\textbf{else}\ \mathtt{C)(x,y)}}.$$

Пусть A(x: y) — нерекурсивный вызов предиката, а P(x) — предусловие этого предиката. Вхождение R(A)(x, y) под квантором существования декомпозируется следующим правилом:

$$\textbf{EB:}\ \frac{\mathtt{Corr(A,P,Q)(x)};\ \forall \mathtt{y\,(W(x)} \rightarrow \mathtt{P(x))}}{\mathtt{W(x)} \rightarrow \exists \mathtt{y R(A)(x,y)}}.$$

Приведенная система правил позволяет исключить вхождение квантора существования. Вхождения семантики оператора в левой части формулы декомпозируются по следующим правилам:

$$\textbf{FLP:}\ \frac{\mathtt{W(x,y,z)\ \&\ R(B)(x,y)\ \&\ R(C)(x,z)} \rightarrow \mathtt{H(x,y,z)};}{\mathtt{W(x,y,z)\ \&\ R(B\ ||\ C)(x,(y,z))} \rightarrow \mathtt{H(x,y,z)}},$$

$$\textbf{FLS:}\ \frac{\mathtt{W(x,y)\ \&\ R(B)(x,z)\ \&\ R(C)(z,y)} \rightarrow \mathtt{H(x,y)};}{\mathtt{W(x,y)\ \&\ R(B;C)(x,y)} \rightarrow \mathtt{H(x,y)}},$$

$$\mathtt{W(x,y)\ \&\ E\ \&\ R(B)(x,y)} \rightarrow \mathtt{H(x,y)};$$

$$\textbf{FLC:}\ \frac{\mathtt{W(x,y)\ \&}\neg\mathtt{E\&R(B(C))(x,y)} \rightarrow \mathtt{H(x,y)};}{\mathtt{W(x,y)\ \&\ R(}\textbf{if}\ \mathtt{(E)\ B\ }\textbf{else}\ \mathtt{C)(x,y)} \rightarrow \mathtt{H(x,y)}},$$

$$\textbf{FLB:}\ \frac{\mathtt{P_A(x)};\mathtt{W(x,y)\ \&\ Q_A(x,y)} \rightarrow \mathtt{H(x,y)};}{\mathtt{W(x,y)\ \&\ R(A)(x,y)} \rightarrow \mathtt{H(x,y)}}.$$

Модель системы правил вывода в PVS. Для доказательства корректности описанных выше правил была разработана модель в системе PVS.

Для операторов языка P_2 в модели определены теории. Внутри теорий определены входные и выходные параметры операторов, определены логики операторов. В модели представлены теории для условного оператора, параллельного оператора, для четырех видов оператора суперпозиции.

В модели определены теория для тотальности оператора и для однозначности спецификации. Определены теории для формул Corr. Каждое правило представляется в виде теоремы в теории на PVS.

Система верификации

Система верификации разрабатывалась как back-end в системе предикатного программирования. Основным компонентом системы верификации является генератор формул корректности программы, представленной во внутреннем представлении (рис. 2).

В трансляторе с языка P реализован статический контроль типов. Транслятор формирует условия совместимости и пытается автоматически их разрешить. В том случае, если условие совместимости типов не может быть разрешено статически при трансляции, это условие генерируется в виде логической формулы для последующего доказательства в CVC3 или PVS. Данный механизм ранее был реализован в инструменте контроля динамической семантики [15]. Условия совместимости типов добавляются к сгенерированным формулам корректности программы.

Значительная часть генерируемых формул являются простыми для доказательства. Истинность таких формул проверяется автоматически с помощью решателя CVC3. Для доказательства более сложных формул корректности используется система интерактивного доказательства PVS.

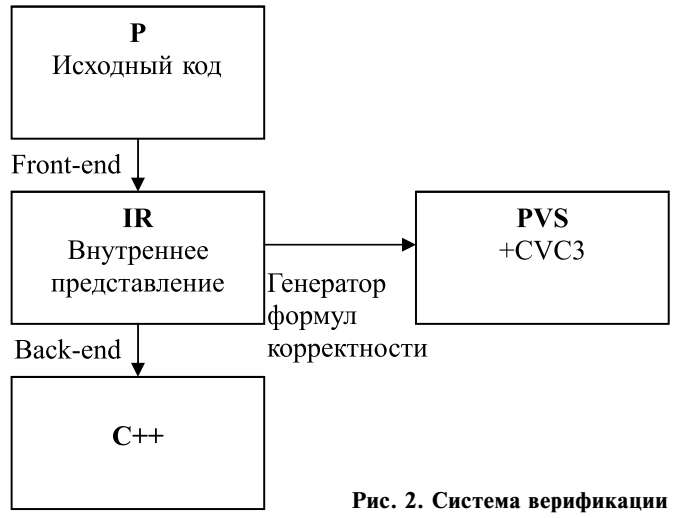


Рис. 2. Система верификации

Генерация формул корректности

Генерация формул корректности проходит по схеме, представленной на рис. 3.

Задачей алгоритма является автоматическое построение формул корректности предикатной программы. На языке P предикатная программа представлена набором определений предикатов. Корректность программы — это корректность всех определенных предикатов.

Из программы последовательно извлекаются определения предикатов вида (1). Для тела предиката, представленного оператором S, реализуется преобразование к канонической форме — это трансляция с языка P на язык P_2, существенно упрощающая структуру исходной программы.

Для преобразованного оператора происходит построение условий корректности.

На основе полученных условий корректности строится теория, которая содержит в себе, помимо самих условий, еще и определение необходимых формул и типов, а также ссылки на импортируемые теории.

Пример

Дадим иллюстрацию работы метода дедуктивной верификации на примере программы умножения двух натуральных чисел a и b, с использованием лишь операций сложения и вычитания.

Для того чтобы получить программу в форме хвостовой рекурсии и автоматически преобразовать ее в программу с циклом for на этапе компиляции, мы должны рассмотреть более общую задачу mult(a, b, d: c) со спецификацией [a ⩾ 0 & b ⩾ 0 & d ⩾ 0, c = a * b + d].

Ниже приведена предикатная программа умножения через сложение.

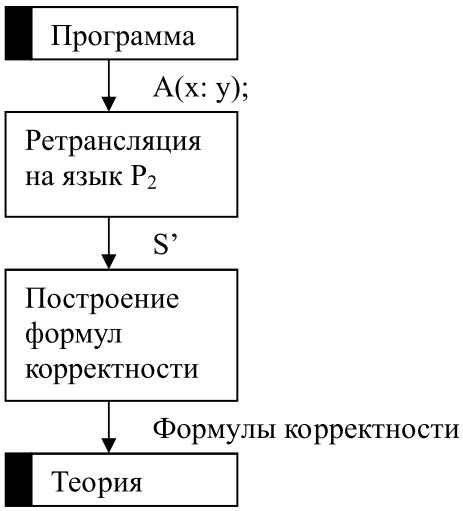


Рис. 3. Общая схема генерации условий корректности

```
mult(nat a, b, d: nat c)
pre a ⩾ 0 & b ⩾ 0 & d ⩾ 0
{
   if (a = 0)
      c = d
   else
      mult(a − 1, b, d + b: c)
}
post c = a * b + d
measure a;
```

Определим предусловие, постусловие и меру в виде отдельных формул:

```
formula P _ mult(nat a, b, d) =
= a ⩾ 0 & b ⩾ 0 & d ⩾ 0;
formula Q _ mult(nat a, b, d, c) = c = a * b + d;
formula m(nat a: nat) = a;
```

Корректность предиката mult, по определению (2), выражается формулой

```
Corr(mult, P _ mult, Q _ mult)(a, b, d).        (3)
```

Таким образом, предикат mult будет корректен, если нам удастся доказать истинность формулы (3). В рамках эксперимента можно раскрыть эту формулу по определению и попытаться доказать. Это будет намного сложнее по сравнению с нашим методом, описанным ниже.

Разложим формулу (3), заменив в ней вхождение предиката mult на его тело, представленное условным оператором. Применение правила **RC** порождает две новые цели:

```
        Corr(c = d, λa, b, d.
P _ mult(a, b, d) & a = 0, Q _ mult)            (4)
 Corr(mult(a − 1, b, d + b: c), λa, b, d.
P _ mult(a, b, d) & ¬ a = 0, Q _ mult)          (5)
```

В формулах (4), (5) в качестве предусловий были использованы λ-абстракции. Формула (4) раскрывается по определению (2) в виде следующей леммы:

```
lemma P _ mult(a, b, d) & a = 0 & c =
    = d ⇒ Q _ mult(a, b, d, c);                 (6)
```

К формуле (5) применяется правило **RB**, где B(a, b, d) = |a − 1, b, d + b| и PB = a − 1 ⩾ 0. Первая посылка правила **RB** опускается ввиду рекурсивности предиката mult. Вторая посылка тождественно истинна. Третья и четвертая посылки записываются в виде следующих лемм:

```
lemma P _ mult(a, b, d) & ¬ a =
=  0 ⇒ a − 1 ⩾ 0 & m(a − 1) < m(a);             (7)
lemma P _ mult(a, b, d) & ¬ a =
= 0 & Q _ mult(a − 1, b, d + b, c)⇒
    ⇒ Q _ mult(a, b, d, c);                     (8)
```

Доказательство корректности предиката mult сводится к доказательству истинности трех описанных выше лемм. Эти леммы проверяются решателем CVC3.

Для оценки эффективности метода проведем сравнение описанного процесса верификации предиката `mult` с классическим процессом верификации по Хоару для императивной программы, полученной трансляцией предиката `mult` на императивное расширение языка P, в ходе которой хвостовая рекурсия заменяется циклом **while**. В качестве инструмента воспользуемся платформой Why3. Императивная программа, соответствующая предикату `mult`, на языке WhyML представляется следующим образом:

```
module Mul

use import int.Int
use import ref.Ref

predicate pre (x y z: int) =
     x > = 0 /\ y > = 0 /\ z > = 0

predicate post (x y z r: int) =
     r = x*y + z

predicate inv (x y z r: int) =
     r = (x - y)*z

let mul (a0 b d: int)
     requires { pre a0 b d }
     ensures { post a0 b d result }
=
     let c = ref 0 in
     let a = ref a0 in
     while!a <> 0 do
          invariant { inv a0 !a b !c }
          a: =!a - 1;
          c: =!c + b;
     done;
     (!c + d)
end
```

Отметим, что введение дополнительной переменной `a0` и оператора `a: = a0` необходимо для построения инварианта `(a0 – a)*b = c`.

Why3 автоматически строит теорию для доказательства корректности функции `Mul`. Для удобства сравнения формулы корректности программы `Mul` приведены здесь к виду, используемому в системе PVS:

```
pre(a0, b, d) = > inv(a0, a0, b, 0)                    (9)

pre(a0, b, d) & inv(a0, a, b, c) & a! = 0
     = > inv(a0, a - 1, b, c + b)                      (10)

pre(a0, b, d) & inv(a0, a, b, c) & a = 0
     = > post(a0, b, d, c + d)                         (11)
```

Наборы формул корректности для программ `mult` и `Mul` принципиально различаются; совпадающих формул нет. Формула (9) аналогична второй посылке правила RB, истинность которой определена на стадии построения формул. Исключим из рассмотрения формулу (7), определяющую условие завершения программы `mult`. Отметим, что формула, генерируемая для доказательства завершения императивной программы `Mul`, сравнима по сложности.

Две группы формул (6, 8) и (10, 11) структурно подобны: вхождениям постусловия `Q _ mult` в первой группе соответствуют вхождения `inv` во второй группе. Вторая группа формул несколько сложнее первой — в ней используются пять переменных против четырех в первой группе. При этом обе группы автоматически доказываются SMT-решателями CVC3 и Alt-Ergo.

Для доказательства корректности программы `Mul` дополнительно требуется построить инвариант `(a0 – a)*b = c`. Это нетривиально, поскольку необходимо догадаться инструментировать программу оператором `a: = a0`. Построение инварианта намного сложнее написания формул предусловия `P _ mult` и постусловия `Q _ mult`.

Доказательство истинности формул корректности

Алгоритм, описанный выше, позволяет автоматически генерировать формулы тотальной корректности предикатной программы. Для каждого определения предиката строится теория, содержащая в себе определения формул корректности и типов. В дальнейшем необходимо доказать истинность этих формул.

SMT-решатель — это инструмент, определяющий разрешимость логических формул в теории. Интерактивный инструмент доказательства теорем — это автоматизированная система, поддерживающая построение пользователем формальных доказательств теорем.

Все формулы проходят проверку на SMT-решателе CVC3 [16]. Формулы транслируются во внутреннее представление решателя. Решатель осуществляет проверку на истинность и в зависимости от результата выставляет формуле статус valid, invalid или unknown.

Для доказательства истинности формул, которые решатель не смог проверить, используется система PVS [8]. Теории, построенные генератором, транслируются на язык спецификаций системы. После чего пользователю необходимо в интерактивном режиме построить доказательство на языке команд системы PVS.

Опыт применения системы верификации

В рамках учебного курса "Формальные методы в описании языков и систем программирования" магистрантам факультета информационных технологий Новосибирского государственного университета было предложено задание по написанию формальной спецификации содержательно сформулированной задачи. Каждый магистрант выбрал одну из 40 предложенных задач. В качестве дополнительного задания для желающих (10 магистрантов) предложено написать предикатную программу и провести доказательство ее корректности в системе автоматического доказательства PVS.

Построение формул корректности для 10 предикатных программ проводили с применением системы верификации, описанной в настоящей работе. Это первый опыт производственной эксплуатации данной системы верификации.

Суммарно по всем задачам было сгенерировано 363 формулы. Из них 260 — это условия тотальной корректности. Остальные 103 — это условия совместимости типов.

Программы небольшие. Число операторов не более 13. Однако количество формул корректности и семантики весьма значительно. В связи с этим от работы решателя CVC3 требуется показать приемлемый результат. Условия совместимости типов желательно проверить полностью автоматически.

Решатель смог проверить 72 % формул семантической корректности и 45 % формул тотальной корректности.

Заключение

В работе описан метод, позволяющий доказывать тотальную корректность предикатных программ. На основании данного метода была разработана система дедуктивной верификации предикатных программ, которая автоматически генерирует формулы корректности для программ с исходным кодом на языке P. Проведены результаты апробация разработанной системы в рамках курса "Формальные методы в описании языков и систем программирования" в 2013—2015 гг.

Разработанный метод дедуктивной верификации предикатных программ имеет существенные преимущества по сравнению с классическим методом Хоара. Верификация предикатной программы требует в 2—3 раза меньше трудозатрат и времени верификации аналогичной императивной программы. Это определяет целесообразность разработки и верификации сложных и критических фрагментов кода программного комплекса в системе предикатного программирования с получением итогового кода таких фрагментов на языке C++.

В дальнейшем планируется разработать метод доказательства корректности для полного языка P, в частности, для предикатов с переменными предикатного типа в качестве параметров. Предполагается снабдить генерируемую теорию на PVS подробными комментариями.

Работа выполнена при поддержке РФФИ, грант № 16-01-00498.

Список литературы

1. **Карнаухов Н. С., Першин Д. Ю., Шелехов В. И.** Язык предикатного программирования P. Препринт № 153. Новосибирск: ИСИ СО РАН, 2010. 42 с. URL: http://persons.iis.nsk.su/files/persons/pages/plang12.pdf

2. **Шелехов В. И.** Методы доказательства корректности программ с хорошей логикой // Межд. конф. "Современные проблемы математики, информатики и биоинформатики", посвященная 100-летию со дня рождения А. А. Ляпунова. 2011. 17 с. URL: http://conf.nsc.ru/files/conferences/Lyap-100/fulltext/74974/75473/Shelekhov_prlogic.pdf

3. URL: http://www.iis.nsk.su/persons/vshel/files/rules.zip

4. **Meyer B.** Towards a Calculus of Object Programs. ETH Zurich, ITMO & Eiffel Software Technical Report. 2011, July.

5. **Pnueli M.** Axiomatic Approach to Total Correctness of Programs // Acta Informatica. 1974. Vol. 3, Iss. 3. P. 243—263.

6. **Dijkstra E. W.** Guarded commands, nondeterminancy and formal derivation of a programms // Communications of the ACM. 1975. Vol. 18, Iss. 8. P. 453—457.

7. **Reynolds J. C.** Separation Logic: A Logic for Shared Mutable Data Structures // LICS '02 Proc. of the 17th Annual IEEE Symposium on Logic in Computer Science. 2002. P. 55—74.

8. **Owre N., Shankar N., Rushby J. M., Stringer-Calvert D. W. J.** PVS Language Reference.

9. **Shelekhov V. I.** Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements // Automatic Control and Computer Sciences. 2011. Vol. 45, N. 7. P. 421—427.

10. **Шелехов В. И.** Верификация и синтез эффективных программ стандартных функций в технологии предикатного программирования // Программная инженерия. 2011. № 2. С. 14—21.

11. **Вшивков В. А., Маркелова Т. В., Шелехов В. И.** Об алгоритмах сортировки в методе частиц в ячейках // Научный вестник НГТУ. 2008. Т. 4 (33). С. 79—94.

12. **Шелехов В. И.** Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования. Препринт ИСИ СО РАН. № 164. Новосибирск, 2012. 30 с.

13. **Шелехов В. И.** Семантика языка предикатного программирования // ЗОНТ-15. Новосибирск, 2015. 13 с. URL: http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf

14. **Предиктное** программирование: Учеб. пособие / Под ред. В. И. Шелехова. Новосибирск: Изд. НГУ, 2009. 111 с.

15. **Каблуков И. В., Шелехов В. И.** Контроль динамической семантики предикатной программы. Препринт ИСИ СО РАН. № 162. Новосибирск, 2012. 28 с.

16. **Barrett C., Tinelli C.** CVC3 // Proc. of the 19th Internat. Conf. on Computer Aided Verification (CAV '07). 2007. Vol. 4590 of Lecture Notes in Computer Science. P. 298—302. Springer, Berlin, Germany.

System for Deductive Verification of Predicate Programs

M. S. Chushkin, chushkinm@rambler.ru, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation

Corresponding author:

Chushkin Michail S., Postgraduate Student, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation,
e-mail: chushkinm@rambler.ru

Received on February 18, 2016
Accepted on March 03, 2016

The programming language P defines a class of programs which do not interact with the external environment. Infinite execution of such programs is meaningless, so the execution of a correct P program must terminate. Loops and pointers are prohibited in the P language.

In present article the method of deductive verification for predicate programs is developed. The formula of total correctness for a predicate program with respect to its specification using the formal operation semantics is presented. The rules of decomposition the total correctness formula into simple and short formulas are developed. The correctness of rules is proved in PVS.

A tool for deductive verification that generates formulas of program correctness is implemented. The correctness formulas are generated by using the system of rules. The correctness formulas are then translated to the SMT-solver CVC3 and the specification language of PVS-system.

The type compatibility errors can lead to an unpredictable behavior of a program. The translator of P language performs full type checking. Some checks couldn't be discharged statically. They are translated to the PVS specification language for use to prove it in the PVS.

All correctness formulas are first tested in the SMT-solver CVC3. Unsolved formulas are then translated to the set of theories in the PVS specification language.

Keywords: *predicate programming, total correctness of a program, deductive verification, proof assist system PVS, SMT-solver CVC3*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project № 16-01-00498.*

For citation:

Chushkin M. S. System for Deductive Verification of Predicate Programs, *Programmnaya Ingeneria*, 2016, vol. 7, no. 5, pp. 202—210.

DOI: 10.17587/prin.7.202-210

References

1. **Karnauhov N. S., Pershin D.Ju., Shelehov V. I.** Jazyk predikatnogo programmirovanija P, *Preprint N 153*, Novosibirsk, Publishing house of ISI SO RAN, 2010. 42 p. (in Russian).

2. **Shelehov V. I.** Metody dokazatel'stva korrektnosti programm s horoshej logikoj, *Mezhd. konf. "Sovremennye problemy matematiki, informatiki i bioinformatiki", posvjashhennaja 100-letiju so dnja rozhdenija A. A. Ljapunova*, 2011, 17 p. (in Russian).

3. **Available** at: http://www.iis.nsk.su/persons/vshel/files/rules.zip

4. **Meyer B.** Towards a Calculus of Object Programs, ETH Zurich, ITMO & Eiffel Software Technical Report, July 2011.

5. **Pnueli M.** Axiomatic Approach to Total Correctness of Programs, *Acta Informatica*, September 1974, vol. 3, iss. 3, pp. 243—263.

6. **Dijkstra E. W.** Guarded commands, nondeterminancy and formal derivation of a programs, *Communications of the ACM*, Aug. 1975, vol. 18, iss. 8, pp. 453—457.

7. **Reynolds J. C.** Separation Logic: A Logic for Shared Mutable Data Structures, *LICS'02 Proc. of the 17th Annual IEEE Symposium on Logic in Computer Science*, 2002, pp. 55—74.

8. **Owrs N., Shankar N., Rushby J. M., Stringer-Calvert D. W. J.** PVS Language Reference.

9. **Shelekhov V. I.** Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements, *Automatic Control and Computer Sciences*, 2011, vol. 45, no. 7, pp. 421—427.

10. **Shelehov V. I.** Verifikacija i sintez jeffektivnyh programm standartnyh funkcij v tehnologii predikatnogo programmirovanija, *Programmnaya ingeneria*, 2011, no. 2, pp. 14—21 (in Russian).

11. **Vshivkov V. A., Markelova T. V., Shelehov V. I.** Ob algoritmah sortirovki v metode chastic v jachejkah, *Nauchnyj vestnik NGTU*, 2008, vol. 4 (33), pp. 79—94. (in Russian).

12. **Shelehov V. I.** Razrabotka i verifikacija algoritmov piramidal'noj sortirovki v tehnologii predikatnogo programmirovanija, *Prepr. ISI SO RAN, no. 164*, Novosibirsk, 2012, 30 p. (in Russian).

13. **Shelehov V. I.** Semantika jazyka predikatnogo programmirovanija, *ZONT-15,* Novosibirsk, 2015, 13 p., available at: http://persons.iis.nsk.su/files/persons/pages/semZontl.pdf (in Russian)

14. **Shelehov V. I.** *Prediktnoe programmirovanie.* Novosibirsk, Publishing house of NGU. 2009, 111 p. (in Russian).

15. **Kablukov I. V., Shelehov V. I.** Kontrol' dinmaicheskoj semantiki predikatnoj programmy, *Prepr. ISI SO RAN N 162*, Novosibirsk, 2012, 28 p. (in Russian).

16. **Barrett C., Tinelli C.** CVC3, *Proc. of the 19th Internat. Conf. on Computer Aided Verification (CAV'07)*, 2007, vol. 4590 of Lecture Notes in Computer Science, pp. 298—302, Springer, Berlin, Germany.

УДК 004.5 DOI: 10.17587/prin.7.211-214

Я. А. Туровский, канд. мед. наук, доц., зав. лаб., e-mail: yaroslav_turovsk@mail.ru,
С. Д. Кургалин, д-р физ.-мат. наук., проф., зав. каф.,
А. В. Алексеев, магистрант, техник, **А. А. Вахтин,** канд. физ.-мат. наук, доц., вед. программист,
Воронежский государственный университет

Организация дополнительного канала обратной связи интерфейса мозг–компьютер

Рассмотрены преимущества использования дополнительного канала обратной связи человека и внешнего устройства, управляемого с помощью интерфейса мозг—компьютер. Продемонстрировано, что имеющиеся программно-аппаратные средства существенно ограничивают возможности управления внешними устройствами и программами в рамках интерфейса мозг—компьютер. Предложено создание новых программно-аппаратных средств для интерфейса мозг—компьютер, реализуемых как для синхронных, так и для асинхронных интерфейсов. Такие средства сформируют дополнительный канал обратной связи, информирующий человека о ходе выработки интерфейсом решения о выборе той или иной команды для внешних устройств и о проводимых при этом вычислениях. Подобный подход позволяет человеку активнее, чем при применении классической схемы интерфейса мозг—компьютер, изменять функционирование тех или иных отделов мозга и обеспечивать процесс управления.

Ключевые слова: *человеко-машинный интерфейс, интерфейс мозг—компьютер, обратная связь, программно-аппаратный комплекс*

Введение

Традиционно под нейрокомпьютерным интерфейсом (НКИ, интерфейс мозг—компьютер) подразумевают коммуникационную систему, в которой сообщения или команды, посылаемые индивидуумом во внешний мир, проходят не через обычные выходные каналы мозга в виде периферийных нервов и мышц, а осуществляются (передаются) иным образом [1].

В общем случае взаимодействие человека и компьютера в рамках парадигмы НКИ выглядит следующим образом [2—4].

1. Пользователь, желая достичь определенного результата, изменяет активность своего головного мозга, что проявляется в динамике волновых паттернов электроэнцефалограммы (ЭЭГ), данных магнитоэнцефалограммы, степени оксигенации крови, метаболизма тех или иных отделов мозга.

2. Специальный прибор (электроэнцефалограф, магнитоэнцефалограф, трансчерепной оксиметр, магнитно-резонансный томограф) регистрирует полученные паттерны и пересылает их для дальнейшей обработки в компьютер.

3. Компьютер, используя ряд алгоритмов обработки и классификации данных, определяет, какое именно действие необходимо совершить, и реализует его через устройство-эффектор или компьютерную программу [5].

Для детектирования изменения активности мозга требуется определенное время как на накопление необходимого числа отсчетов сигнала, содержащего требуемую активность, так и на обработку этих отсчетов. При этом в рамках классической схемы управления с использованием НКИ, в период от начала осознанной генерации пользователем команды до, собственно, вывода этой команды на управляемое устройство человек не имеет представления о том, какие команды рассматриваются программно-аппаратным комплексом. Не имеет пользователь и представления о том, какие из них выбраны для отправки на устройство-эффектор, как соотносятся вероятности выбора этих команд и т. д. Как результат, человек увидит, какую именно команду реализовал интерфейс только тогда, когда устройство или программа-эффектор уже выполнили эту команду. Учитывая, что скорость работы разных НКИ варьируется в широких пределах (от 1 [6] до 5...30 с [7] на одну команду), то очевидно, что у пользователя интерфейса достаточно долгое время отсутствует информация о его работе, что не позволяет пользователю контролировать и/или изменять свои так называемые ментальные состояния. Это обстоятельство, в свою очередь, не дает возможности контролировать и/или изменять активность тех или иных отделов головного мозга и, следовательно, характер считываемых с его мозга временных рядов

данных. Подобная ситуация, очевидно, негативно влияет на скорость и точность работы НКИ.

Цель работы, результаты которой представлены в статье, предложить и реализовать способ повышения эффективности функционирования НКИ путем формирования в нем дополнительного канала обратной связи человек—компьютер для непосредственного контроля за работой этого интерфейса.

Принципы создания программно-аппаратных решений при организации дополнительного канала обратной связи для интерфейса мозг—компьютер

Схема потоков информации при работе НКИ с учетом дополнительного канала коммуникации, обеспечивающего человека-оператора НКИ в режиме реального времени (или с минимальной временной задержкой) детальной информацией о ходе обработки паттернов мозговой активности, представлена на рис. 1.

Как видно на рис. 1, программы — обработчики данных, поступающих от мозга, выделяющие паттерны мозговой активности, интерпретируемые в качестве команд при стандартном проектировании НКИ, имеют выход только на управляемые устройства или программы-эффекторы. Иными словами, выход информации из программ и модулей обработки реализован только на внешние устройства или в программы, непосредственно выполняющие команды пользователя (на рис. 1 это показано сплошными стрелками).

В свете отмеченного выше необходим дополнительный канал выхода информации непосредственно на пользователя интерфейса, при реализации которого программы-обработчики данных с мозга в той или иной форме смогут сообщать ему о ходе проводимого анализа данных (см. рис. 1).

Подобное решение достигается тем, что при управлении устройствами, включая и управление компьютером, на основе НКИ происходит регистрация активности головного мозга следующими методами как по отдельности, так и совместно

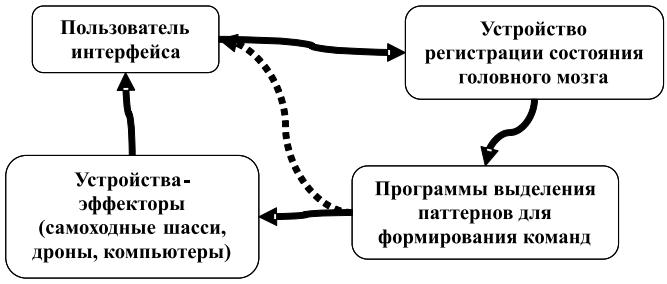


Рис. 1. Схема потоков информации при стандартном виде работы НКИ (сплошные стрелки) и при появлении дополнительного канала обратной связи (пунктирная стрелка)

в любой их комбинации: электроэнцефалографическим, магнитноэнцефалографическим, магнитно-резонансным томографическим (включая функциональный магнитно-резонансный томографический), транскраниальным оксиметрическим как с инвазивными, так и с неинвазивными датчиками. Полученные данные обрабатываются компьютером в целях выделения паттернов, интерпретируемых как команды из зарегистрированных временных последовательностей данных. Затем осуществляется передача сформированных команд на внешние по отношению к устройству регистрации и обработки сигналов с мозга устройства (самоходные шасси, летающие платформы, компьютеры и т. д.). При этом в ходе обработки зарегистрированных с мозга временных последовательностей оператору предоставляется информационное сообщение в период времени между началом накопления и обработки временной последовательности для выбора управляющей команды и завершением этой обработки с последующей передачей команды на внешнее устройство. Сообщение предоставляется в доступной и/или удобной для оператора форме. Очевидно, что предварительные результаты выбора команды сообщаются из числа всех возможных при текущем состоянии интерфейса (рис. 2). Данное сообщение доставляется с использованием как по отдельности, так и в любых комбинациях зрительного, звукового, тактильного или иного другого возможного канала коммуникации.

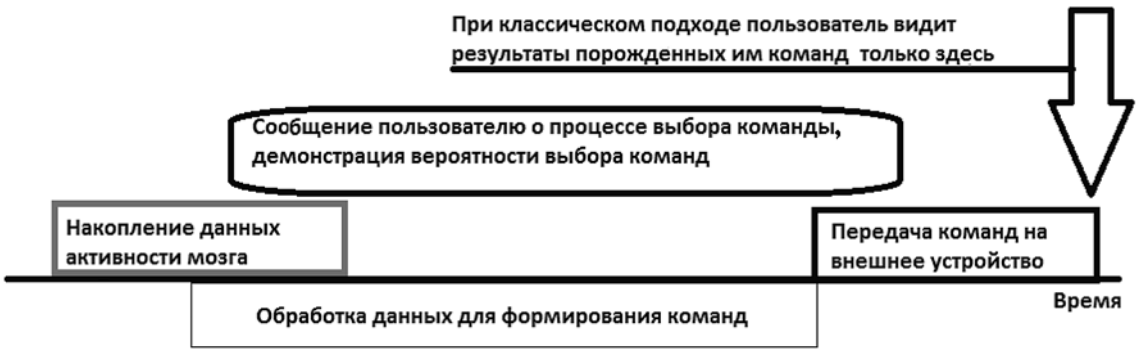


Рис. 2. Схема работы НКИ с использованием дополнительного канала обратной связи

Программно-аппаратные решения при реализации дополнительного канала обратной связи в интерфейсе мозг—компьютер

При реализации программно-аппаратных решений для использования дополнительного канала обратной связи НКИ следует учитывать существование как синхронных, так и асинхронных интерфейсов. В первом виде интерфейсов реализованы технологии, основанные на применении вызванных потенциалов головного мозга. В асинхронных интерфейсах анализируется фоновая активность мозга и из нее выделяются паттерны-команды для управления программами и устройствами. Очевидно, что для обоих случаев программно-аппаратные решения существенно различаются.

В случае синхронных интерфейсов наибольшее удобство для пользователя продемонстрировал разработанный и реализованный в рамках выполнения описанной в данной статье работы программно-аппаратный комплекс. Этот комплекс обеспечивает изменение цвета индикации разнообразных команд, передаваемых пользователем на устройство-эффектор. В случае, если алгоритм управления через НКИ допускает принятие промежуточного решения, например, результата классификации паттерна активности мозга, не выводящегося напрямую на управляемое устройство, а требующего дополнительной команды-подтверждения от пользователя, возможно изменение структуры индикации работы НКИ. Изменение цвета или иные изменения, проявляющиеся в интерфейсе программы, покажут результат выбора как промежуточной команды, так и дополнительной команды-подтверждения. Для синхронных НКИ, создаваемых с использованием LED, в которых фотостимуляция осуществляется без изменения цвета фигур или объектов на экране монитора (для НКИ на основе потенциалов SSVEP или VEP), реализовано программное обеспечение, обеспечивающее индикацию состояния принятия решения НКИ в виде подсветки определенных участков интерфейса программы. При этом оптимальной является такая конфигурация системы используемой стимуляции, когда стимулирующие фотодиоды находятся на минимальном расстоянии от областей формы программы, где осуществляется цветовая индикация. Для асинхронных интерфейсов (использующих уровни α-ритма и β-ритма, фоновые паттерны ЭЭГ и т. п.) в качестве визуального ряда данных при выборе команды применялся набор столбцов гистограммы. Такой набор демонстрирует процесс выбора команды программно-аппаратной частью интерфейса. Более высокие столбцы гистограммы при этом соответствуют большей вероятности выбора команды.

Использование тактильных стимуляторов, исходя из предварительного тестирования пяти пользователей-мужчин в возрасте 20 лет (медианное значение), показало достаточно низкую оценку испытуемыми эффективности такого вида обратной связи. В проведенном эксперименте через одноплатный компьютер Raspberry Pi 2 Model B [8] и модуль Ke-USB24A [9] осуществлялось управление электромагнитами, обеспечивающими вибрацию элементов стимулятора с определенной частотой. Наиболее вероятная по выбору программно-аппаратной части НКИ команда отображалась на предплечье испытуемого в виде активности одного из нескольких закрепленных электромагнитных реле. Все испытуемые при этом указали на затруднения в сосредоточении одновременно на управлении, например, самоходным шасси, и на тактильных ощущениях, поступающих с конечностей. Все они отдали предпочтение использованию видеоинформации, отражающейся непосредственно на экране монитора, куда транслировалось изображение с видеокамеры, установленной на самоходном шасси.

При использовании звукового индикатора для организации обратной связи изменялась частота звука для разных команд. В данном случае испытуемые отметили, что при числе команд, большем четырех, не всегда удавалось корректно распознать стимул.

Применение голосовых команд также не дало устойчивого результата для формирования канала обратной связи: сосредоточившись на процессе управления, ни один из испытуемых не смог точно учесть все сообщения, которые ему передавались по этому каналу коммуникации.

Таким образом, согласно предварительным результатам проведенных исследований было установлено, что для формирования дополнительного канала обратной связи в рамках использования НКИ предпочтение следует отдать применению визуального способа индикации команд НКИ. При этом, однако, нельзя исключить, что эффективность разных каналов коммуникации носит сугубо индивидуальный характер и может изменяться.

Заключение

Рассмотрены преимущества использования дополнительного канала обратной связи человека и управляемого им по этому каналу интерфейса мозг—компьютер внешнего устройства. Продемонстрировано, что имеющиеся в настоящее время программно-аппаратные средства существенно ограничивают возможности управления внешними устройствами в рамках применения интерфейса мозг—компьютер. Предложен подход, заключающийся в создании новых программно-аппаратных средств для интерфейса мозг—компьютер, реализованного для разных типов НКИ. При данном подходе формируется дополнительный канал обратной связи, информирующий пользователя о ходе вычислений и принятия решений по выбору интерфейсом той или иной команды для внешних устройств. Подобный подход позволяет пользователю более активно, чем в классической схеме интерфейса мозг—компьютер, изменять функционирование тех или иных отделов мозга.

Список литературы

1. **Wolpaw J. R., McFarland D. J.** Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans // Proceedings of the National Academy of Science. 2004. Vol. 101 (51). P. 17849—17854.
2. **Nicolas-Alonso L. F., Gomez-Gil J.** Brain Computer Interfaces a Review // Sensors. 2012. Vol. 12. P. 1211—1279.
3. **Brunner P., Bianchi L., Guger C., Cincotti F., Schalk G.** Current trends in hardware and software for brain—computer interfaces (BCIs) // Journal of Neural Engineering. 2011. Vol. 8, N. 2. P. 025001. DOI:10.1088/1741-2560/8/2/025001
4. **Wolpaw J. R., Birbaumer N., McFarland D., Pfurtscheller G., Theresa M.** Brain—computer interfaces for communication and control // Clinical Neurophysiology. 2002. Vol. 113, Issue 6. P. 767—791.
5. **Lotte F., Congedo M., L'ecuyer A., Lamarche F., Arnaldi B.** A review of classification algorithms for EEG-based brain—computer interfaces// Journal of Neural Engineering. 2007. Vol. 4, N. 2. R1—R13. DOI:10.1088/1741-2560/4/2/R01
6. **Gao X., Xu D., Cheng M., Gao S.** A BCI-Based Environmental Controller for the Motion- Disabled // IEEE Transactions on Neural Systems and Rehabilitation Engineering. 2003. Vol. 11, N 2. P. 137—140.
7. **Борзунов С. В., Кургалин С. Д., Максимов А. В., Туровский Я. А.** Оценка скорости работы нейрокомпьютерного интерфейса, основанного на технологии SSVEP // Известия РАН. Теория и системы управления. 2014. № 1. С. 121—129.
8. **Raspberry** Pi 2 Model B. URL: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
9. **USB** модуль Ke-USB24A. URL: http://kernelchip.ru/Ke-USB24A.php

The Additional Feedback Channel Creation for Brain–Computer Interfaces

Ya. A. Turovsky, e-mail: yaroslav_turovsk@mail.ru, **S. D. Kurgalin**, e-mail: kurgalin@bk.ru, **A. V. Alekseev**, e-mail: a_v_alekseev@bk.ru, **A. A. Vahtin,** e-mail: alvahtin@gmail.com, Voronezh State University, 395006, Voronezh, Russian Federation,

Corresponding author:

Turovsky Yaroslav A., Associate Professor, Voronezh State University, 395006, Voronezh, Russia e-mail: yaroslav_turovsk@mail.ru

Received on January 20, 2016
Accepted on January 26, 2016

The purpose of this work is to suggest an implement a means to increase the effectiveness of neurocomputer interface functioning by forming an additional brain—computer feedback channel for the immediate control of the work of this interface. In the classical control scheme, which uses the brain–computer interface, between the beginning of conscious generation of commands by user and the receiving of those commands from the neurocomputer interface by the managed device, the user does not now whichever command is considered by the combined system as the most probable for evaluating. Neither is it known whichever of them are selected for transmission to the effector device, and what the relationships between their probabilities are, etc. As a result, the commands selected for evaluation will be visible only when the interface or the effector program have already evaluated them. In this article are considered the benefits of using an additional feedback channel for the brain—computer interfaces. Thus it is demonstrated that the existing combined software-hardware systems impose significant restrictions on the control of the brain–computer interface by the user. A new combined solution is demonstrated, that is suitable for both synchronous and asynchronous brain–computer interfaces. Such programs form a new feedback channel, that informs a human about the command creations in the software part of the brain–computer interface. This approach allows one to change the functioning of various brain structures more efficiently, than with a classical brain–computer interface scheme.

Keywords: *human—computer interfaces, brain—computer interfaces, feedback, soft-hard complex*

For citation:

Turovsky Ya. A., Kurgalin S. D., Alekseev A. V., Vahtin A. A. The Additional Feedback Channel Creation for Brain—Computer Interfaces, *Programmnaya Ingeneria*, 2016, vol. 7, no. 5, pp. 211—214.

DOI: 10.17587/prin.7.211-214

References

1. **Wolpaw J. R., McFarland D. J.** Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans, *Proceedings of the National Academy of Science*, 2004, vol. 101 (51), pp. 17849—17854.
2. **Nicolas-Alonso L. F., Gomez-Gil J.** Brain Computer Interfaces a Review, *Sensors*, 2012, vol. 12, pp. 1211—1279.
3. **Brunner P., Bianchi L., Guger C., Cincotti F., Schalk G.** Current trends in hardware and software for brain—computer interfaces (BCIs), *Journal of Neural Engineering*, 2011, vol. 8, no. 2, pp. 025001, DOI:10.1088/1741-2560/8/2/025001.
4. **Wolpaw J. R., Birbaumer N., McFarland D., Pfurtscheller G., Theresa M.** Brain—computer interfaces for communication and control, *Clinical Neurophysiology*, 2002, vol. 113, issue 6, pp. 767—791.
5. **Lotte F., Congedo M., L'ecuyer A., Lamarche F., Arnaldi B.** A review of classification algorithms for EEG-based brain—computer interfaces, *Journal of Neural Engineering,* 2007, vol. 4, R1—R13, DOI:10.1088/1741-2560/4/2/R01.
6. **Gao X., Xu D., Cheng M., Gao S.** A BCI-Based Environmental Controller for the Motion- Disabled, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 2003, vol. 11, no. 2, pp. 137—140.
7. **Borzunov C. V., Kurgalin S. D., Maksimov A. V., Turosky Ya. A.** Ocenka skorosti raboty nejrokomp'juternogo interfejsa, osnovannogo na tehnologii SSVEP (Estimation of speed neurocomputer interface, based on the technology of SSVEP), *Izvestija RAN. Teorija i sistemy upravlenija*, 2014, no. 1, pp. 121—129 (in Russian).
8. **Raspberry** Pi 2 Model B, available at: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
9. **USB** modul' Ke-USB24A (USB Module Ke-USB24A), available at: http://kernelchip.ru/Ke-USB24A.php (in Russian).

УДК 629.73.02; 629.73.05/.06; 535.643 DOI: 10.17587/prin.7.215-220

И. О. Жаринов, д-р техн. наук, доц., зав. каф., e-mail: igor_rabota@pisem.net, Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (Университет ИТМО), АО "ОКБ "Электроавтоматика",
О. О. Жаринов, канд. техн. наук, доц., e-mail: zharinov73@hotbox.ru, Санкт-Петербургский государственный университет аэрокосмического приборостроения (ГУАП),

Автоматизация формирования цветовой палитры бортового средства индикации на основе обработки энергетических характеристик цветов с максимальными перцепционными отличиями

Рассмотрена практическая задача поиска цветов, индицируемых на экране бортовых средств индикации и обладающих наилучшими характеристиками восприятия для наблюдателя. Предложен алгоритм автоматизации процедуры поиска, реализуемый в составе автоматизированного рабочего места. Основу алгоритма составляет тройной цикл полного перебора кодов основных цветов (красный, зеленый, синий), задаваемых программно на инструментальной ЭВМ. Критерием качества поиска является цвет, обладающий максимальным цветовым контрастом. Показано, что поиск цветов с наилучшими характеристиками восприятия необходимо осуществлять не во всей области цветов треугольника цветового охвата, а в областях цветов с максимальными перцепционными отличиями. Уменьшение площади областей поиска позволяет снизить на порядок время автоматизированного поиска. Границы областей цветов с максимальными перцепционными отличиями аппроксимированы фигурой четырехугольника. Вершины четырехугольников заданы числовыми значениями координат цветности на XY- и на UV-плоскости. Предложен принцип расчета координат цветности вершин многоугольника, содержащего отображаемые на средстве индикации цвета с максимальными перцепционными отличиями. Расчет координат цветности вершин многоугольника может быть выполнен непосредственно в алгоритме поиска с использованием результатов решения графоаналитическим методом плоскостной задачи вычисления координат точек пересечения двух прямых в декартовой системе координат. Представлены графические изображения областей, отображаемых на средстве индикации цветов с максимальными перцепционными отличиями на XY- и UV-плоскостях.

Ключевые слова: *цветовой охват, контраст изображения, максимум, цвет, перцепционные отличия, авионика*

Введение

В процессе проектирования бортового индикационного оборудования [1] разработчики программного обеспечения сталкиваются с проблемным вопросом выбора компонентов программного кода, задаваемого в цветовой модели RGB (R — Red, G — Green, B — Blue) для каждого отображаемого на экране цвета из состава цветовой палитры [2].

Выбор цвета обусловлен необходимостью соблюдения требований нормативно-технической документации, ограничивающей номенклатуру разрешенных к использованию в индикационных кадрах авионики цветов, и желанием разработчиков получить цветовую палитру, обладающую наилучшими характеристиками восприятия для наблюдателя [3].

Основной характеристикой воспринимаемого наблюдателем изображения является яркостной контраст

$$C_i = (L_i - L_0)/L_0,$$

рассчитываемый на основе результатов измерений яркости L_i каждого отображаемого цвета и яркости L_0

фона изображения. Фон изображения в авионике — черный цвет, реже — серый (для ночных полетов). Цветовая палитра с наилучшими характеристиками восприятия включает компоненты кода *RGB* для цветов, обладающих максимальным контрастом в пределах каждого разрешенного к использованию в бортовых средствах индикации наименования цвета.

Очевидно, что ручной перебор кодов *RGB* с последующим измерением светотехнических характеристик L_i, L_0 и расчетом C_i для каждой комбинации *RGB* — это достаточно трудоемкий и длительный процесс, подлежащий автоматизации. Цель исследования, результаты которого приведены в статье, заключается в поиске программно реализуемого алгоритма, позволяющего автоматизировать процесс определения цветовой палитры с наилучшими характеристиками восприятия для наблюдателя в пределах воспроизводимых средством индикации цветов.

Алгоритм автоматизированного поиска максимума цветового контраста

Алгоритм автоматизированного поиска точки глобального максимума C_i для каждого цвета реализуется на инструментальной ЭВМ (ИЭВМ) и приведен на рис. 1. Основу алгоритма составляет математический аппарат метода пошагового поиска [4], являющегося разновидностью методов автоматизированного поиска оптимальных проектных решений.

Метод обеспечивает пошаговый анализ значений дискретной функции поверхности, в которых определен контраст изображения для каждой точки цвета, заданного кодом *RGB*, с вычислением целевой функции вида $\max(C_i)$.

Начальными условиями алгоритма поиска являются:

- матрица профиля жидкокристаллической панели средства бортовой индикации с коэффициентами X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b, значения которых определяются стандартом Международной комиссии по освещению (МКО) и выбранным в нем оператором ИЭВМ типом системы задания баланса белого цвета (точки белого цвета жидкокристаллической панели) — D-75, D-65, D-55, D-50;
- функции $Q_i(x, y)$, $i = 1, 2, ..., N$, задающие в цветовом пространстве граничные условия для i-го цвета, в пределах которого наименование цвета остается "постоянным" (N — число цветов цветовой палитры).

Основу алгоритма составляет тройной цикл полного перебора кодов *RGB*, изменяющих свое значение в пределах [0...255] последовательно для всех трех компонентов цветов (красного, зеленого и синего) с вычислением в каждой точке (x, y)-координат цветности, удовлетворяющих условию $x, y \in Q_i(x, y)$, на основе значений координат цвета X, Y, Z [5, 6]:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix}^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \qquad (1)$$

$$x = \frac{X}{X + Y + Z}, \quad y = \frac{Y}{X + Y + Z}. \qquad (2)$$

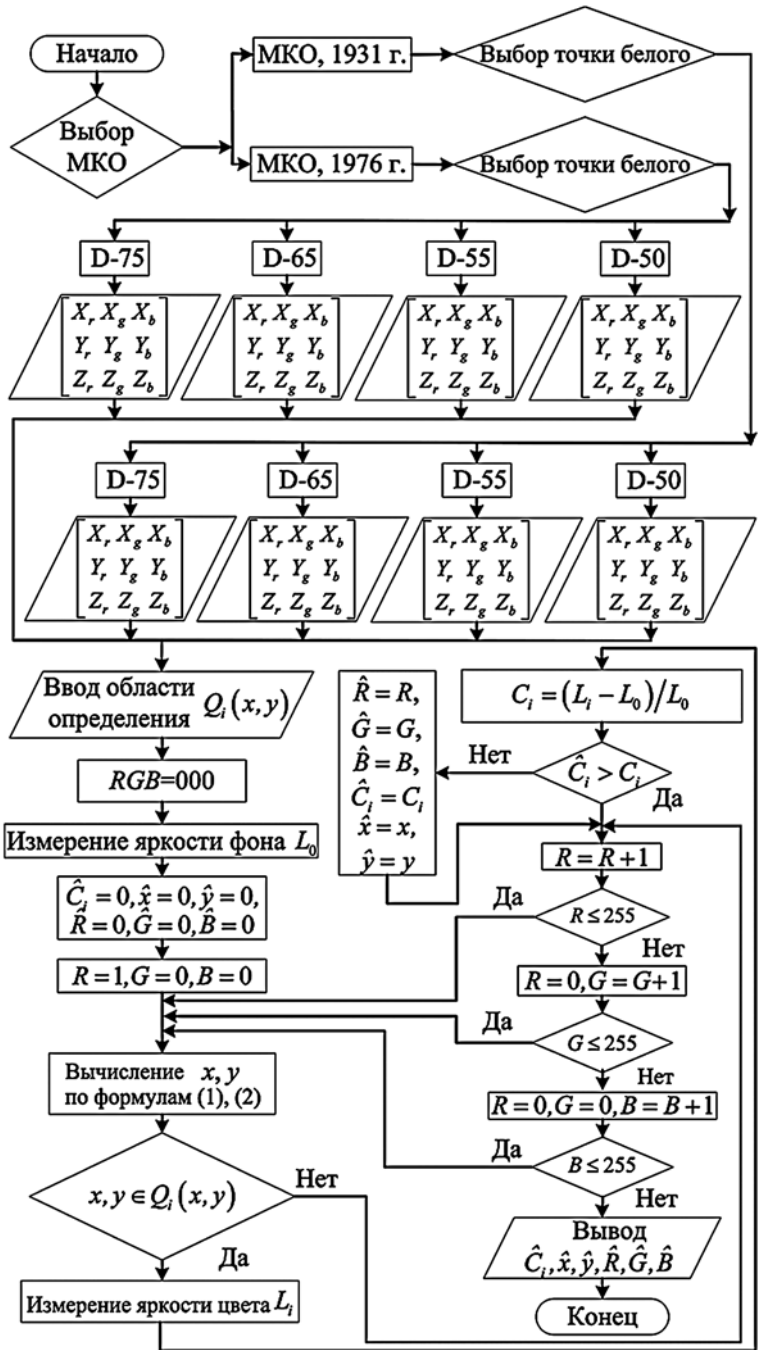


Рис. 1. Алгоритм поиска максимума контраста i-го цвета в области $Q_i(x, y)$

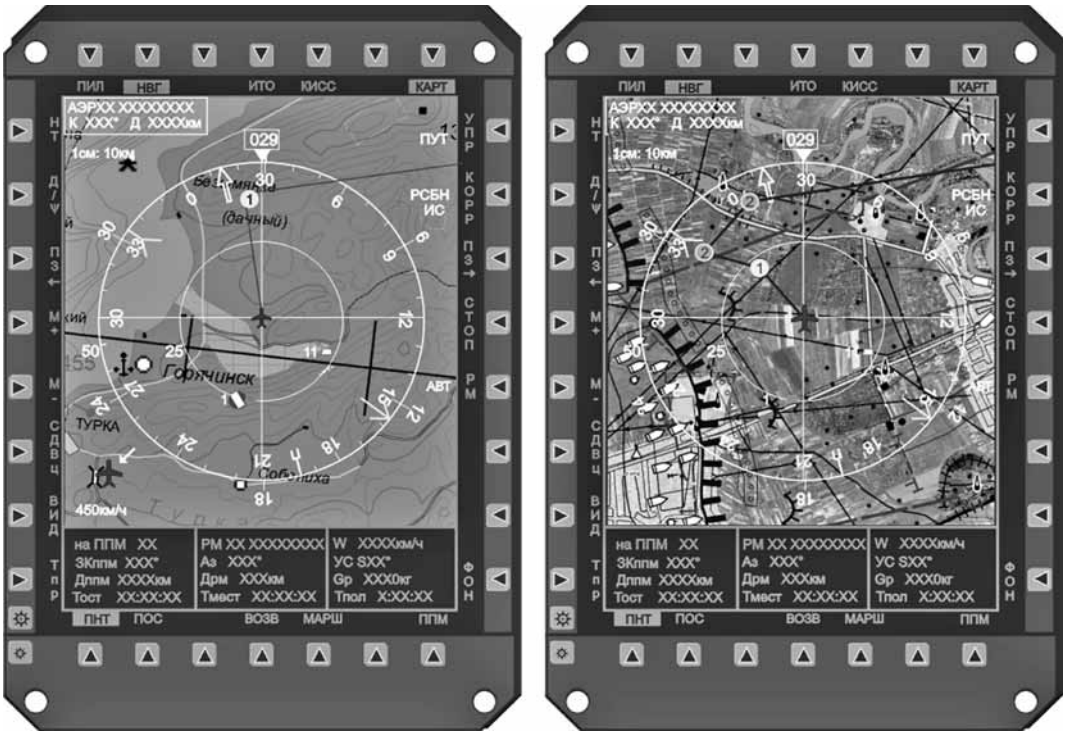


Рис. 2. Примеры индикационных кадров авионики

Шаг инкрементирования каждого компонента кода RGB минимально возможный, дискретный, равен единице.

Одновременно в цикле модифицируются переменные, определяющие точку экстремума $\left\{\widehat{C}_i,\left(\hat{x}_i,\hat{y}_i\right)\right\}$ с максимальным значением контраста $\widehat{C}_i$ в данном цвете. Фиксируется также значение кода $\widehat{R}\widehat{G}\widehat{B}$, при котором достигается точка $\widehat{C}_i$. Яркость фона L_0 в алгоритме определяется однократно на начальном этапе путем измерения на жидкокристаллической панели средства индикации яркости изображения, индицируемого с программно заданным кодом $R = G = B = 0$.

Принцип расчета областей цветов с максимальными перцепционными отличиями

Граничные условия $Q_i(x, y)$ определяются видом и параметрами ограничивающей функции для каждого отображаемого на экране средства индикации цвета (рис. 2) и могут быть рассчитаны непосредственно в цикле анализа значений точек поверхности. Перспективными для практического использования могут быть ограничивающие функции типа многоугольник.

На рис. 3 показаны области цветов с максимальными перцепционными [7] отличиями в цветовых пространствах XY и UV. В тело цветовых локусов вписаны два треугольника цветового охвата, образованных вершинами $R_1G_1B_1$, $R_2G_2B_2$. Треугольник цветового охвата представляет собой область цветов, воспроизводимых на конкретном средстве

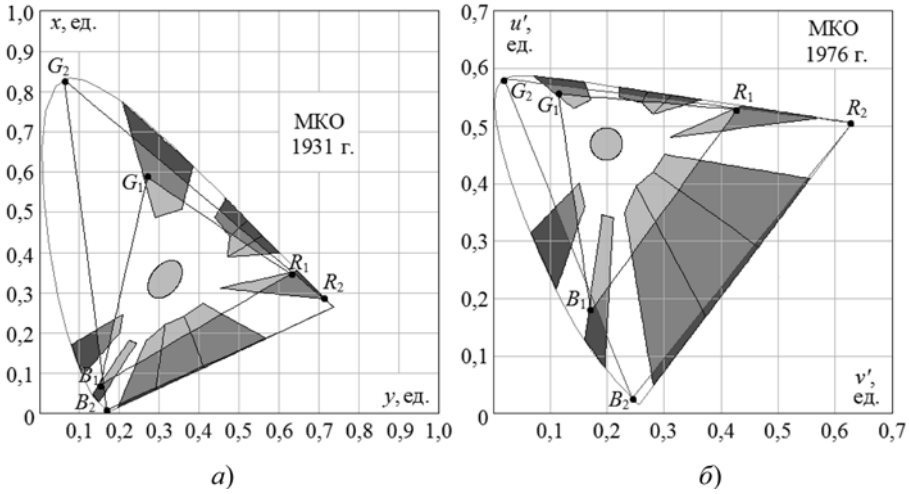


Рис. 3. Граничные области цветов с максимальными перцепционными отличиями в цветовых пространствах:
а — XY; б — UV

индикации, свойства цветовоспроизведения которого определяются коэффициентами профиля жидкокристаллической панели: X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b.

Треугольник $R_1G_1B_1$ показан для следующего профиля жидкокристаллической панели: X_r = = 0,478; X_g = 0,299; X_b = 0,175; Y_r = 0,263; Y_g = 0,650; Y_b = 0,081; Z_r = 0,020; Z_g = 0,160; Z_b = 0,908. Треугольник $R_2G_2B_2$ — это вписанный треугольник цветового охвата максимально возможной площади [8], вершины которого лежат на линии спектральных цветностей, интерполированной сплайнами Безье.

Оттенками серого цвета на рис. 3 закрашены области (в форме многоугольника) для цветов, обладающих максимальными перцепционными отличиями и воспроизводимых на средствах индикации с треугольниками цветового охвата $R_1G_1B_1$, $R_2G_2B_2$.

Можно показать, что координаты точек вершин закрашенных многоугольников вычисляются в результате решения графоаналитическим методом задачи поиска координат точек пересечения прямых линий на плоскости.

Решение задачи поиска координат (x_p, y_p) точки пересечения двух прямых, проходящих в общем случае через точки (x_1, y_1), (x_2, y_2) и (x_3, y_3), (x_4, y_4), на декартовой плоскости имеет следующий вид:

$$\begin{cases} x_p = -\det\begin{pmatrix} \det\begin{pmatrix} x_2 & y_2 \\ x_1 & y_1 \end{pmatrix} & x_1 - x_2 \\ \det\begin{pmatrix} x_4 & y_4 \\ x_3 & y_3 \end{pmatrix} & x_3 - x_4 \end{pmatrix} \Big/ \det\begin{pmatrix} y_2 - y_1 & x_1 - x_2 \\ y_4 - y_3 & x_3 - x_4 \end{pmatrix} \\ y_p = -\det\begin{pmatrix} y_2 - y_1 & \det\begin{pmatrix} x_2 & y_2 \\ x_1 & y_1 \end{pmatrix} \\ y_4 - y_3 & \det\begin{pmatrix} x_4 & y_4 \\ x_3 & y_3 \end{pmatrix} \end{pmatrix} \Big/ \det\begin{pmatrix} y_2 - y_1 & x_1 - x_2 \\ y_4 - y_3 & x_3 - x_4 \end{pmatrix}. \end{cases} \quad (3)$$

Координаты цветности вершин четырехугольников для цветов с максимальными перцепционными отличиями в цветовых пространствах *XY* и *UV*

Цвет	Вершины четырехугольников							
	1		2		3		4	
	u′, ед.	*v′*, ед.	*u′*, ед.	*v′*, ед.	*u′*, ед.	*v′*, ед.	*u′*, ед.	*v′*, ед.
Голубой	0,16	0,36	0,11	0,215	0,065	0,314	0,15	0,4
Зеленый	0,14	0,53	0,07	0,5864	0,16	0,5755	0,17	0,54
Желтый	0,22	0,55	0,22	0,567	0,27	0,5594	0,26	0,54
Янтарный	0,26	0,54	0,27	0,5594	0,311	0,5529	0,28	0,52
Оранжевый	0,28	0,52	0,311	0,5529	0,365	0,545	—	—
Красный	0,31	0,48	0,435	0,5346	0,575	0,5134	—	—
Розовый	0,30	0,45	0,555	0,408	0,468	0,29	0,28	0,42
Малиновый	0,28	0,42	0,465	0,289	0,372	0,168	0,25	0,395
Пурпурный	0,25	0,395	0,372	0,168	0,28	0,048	0,23	0,35
Синий	0,21	0,34	0,195	0,08	0,16	0,128	0,19	0,345
Белый	0,1978	0,4683	Радиус окружности 0,028 ед. на *UV*-плоскости					
Цвет	*x*, ед.	*y*, ед.	*x*, ед.	*y*, ед.	*x*, ед.	*y*, ед.	*x*, ед.	*y*, ед.
Голубой	0,20	0,20	0,1074	0,0933	0,0794	0,1705	0,2077	0,2462
Зеленый	0,289	0,4862	0,2074	0,7722	0,3838	0,6135	0,3493	0,4932
Желтый	0,4381	0,4867	0,4661	0,5339	0,5204	0,4792	0,4756	0,439
Янтарный	0,4756	0,439	0,5204	0,4792	0,5576	0,4406	0,4701	0,3881
Оранжевый	0,4701	0,3881	0,5576	0,4406	0,6005	0,3985	—	—
Красный	0,4515	0,3107	0,6464	0,3531	0,7152	0,2838	—	—
Розовый	0,4091	0,2727	0,5675	0,1854	0,4142	0,1141	0,3621	0,2414
Малиновый	0,3621	0,2414	0,4117	0,1137	0,29	0,0582	0,3134	0,2201
Пурпурный	0,3134	0,2201	0,29	0,0582	0,1952	0,0149	0,2661	0,1799
Синий	0,2417	0,1739	0,1476	0,0269	0,132	0,0469	0,2244	0,1811
Белый	0,3126	0,329	Центр эллипса на *XY*-плоскости					

Для решения задачи поиска координат вершин многоугольников, воспроизводимых на жидкокристаллической панели цветов с максимальными перцепционными отличиями, в цветовых пространствах XY и UV используются две группы прямых линий, образующие:

- четырехугольники в области цветов с максимальными перцепционными отличиями (таблица);
- стороны треугольников цветового охвата средства индикации.

Координаты вершин треугольника цветового охвата средства индикации вычисляют по формулам (1), (2) для красного, зеленого и синего цветов соответственно по программным кодам $RGB = (255, 0, 0); (0, 255, 0); (0, 0, 255)$.

Координаты цветности (u', v') вершин многоугольников, ограничивающих области цветов с максимальными перцепционными отличиями в цветовом пространстве UV, введенном МКО в 1976 г., связаны с координатами цвета XYZ и цветности (x, y) следующими соотношениями [6]:

$$u' = \frac{4X}{X + 15Y + 3Z}, \quad v' = \frac{9Y}{X + 15Y + 3Z},$$

$$X = \frac{9Yu'}{4v'}, \quad Z = \frac{3Y(4 - u')}{4v'} - 5Y,$$

$$u' = \frac{2x}{6y - x + 1{,}5}, \quad v' = \frac{4{,}5y}{6y - x + 1{,}5},$$

$$x = \frac{4{,}5u'}{3u' - 8v' + 6}, \quad y = \frac{2v'}{3u' - 8v' + 6}.$$

Таким образом, может быть выполнен однозначный пересчет координат.

Система (3) является исходной для вычисления координат цветности вершин многоугольников, образующих закрашенные на рис. 3 области цветов $Q_i(x, y)$ для алгоритма, представленного на рис. 1.

Заключение

В отличие от методов случайного поиска [4], где анализируются значения функции поверхности C_i в случайно выбираемых точках (x_i, y_j), наличие граничных условий $Q_i(x, y)$ и априори дискретный характер изменения кода RGB позволяют методу пошагового поиска гарантировать нахождение глобального максимума C_i автоматизированным способом за ограниченное время.

Поскольку алгоритм поиска реализуется на ИЭВМ в составе автоматизированного рабочего места, который подробно рассмотрен в работе [9], у оператора нет необходимости проводить процедуру измерения яркости каждого цвета (оттенка) и процедуру последующей оптимизации поиска максимума контраста изображения на измеренном множестве энергетических характеристик раздельно. Расчеты контраста и измерения яркости осуществляются непосредственно при выполнении основного цикла алгоритма, представленного на рис. 1, автоматизированным способом и только для тех координатных точек, которые оказываются внутри многоугольника, периметр которого ограничен функцией $Q_i(x, y)$ для i-го анализируемого цвета.

Объединение проектных процедур позволяет оператору на порядок сократить время (трудоемкость) поиска оценок $\widehat{RGB}$ значений компонентов кодов RGB, обладающих наилучшими визуальными характеристиками восприятия для человека в пределах цветов с максимальными перцепционными отличиями.

Список литературы

1. **Жаринов И. О., Жаринов О. О.** Бортовые средства отображения информации на плоских жидкокристаллических панелях: учеб. пособие. Информационно-управляющие системы. СПб.: ГУАП, 2005. 144 с.
2. **Barber S.** et al. US Patent 7,417,641 B1: Aeronautical chart display apparatus and method, Aug. 26. 2008.
3. **Костишин М. О., Жаринов И. О., Жаринов О. О.** Исследование визуальных характеристик средств отображения пилотажно-навигационных параметров и геоинформационных данных в авионике // Информационно-управляющие системы. 2014. № 4. С. 61—67.
4. **Гайкович А. И.** Основы теории проектирования сложных технических систем. СПб.: НИЦ "МОРИНТЕХ", 2001. 432 с.
5. **Kumar S. V., Ramana P. V.** Color selection algorithm design for smart lighting application // International Journal of Computer Science and Information Technology & Security. 2014. Vol. 4, N. 1. P. 8—13.
6. **Ibraheem N. A., Hasan M. M., Khan R. Z., Mishra P. K.** Understanding color models: a review // ARPN Journal of science and technology. 2012. Vol. 2, N. 3. P. 265—275.
7. **Hunt R. W.** The reproduction of color. Harrow, Kodak Limited, 2004. 887 p.
8. **Жаринов И. О., Жаринов О. О.** Решение задачи оценки параметров математической модели цветовоспроизведения дисплея с максимальным цветовым охватом на основе интерполяции линии спектральных цветностей // Программная инженерия. 2015. № 10. С. 19—30.
9. **Жаринов И. О., Жаринов О. О.** Исследование распределения оценки разрешающей способности преобразования Грассмана в системах кодирования цвета, применяемых в авионике // Программная инженерия. 2014. № 8. С. 40—47.

An Automatization of Designing of Color Palette of On-Board Indication Equipment Unit by Processing of Energy Characteristics of Colors with Maximal Perceptual Differences

I. O. Zharinov, igor_rabota@pisem.net, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), Saint Petersburg, 197101, Russian Federation, Design Bureau "Electroavtomatika", Saint Petersburg, 198095, Russian Federation
O. O. Zharinov, zharinov73@hotbox.ru, Saint-Petersburg State University of Aerospace Instrumentation, Saint Petersburg, 190000, Russian Federation

Corresponding author:

Zharinov Igor O., Chef of Department, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), 197101, Saint Petersburg, Russian Federation, e-mail: igor_rabota@pisem.net

Received on February 20, 2016
Accepted on February 29, 2016

The practical problem of finding colors to be used for on-board indication equipment color palette, having best characteristics of perception for an observer, is considered. The block diagram of the developed algorithm, which provides automatization of search procedure within automated workstation, is proposed. Main part of the algorithm is triple program loop, which implements the method of complete enumeration, trying different combinations of code values of basic colors (red, green, blue), which are formed by workstation computer. The criterion for any given color to be included into color palette set, is its maximal contrast. It was shown that searching procedure to find colors with best perceptual properties doesn't need to enumerate all possible RGB codes combinations lying within colour gamut triangle, but only particular set codes, relating to areas of colors with maximal perception differences. Reduction of the searching area leads to decrease of duration of automated search procedure. Borders of each given area, corresponding to color with maximal perceptual differences from other ones, were approximated by quadrangles. Vertices of quadrangles are defined by numerical values of chromaticity coordinates on XY- and UV- color planes. Principle of calculation of chromaticity coordinates of vertices of polygons, which contain colors with maximal perceptive differences and may be displayed on the indication equipment, is proposed. Calculation of chromaticity coordinates of vertices of polygons may be performed immediately within the whole searching algorithm, with the use of obtained results of the task of calculation of coordinates of the point of intersection of two straight lines in the Cartesian coordinate system coordinates. Obtained results are presented at the XY- and UV- chromaticity planes as areas, corresponding to the sets of colors, having maximal perceptual differences, which can be displayed on the screen of on-board indication equipment.

Keywords: *colour gamut, image contrast, maximum, color, perceptual differences, avionics*

For citation:

Zharinov I. O., Zharinov O. O. An Automatization of Designing of Color Palette of On-Board Indication Equipment Unit by Processing of Energy Characteristics of Colors with Maximal Perceptual Differences, *Programmnaya Ingeneria,* 2016, vol. 7, no. 5, pp. 215—220

DOI: 10.17587/prin.7.215-220

References

1. **Zharinov I. O., Zharinov O. O.** *Bortovye sredstva otobrazhenija informacii na ploskih zhidkokristallicheskih paneljah* (On-board Display on Flat Liquid Crystal Panels), Saint Petersburg, GUAP, 2005, 144 p. (in Russian).
2. **Barber S.** et al. Aeronautical chart display apparatus and method, US Patent 7,417,641 B1, Aug. 26, 2008.
3. **Kostishin M. O., Zharinov I. O., Zharinov O. O.** Issledovanie vizual'nyh harakteristik sredstv otobrazhenija pilotazhno-navigacionnyh parametrov i geoinformacionnyh dannyh v avionike (Visual characteristics of displaying air navigation parameters and geoinformation data in avionics), *Informacionno-Upravljajushhie Sistemy*, 2014, no. 4, pp. 61—67 (in Russian).
4. **Gajkovich A. I.** *Osnovy teorii proektirovanija slozhnyh tehnicheskih system* (Fundamentals of complex technical systems), Saint Petersburg, NIC "MORINTEH", 2001, 432 p.
5. **Kumar S. V., Ramana P. V.** Color selection algorithm design for smart lighting application, *International Journal of Computer Science and Information Technology & SEcurity*, 2014, vol. 4, no. 1, pp. 8—13.
6. **Ibraheem N. A., Hasan M. M., Khan R. Z., Mishra P. K.** Understanding color models: a review, *ARPN Journal of Science and Technology*, 2012, vol. 2, no. 3, pp. 265—275.
7. **Hunt R. W.** *The reproduction of color*, Harrow, Kodak Limited, 2004, 887 p.
8. **Zharinov I. O., Zharinov O. O.** Reshenie zadachi ocenki parametrov matematicheskoj modeli cvetovosproizvedenija displeja s maksimal'nym cvetovym ohvatom na osnove interpoljacii linii spektral'nyh cvetnostej (The solution to the problem of evaluation of parameters of the mathematical model of color reproduction for display with maximal gamut based of spectrum locus interpolation), *Programmnaya Ingeneria*, 2015, no. 10, pp. 19—30 (in Russian).
9. **Zharinov I. O., Zharinov O. O.** Issledovanie raspredelenija ocenki razreshajushhej sposobnosti preobrazovanija Grassmana v sistemah kodirovanija cveta, primenjaemyh v avionike (Research of properties of an assessment of the resolution of Grassmann's transformation in chromaticity coding systems, applied in avionic equipment), *Programmnaya Ingeneria*, 2014, no. 8, pp. 40—47 (in Russian).

УДК 004.75 + 004.41 DOI: 10.17587/prin.7.221-230

А. С. Шундеев, канд. физ.-мат. наук, вед. науч. сотр., e-mail: alex.shundeev@gmail.com,
НИИ механики МГУ имени М. В. Ломоносова, г. Москва

Программные основы численных методов

Изложен подход к обучению программным основам реализации численных методов. Данный подход был апробирован автором при проведении практических занятий в рамках учебного курса "Работа на ЭВМ и программирование" для студентов первого курса механико-математического факультета МГУ имени М. В. Ломоносова.

Ключевые слова: *арифметика чисел с плавающей точкой, IEEE754, численные методы, стандартная библиотека языка программирования Си, ассемблер YASM, SSE*

Введение

Рассмотрим следующую учебную задачу. Требуется написать программу на языке программирования Си, которая для заданного натурального числа n вычисляет сумму $S_n = 1/1^2 + 1/2^2 + 1/3^2 + ... + 1/n^2$.

Решение необходимо оформить в виде функции, имеющей прототип `double sum(int n)`.

Несмотря на кажущуюся простоту постановки, эта задача может породить целый ряд неправильных решений, каждое из которых, как это ни странно, представляет отдельный интерес. Каждый неправильный вариант функции `sum` может быть положен в основу самостоятельной задачи, в рамках решения которой требуется выявить и объяснить причину возникновения ошибки. Приведем четыре варианта функции `sum`, начиная с неправильных.

Вариант 1 приведен на листинге 1. При малых значениях n программа будет выдавать ответ $S_n = 1$. Однако начиная с некоторого значения n программа будет аварийно завершаться. Например, это произойдет при попытке вычислить $S_{100\,000}$.

```
1. double sum(int n) {
2.     int    k;
3.     double s;
4.     for(k = 1, s = 0.; k <= n; k ++)
5.         s += 1 / (k * k);
6.     return s;
7. }
```

Листинг 1. Первый неправильный вариант функции `sum`

Основная ошибка приведенного решения кроется в строке 5. В языке Си символ / используется как для обозначения целочисленного деления, так и для обозначения деления вещественных чисел. При почти всех $k > 1$ результат вычисления выражения $1/k^2$ будет равен 0.

Вариант 2. Перейти от целочисленного деления к вещественному делению можно, сделав один из операндов (числитель или знаменатель) вещественным. Например, заменить целочисленный литерал `1` на вещественный литерал `1.` (единица с точкой). На листинге 2 приведена соответствующая модификация функции `sum`.

```
1. double sum(int n) {
2.     int    k;
3.     double s;
4.     for(k = 1, s = 0.; k <= n; k ++)
5.         s += 1. / (k * k);
6.     return s;
7. }
```

Листинг 2. Второй неправильный вариант функции `sum`

Этот вариант функции `sum` также является неправильным. При малых значениях n программа будет выдавать результат, похожий на правду, но при попытке вычислить S_{100000} либо вернет ответ *Inf* (бесконечность), либо, как и в предыдущем случае, аварийно завершится.

Вариант 3. Используя отладочную печать или запустив программу из-под отладчика, можно выяснить, что при $k = 65\,536$ результат вычисления выражения k^2 равен 0. Из этого нетрудно сделать вывод, что при больших значениях k некорректно использовать целочисленное умножение. На листинге 3 целочисленное умножение заменено на вещественное умножение путем приведения одного из операндов к вещественному типу.

Конечно, от вычисления квадрата k^2 можно было бы вообще отказаться, заменив выражение `1./((double) k*k)` на выражение с двумя последовательными делениями `1./k/k`.

```
1. double sum(int n) {
2.     int    k;
3.     double s;
4.     for(k = 1, s = 0.; k <= n; k ++)
5.         s += 1. / ((double) k * k);
6.     return s;
7. }
```

Листинг 3. Третий почти правильный вариант функции `sum`

Вариант 4. В приведенных вариантах функции `sum` суммирование осуществляется в порядке возрастания параметра *k*. Порядок суммирования может быть изменен. Сумму можно вычислять в порядке убывания параметра *k*. Этот вариант является наиболее правильным, и он приведен на листинге 4.

```
1. double sum(int n) {
2.     int    k;
3.     double s;
4.     for(k = n, s = 0.; k >= 1; k --)
5.         s += 1. / ((double) k * k);
6.     return s;
7. }
```

Листинг 4. Четвертый правильный вариант функции `sum`

Рассмотренную задачу можно использовать в качестве инструментария для проверки знаний. Например, для студента, знакомого с основами программирования, приемлемым результатом будет третий вариант решения задачи. От специалиста, желающего эффективно разрабатывать и применять численные методы, следует ожидать четвертого варианта решения задачи, а также способность объяснить ошибки предыдущих вариантов и последствия, к которым они приводят.

Таким образом, от специалиста требуется наличие согласованных знаний и навыков одновременно в нескольких областях. Это, прежде всего, знание теоретических основ машинной арифметики [1]. Необходимо понимание принципов реализации машинной арифметики в целевой вычислительной системе, а также владение инструментальными средствами, позволяющими контролировать ее параметры. Настоящая статья является логическим продолжением предыдущей работы автора [2] и посвящена изложению принятого автором подхода к организации учебного процесса по формированию подобных знаний и навыков.

Статья написана на основе материалов, которые автор использует при проведении практических занятий в рамках учебного курса "Работа на ЭВМ и программирование" на механико-математическом факультете МГУ имени М. В. Ломоносова, и имеет следующую структуру. Она состоит из введения, семи разделов и заключения. В разд. 1 описаны способы машинного представления целых чисел. Дано объяснение, почему происходит деление на нуль в первом и втором вариантах функции `sum`. В разд. 2 описаны способы машинного представления вещественных чисел в соответствии со стандартом IEEE 754 [1]. В качестве примера рассмотрены числа с плавающей точкой двойной точности. В разд. 3 рассмотрены средства языка программирования Си, позволяющие вычислять порядковые номера чисел с плавающей точкой, а также формировать числа с плавающей точкой в соответствии с заданными кодирующими последовательностями битов. Разд. 4 посвящен рассмотрению двух взаимосвязанных понятий — округление вещественных чисел и вычислительная погрешность. Описаны стандартные типы ошибок, возникающие в арифметических операциях над числами с плавающей точкой. Дано объяснение, почему вариант 4 функции `sum` является правильным. Разд. 5 и 6 посвящены рассмотрению стандартных методов округления. В разд. 7 в качестве примера реализации машинной арифметики рассмотрена технология SSE[1] [3], реализованная в современных процессорах фирмы Intel. Дано объяснение, почему после возникновения ошибки при использовании второго варианта функции `sum` программа может либо аварийно завершиться, либо продолжить свое выполнение, оперируя специальным значением *Inf*.

1. Представление целых чисел

Рассмотрим способы машинного представления целых чисел. Для этого используются битовые последовательности. Их также называют кодирующими последовательностями. Обычно используются последовательности длины $N = 8, 16, 32, 64, 128$ бит. Соответственно говорят об N-разрядных целых числах. Отдельно рассматриваются способы представления для беззнаковых целых чисел и целых чисел со знаком.

Беззнаковое N-разрядное целое число кодируется последовательностью бит длины N, представляющей собой N-разрядную запись этого числа в двоичной системе счисления.

Для целых чисел со знаком используется прямой, обратный и дополнительный способы кодирования. Эти способы используют старший разряд битовой последовательности для кодирования знака числа (0 кодирует плюс, 1 кодирует минус). Остальные $N - 1$ (мантисса) разрядов кодируют абсолютную величину числа. Мантисса положительного числа представляет собой $(N - 1)$-разрядную запись этого числа в двоичной системе счисления.

Разберемся с кодированием отрицательных чисел. При прямом способе кодирования в старший разряд помещается 1, а мантисса представляет собой $(N - 1)$-

[1] SSE (Single SIMD Extensions; потоковые SIMD-расширения), SIMD (Single Instruction — Multiple Data; одна инструкция — множество данных).

разрядную запись абсолютной величины этого числа в двоичной системе счисления. Обратный код получается инвертированием всех разрядов кода абсолютной величины числа. Дополнительный код получается из обратного кода путем добавления 1 к младшему разряду. В табл. 1 приведены примеры прямого, обратного и дополнительного способов кодирования 8-разрядных целых чисел.

Таблица 1

Примеры кодирования отрицательных целых чисел

Тип кода	Пример числа	
	−1	−127
Прямой	1000 0001	1111 1111
Обратный	1111 1110	1000 0000
Дополнительный	1111 1111	1000 0001

Вернемся к задаче суммирования. Входной аргумент функции `sum` имеет тип `int`, являясь, тем самым, 32-разрядным целым числом со знаком. Максимальным значением типа `int` является число 2 147 483 647. Наибольшим целым числом, квадрат которого может быть представлен в виде значения типа `int`, является число 46 340. Поэтому второй вариант функции `sum` будет корректно вычислять S_n только при $1 \leqslant n \leqslant 46\,340$.

Рассмотрим подробнее, что произойдет при попытке вычислить $S_{46\,341}$ с помощью второго варианта функции `sum`. На последней итерации цикла будет осуществлена попытка вычислить $S_{46\,341} = S_{46\,340} + 1/46\,341^2$. Вычисленное на предыдущей итерации цикла значение $S_{46\,340}$ является корректным. Будет ли корректно вычислено значение дроби $1/46\,341^2$?

Имеем $46\,341^2 = 2\,147\,488\,281$. Запись этого числа в двоичной системе счисления имеет вид `1000 0000 0000 0000 0001 0010 0001 1001`. Оно не может быть представлено как положительное значение типа `int`. При дополнительном кодировании 32-разрядных чисел эта битовая последовательность задает отрицательное число −2147479015. При попытке вычислить $46\,341^2$ произойдет целочисленное переполнение и будет получен некорректный результат $S_{46\,341} = S_{46\,340} - 1/2\,147\,479\,015$. Таким образом, начиная со значения параметра $n > 46\,340$, второй вариант функции `sum` будет возвращать неправильный результат.

Рассмотрим, что происходит при попытке вычислить $S_{65\,536}$. На последней итерации цикла приходится вычислять дробь $1/65\,536^2$. Имеем $65\,536^2 = 4\,294\,967\,296$. Запись этого числа в двоичной системе счисления имеет вид

`1 0000 0000 0000 0000 0000 0000 0000 0000`.

Оно не может быть представлено как значение типа `int`. При попытке вычисления этого числа произойдет целочисленное переполнение. Результатом умножения будет значение типа `int`, которое кодируется младшими 32 битами кодирующей последовательности числа $65\,536^2$. Таким образом, результатом будет число 0. Далее, произойдет попытка деления на 0. Это приведет к аварийному завершению программы в случае первого варианта функции `sum`. В случае второго варианта функции `sum` это приведет либо к аварийному завершению программы, либо к получению в результате вычисления значения *Inf*.

2. Представление вещественных чисел

Битовые последовательности также используются для представления вещественных чисел. Например, числа с плавающей точкой двойной точности [1], которым соответствует тип `double` в языке Си, представляются последовательностями длиной в 64 бита. Кодирующая последовательность имеет следующий формат.

$$s p_{10} p_9 \ldots p_1 p_0 m_1 m_2 \ldots m_{51} m_{52}. \quad (1)$$

Старший бит s задает знак числа. Последовательность битов $p_{10} \ldots p_0$ определяет показатель степени. Будем интерпретировать эту последовательность как запись некоторого числа p в двоичной системе счисления. Наконец, последовательность битов $m_1 \ldots m_{52}$ кодирует мантиссу. Возможны следующие варианты.

Вариант 1: $p \neq 0$ и $p \neq 2047$. Это значит, что последовательность $p_{10} \ldots p_0$ не состоит целиком только из нулей или только из единиц. В этом случае формула (1) задает нормализованное число, которое вычисляется по следующей формуле:

$$(-1)^s \, 1.m_1 m_2 \ldots m_{51} m_{52} \, 2^{p-1023}. \quad (2)$$

В формуле (2) число $1.m_1 m_2 \ldots m_{51} m_{52}$ записано в двоичной системе счисления.

Вариант 2: $p = 0$ и мантисса не состоит целиком из нулей. В этом случае формула (1) задает денормализованное число, которое вычисляется по следующей формуле:

$$(-1)^s \, 0.m_1 m_2 \ldots m_{51} m_{52} \, 2^{-1022}. \quad (3)$$

В формуле (3) число $0.m_1 m_2 \ldots m_{51} m_{52}$ записано в двоичной системе счисления.

Вариант 3: $p = 0$ и мантисса состоит целиком из нулей. В этом случае формула (1) задает нулевые числа. При $s = 0$ (1) задает нуль, а при $s = 1$ задает так называемый отрицательный нуль.

Вариант 4: $p = 2047$ и мантисса состоит целиком из нулей. В этом случае формула (1) задает значение, которое называется бесконечностью. При $s = 0$ это значение называется положительной бесконечностью *Inf*, а при $s = 1$ — отрицательной бесконечностью *−Inf*.

Вариант 5: $p = 2047$ и мантисса не состоит целиком из нулей. В этом случае формула (1) задает значение, которое называется "не число" *NaN*.

Обозначим через **F** множество всех чисел с плавающей точкой двойной точности (далее, просто числа

с плавающей точкой), а через **R** — множество всех вещественных чисел. Множество **F** конечно и состоит из 2^{64} элементов. Пересечение множеств $\mathbf{F} \cap \mathbf{R}$ состоит из нуля, нормализованных и денормализованных чисел.

Последовательность битов (1) может кодировать как число с плавающей точкой $x \in \mathbf{F}$, так и беззнаковое целое число $n(x)$, которое будем называть *порядковым номером x*. Порядковые номера обладают следующими свойствами.

Для положительных чисел $x_1, x_2 \in \mathbf{F} \cap \mathbf{R}$ неравенство $x_1 < x_2$ выполняется тогда и только тогда, когда выполняется неравенство $n(x_1) < n(x_2)$. Если $n(x_2) = n(x_1) + 1$, то открытый интервал вещественных чисел (x_1, x_2) не содержит чисел с плавающей точкой.

Для отрицательных чисел $x_1, x_2 \in \mathbf{F} \cap \mathbf{R}$ неравенство $x_1 < x_2$ выполняется тогда и только тогда, когда выполняется неравенство $n(x_1) > n(x_2)$. Если $n(x_2) = n(x_1) - 1$, то открытый интервал вещественных чисел (x_1, x_2) не содержит чисел с плавающей точкой.

3. Вычисление порядковых номеров чисел с плавающей точкой

Язык программирования Си предоставляет эффективные средства для решения задач вычисления порядкового номера числа с плавающей точкой, нахождения числа с плавающей точкой по его порядковому номеру, формирования числа с плавающей точкой, соответствующего заданной кодирующей последовательности битов. Рассмотрим некоторые из них.

Число с плавающей точкой и его порядковый номер кодируются одинаковой последовательностью битов. Поэтому последовательность из 8 байт, хранящуюся в некоторой области памяти адресного пространства выполняющейся программы, можно интерпретировать и как бинарное представление числа с плавающей точкой, так и как бинарное представление беззнакового целого числа, которое будет являться порядковым номером этого числа с плавающей точкой.

На листинге 5 с помощью операций над указателями область памяти, выделенная под хранение переменной типа `double`, интерпретируется как область памяти, хранящая значение целочисленного типа `uint64 _ t`. Это значение сохраняется в переменную `i` и является порядковым номером числа 1,5.

```
1. #include <inttypes.h>
2. ...
3. double   d = 1.5;
4. uint64_t i = *(uint64_t *)&d;
```

Листинг 5. Пример использования указателей

На листинге 6 продемонстрирован другой подход. Определяется новый тип данных `real _ t` в виде объединения с двумя полями типа `double` и типа `uint64 _ t`. Со значением типа `real _ t` можно одновременно работать и как с числом с плавающей точкой (через поле `d`), и как с его порядковым номером (через поле `i`). Порядковым номером числа с плавающей точкой 1,5 является целое число 4 609 434 218 613 702 656. Поэтому переменные `r1` и `r2` будут иметь одинаковое значение.

```
01. #include <inttypes.h>
02. ...
03. typedef union {
04.    double   d;
05.    uint64_t i;
06. } real_t;
07. ...
08. real_t r1, r2;
09. r1.d = 1.5;
10. r2.i = 4609434218613702656;
```

Листинг 6. Пример использования объединений

Задача формирования числа с плавающей точкой, соответствующего заданной кодирующей последовательности битов, может быть решена двумя способами. Первый способ базируется на использовании битовых операций и введенного типа данных `real _ t`. Второй способ базируется на использовании шестнадцатеричных литералов. Рассмотрим каждый из них.

В качестве примера возьмем максимальное нормализованное число $N_{\max}$ и минимальное положительное денормализованное число $D_{\min}$. Число $N_{\max}$ кодируется последовательностью битов

```
0 11111111110 1111111111111111111111111111111111111111111111111111,
```

а число $D_{\min}$ — последовательностью битов

```
0 00000000000 0000000000000000000000000000000000000000000000000001.
```

Листинг 7 содержит фрагмент программы, в результате выполнения которого значением выражения `r1.d` будет число $N_{\max}$, а значением выражения `r2.d` будет число $D_{\min}$.

```
1. real_t r1, r2;
2. r1.i = ~(UINT64_C(1) << 63 | UINT64_C(1) << 52);
3. r2.i = 1;
```

Листинг 7. формирование чисел $N_{\max}$ и $D_{\min}$

Константы типа `double` могут быть заданы с помощью шестнадцатеричных литералов. На листинге 8 представлено, как с их помощью задаются числа $N_{\max}$ и $D_{\min}$. Отметим, что формат шестнадцатеричного литерала хорошо согласуется с представлениями (2) и (3). Отличий всего два. Для задания мантиссы ис-

```
1. double Nmax = 0x1.fffffffffffffp+1023,
2.        Dmin = 0x0.0000000000001p-1022;
```

Листинг 8. Пример использования шестнадцатеричных литералов

пользуется шестнадцатеричная система счисления вместо двоичной системы. Показатель степени числа 2 задан явно.

Более подробно материал данного раздела изложен в работе автора [2].

4. Округление вещественных чисел и вычислительная погрешность

Как уже было отмечено ранее, не каждое вещественное число может быть точно представлено в виде числа с плавающей точкой. В этом случае, вместо вещественного числа $x \in \mathbf{R}$ используется некоторое приближенное к нему значение $R(x) \in \mathbf{F}$. Функция R называется *функцией округления*.

Необходимость в округлении возникает в следующих ситуациях. В текстах программ вещественные константы обычно задаются с помощью десятеричных литералов. В результате компилятору во время трансляции программы приходится округлять значение каждого десятеричного литерала, когда это значение не может быть точно представлено в виде числа с плавающей точкой. Примером такого литерала может служить 0,1. Отметим, что такая необходимость не возникает при использовании шестнадцатеричных литералов, однако, они не так широко используются.

Потребность в округлении также возникает при выполнении арифметических операций. Например, если $x_1, x_2 \in \mathbf{R} \cap \mathbf{F}$, то результатом вычисления выражения $x_1 + x_2$ будет значение $R(x_1 + x_2) \in \mathbf{F}$.

Рассмотрим общие для всех функций округления свойства. Очевидно, что должно выполняться равенство $R(x) = x$ для всякого $x \in \mathbf{F}$. Точно также очевидно, что $R(y) \neq y$ для всякого $y \in \mathbf{R} \backslash \mathbf{F}$. Несмотря на распространенность ситуации, когда $R(y) \neq y$, она трактуется как ошибка в операции над числами с плавающими точками (далее, ошибками) *"неточный результат"* (*inexact*). Выделяют три случая, схематично изображенные на рис. 1.

Случай 1, $D_{\min} < y < N_{\max}$. В этом случае существуют два подряд идущих числа с плавающей точкой $x_1, x_2 \in \mathbf{F}$ такие, что $y \in (x_1, x_2)$. В качестве значения $R(y)$ выбирается один из концов этого интервала.

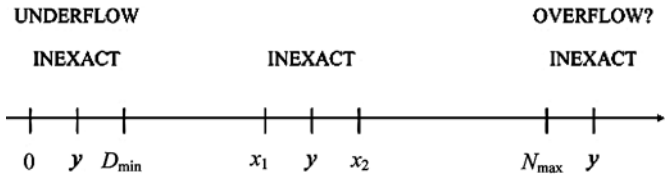


Рис. 1. Округление и ошибки. Три случая

Случай 2, $y > N_{\max}$. В качестве значения $R(y)$ будет выступать либо максимальное нормализованное число $N_{\max}$, либо специальное значение бесконечность *Inf*. Если $R(y) = Inf$, то это трактуется как ошибка *"переполнение"* (*overflow*). В зависимости от реализации функции округления ситуация, когда $R(y) = N_{\max}$ также может трактоваться как переполнение.

Случай 3, $0 < y < D_{\min}$. В качестве значения $R(y)$ будет выступать либо нуль, либо минимальное положительное денормализованное число $D_{\min}$. Эта ситуация всегда трактуется как ошибка *"антипереполнение"* (*underflow*).

Аналогично можно выделить и рассмотреть три случая для отрицательного числа y.

К числу стандартных ошибок также относят ошибки *"деление на нуль"* (*divbyzero*) и *"недействительную операцию"* (*invalid*). Ошибка деления на нуль уже встречалась в статье при рассмотрении второго вариант функции `sum`. При возникновении этой ошибки результатом арифметической операции становится специальное значение бесконечность *Inf*. В этом примере также было показано, что прибавление к значению *Inf* числа снова порождает значение *Inf*, и это выглядит вполне логично. Однако для ряда случаев трудно дать разумную интерпретацию результату арифметической операции. Например, $Inf - Inf$, $0 \times Inf$, $0/0$, Inf / Inf. В этих случаях результат арифметической операции полагают равным специальному значению *NaN* и трактуют эту ситуацию как ошибка "недействительная операция".

С понятием округления тесно связано понятие вычислительной погрешности. В силу необходимости выполнения округлений во время выполнения последовательности арифметических операций получается не точный результат x, а его некоторое приближение x^*. Если имеет место неравенство $|x - x^*| \leqslant \Delta$, то величину Δ называют *абсолютной погрешностью* приближения.

Существуют теоретические оценки [4] для абсолютных погрешностей результатов арифметических операций. Например, при вычислении суммы n чисел $x_1 + x_2 + ... + x_n$ погрешность будет иметь следующий вид:

$$E_1|x_1| + E_2|x_2| + ... + E_n|x_n|, \qquad (4)$$

где $E_1 > E_2 > ... > E_n > 0$. Оценка (4) будет минимальной, если осуществлять суммирование в порядке возрастания абсолютных величин слагаемых. С этой точки зрения четвертый вариант функции `sum` является более правильным.

5. Стандартные методы округления (случай 1)

В стандарте [1] приведены четыре основных метода округления *RN*, *RU*, *RD*, *RZ*. Результаты их использования для случая, когда округляется число из интервала $(D_{\min}, N_{\max})$ или интервала $(-N_{\max}, -D_{\min})$, схематично изображены на рис. 2. На рис. 2 выделены два подряд идущих числа с плавающей точкой

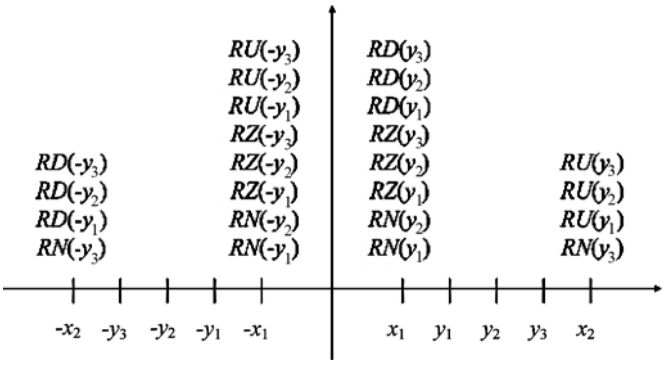


Рис. 2. Стандартные способы округления (случай 1)

$x_1, x_2 \in \mathbf{F}$ $(x_1 > 0)$, а также три вещественных числа $y_1, y_2, y_3 \in (x_1, x_2)$. При этом y_2 является серединой интервала (x_1, x_2), и выполняются строгие неравенства $y_1 < y_2 < y_3$.

Метод *RN* осуществляет округление в сторону ближайшего числа с плавающей точкой. Поэтому $RN(y_1) = x_1$ и $RN(y_3) = x_2$. Вещественное число y_2 находится ровно посередине интервала (x_1, x_2) и равноудалено от его концов. В этом случае рассматривают порядковые номера концов интервала и выбирают конец интервала с четным номером. На рис. 2 изображена ситуация, когда $n(x_1)$ четно, поэтому $RN(y_2) = x_1$.

Метод *RU* осуществляет округление в сторону положительной бесконечности. Всегда выбирается правый конец интервала, поэтому $RU(y_1) = RU(y_2) = RU(y_3) = x_2$. Метод *RD* осуществляет округление в сторону отрицательной бесконечности. Всегда выбирается левый конец интервала, поэтому $RD(y_1) = RD(y_2) = RD(y_3) = x_2$. Наконец, метод *RZ* осуществляет округление в сторону нуля. Всегда выбирается конец, ближайший к нулю, поэтому, например, $RZ(y_2) = x_1$ и $RZ(-y_2) = -x_1$.

При закреплении данного материала на практических занятиях можно в качестве x_1 выбрать число 2^{56}, при этом окажется, что $x_2 - x_1 = 16$. Серединой интервала (x_1, x_2) будет число $y_2 = x_1 + 8$. В качестве y_1 и y_3 можно взять следующие числа. Пусть числа $a, b \in \mathbf{F}$ такие, что $n(a) = n(8) - 1$ и $n(b) = n(8) + 1$, тогда положим $y_1 = x_1 + a$ и $y_3 = x_1 + b$.

Рекомендуется написать программу, которая в разных режимах округления вычисляет y_1, y_2, y_3 и сравнивает их с x_1, x_2.

6. Стандартные методы округления (случаи 2 и 3)

Результаты использования стандартных методов округления для ситуации, когда округляется число, превосходящее N_{max}, схематично изображены на рис. 3.

Если проанализировать формулу (2), то можно обнаружить, что вслед за числом N_{max} должно было бы следовать число 2^{1024}. Однако это число не было включено в состав чисел с плавающей точкой. Его "место" было отдано под специальное значение *Inf*.

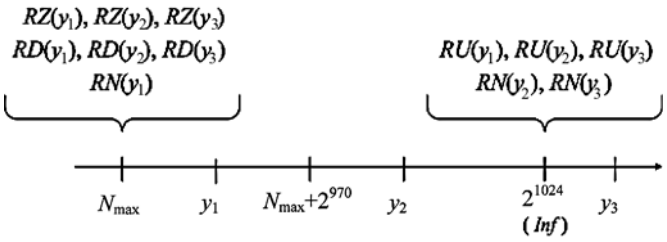


Рис. 3. Стандартные способы округления (случай 2)

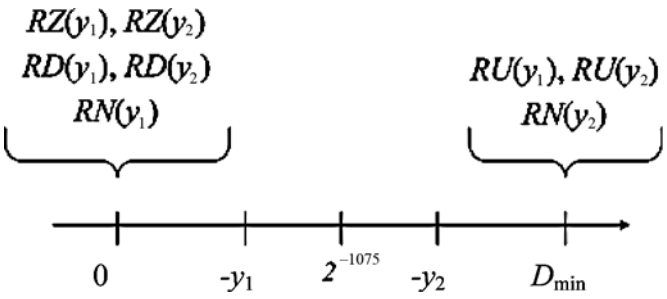


Рис. 4. Стандартные способы округления (случай 3)

Несложно вычислить, что интервал $(N_{max}, 2^{1024})$ имеет длину 2^{971}. Его серединой является число $N_{max} + 2^{970}$. На рис. 3 выделены три числа $y_1, y_2, y_3 \in \mathbf{R}$ такие, что $N_{max} < y_1 < N_{max} + 2^{970} < y_2 < 2^{1024} < y_3$. Покажем, как эти числа округляются стандартными методами.

Метод *RN* округляет следующим образом $RN(y_1) = N_{max}$, $RN(y_2) = RN(N_{max} + 2^{970}) = RN(y_3) = Inf$. При округлении чисел y_2, $N_{max} + 2^{970}$, y_3 возникает ошибка "переполнение".

Результатом округления с помощью метода *RU* будет специальное значение *Inf*, что вызовет ошибку "переполнение".

В рассматриваемой ситуации методы *RD* и *RZ* ведут себя одинаковым образом. Результатом округления рассматриваемых чисел будет число N_{max}. Однако округление числа y_3 вызовет ошибку "переполнение".

При закреплении материала на практических занятиях можно в качестве y_1, y_2, y_3 выбрать соответственно числа $N_{max} + 2^{969}$, $N_{max} + 1.f\,2^{970}$ и $N_{max} + N_{max}/2$. Рекомендуется написать программу, вычисляющую их в разных режимах округления.

Результаты использования стандартных методов округления для ситуации, когда округляется число из интервала $(0, D_{min})$, схематично изображены на рис. 4. Правила округления этого случая совпадают с правилами, описанными при рассмотрении случая 1. В результате округления всегда возникает ошибка "антипереполнение".

7. Реализация машинной арифметики

В качестве примера реализации машинной арифметики рассмотрим технологию SSE, реализованную в современных процессорах фирмы Intel. Данная технология включает в себя набор специализированных регистров и инструкций. Среди регистров

следует отметить XMM-регистры и регистр MXCSR. В XMM-регистры могут быть загружены значения чисел с плавающей точкой для последующего выполнения арифметических операций над ними. При этом могут быть загружены как индивидуальные значения, так и наборы значений. Регистр MXCSR используется для задания параметров выполнения операций над числами с плавающей точкой и хранит состояние их выполнения.

Будут рассмотрены инструкции двух типов. Инструкции первого типа используются для выполнения арифметических операций. При этом выделяются индивидуальные инструкции, оперирующие над индивидуальными значениями, и пакетные инструкции, оперирующие над наборами значений. Например, с помощью пакетной инструкции можно выполнить сразу две операции сложения. Инструкции второго типа позволяют получить и заменить состояние регистра MXCSR.

Рассмотрим программу, представленную на листинге 9. Программа выполняет две операции сложения. Первые слагаемые задаются в глобальном массиве `a`, вторые слагаемые задаются в глобальном массиве `b`, результаты сохраняются в глобальный массив `x`.

```
01. double a[2] = {1.0, 2.0},
02.        b[2] = {1.5, 2.5},
03.        x[2];
04.
05. int main(void) {
06.    x[0] = a[0] + b[0];
07.    x[1] = a[1] + b[1];
08.    return 0;
09. }
```

Листинг 9. Тестовая программа на языке Си

Схема трансляции программы включает стадии препроцессирования, трансляции в ассемблер, ассемблирования и компоновки. При использовании компилятора GCC получить промежуточный результат трансляции в ассемблер можно, выполнив следующую команду:

```
gcc -S -masm = intel -O3 prog.c
```

Удалив лишние метки и инструкции, а также выполнив переименование идентификаторов, можно получить программу на языке ассемблера аналогичную той, что приведена на листинге 10.

Строкам 6—7 листинга 9 соответствуют строки 13—15 листинга 10. С помощью одной инструкции `movapd` в регистр XMM0 загружаются сразу два значения типа `double` (элементы массива `a`). С помощью одной инструкции `addpd` вычисляются сразу две суммы. Суммируются значения, ранее сохраненные в регистр XMM0, с элементами массива `b`.

```
01.    section    .data
02. align          16
03. a              dq        1.0, 2.0
04. b              dq        1.5, 2.5
05.
06.    section    .bss
07. x              resq      2
08.
09.    section    .text
10.    global     _start
11.
12. _start:
13.    movapd     xmm0,     [a]
14.    addpd      xmm0,     [b]
15.    movupd     [x],      xmm0
16.
17.    mov        rdi,      0
18.    mov        rax,      60
19.    syscall
```

Листинг 10. Результат трансляции программы из листинга 9 в ассемблер

Вычисленные суммы содержатся в регистре XMM0. Далее, с помощью одной инструкции `movupd` две вычисленные суммы копируются из регистра XMM0 в массив `x`.

В приведенном примере использование пакетных инструкций `movapd`, `addpd`, `movupd` позволяет увеличить скорость вычислений в 2 раза. Без их использования пришлось бы выполнить шесть индивидуальных инструкций (четыре `movsd`, две `addsd`), оперирующих с единичными значениями.

Рассмотренный пример позволяет сделать важный вывод. При программировании численных методов на высокоуровневых языках программирования (подразумеваются языки Си или Си++) рекомендуется анализировать результат промежуточной трансляции исходной программы в ассемблер. Необходимо обращать внимание на то, какие инструкции были выбраны компилятором для выполнения арифметических операций (пакетные инструкции или индивидуальные инструкции, оперирующие единичными значениями). Следует пытаться структурировать исходный текст программы таким образом, чтобы компилятор по возможности порождал использование пакетных инструкций.

На листинге 11 приведен пример программы на языке ассемблера YASM [3], в которой используются индивидуальные инструкции. В этой программе объявлены четыре глобальных переменных `a`, `b`, `c`, `d` типа `double` (строки 2—5). Переменным `a`, `b`, `c` присвоены значения 0,5, 1,0, 3,0 соответственно. В строках 11—13 вычисляется значение выражения `d = a + b`, а в строках 15—17 вычисляется значение выражения `d = b/c`.

Рассмотрим, как меняется состояние управляющего регистра MXCSR по мере выполнения арифметических операций. Для этого программу нужно запустить из-под отладчика GDB, и сделать оста-

```
01. section    .data
02. a              dq        0.5
03. b              dq        1.0
04. c              dq        3.0
05. d              dq        0.0
06.
07.     section    .text
08.     global     _start
09.
10. _start:
11.     movsd      xmm0,     [a]
12.     addsd      xmm0,     [b]
13.     movsd      [d],      xmm0
14.
15.     movsd      xmm0,     [b]
16.     divsd      xmm0,     [c]
17.     movsd      [d],      xmm0
18.
19.     mov        rdi,      0
20.     mov        rax,      60
21.     syscall
```

Листинг 11. Пример использования индивидуальных инструкций

новки на строках 11, 13, 17 до выполнения каких-либо арифметических операций, после выполнения суммирования, после выполнения деления соответственно.

Для просмотра состояния регистра MXCSR можно воспользоваться командой `p/t $mxcsr`. Результатом будет значение регистра MXCSR в двоичной системе счисления. В первых двух случаях значением будет последовательность `1111110000000`, а в последнем случае — `1111110100000`. Вычисление суммы 0,5 + 1,0 не могло привести к какой-либо ошибке. Поэтому состояние регистра MXCSR не было изменено. Выполнение деления 1,0/3,0 приводит к ошибке "неточный результат". Это событие было отражено изменением значения пятого разряда (нумерация начинается с нуля) регистра MXCSR. Значение изменилось с 0 на 1.

Обозначения для шестнадцати младших управляющих разрядов регистра MXCSR приведены в табл. 2. Разряды PM, UM, OM, ZM, DM, IM являются масками исключений, а разряды PE, UE, OE, ZE, DE, IE — флагами исключений, соответственно, для ошибок "неточный результат", "антипереполнение", "переполнение", "деление на нуль", "денормализованный операнд", "недействительная операция". Отметим, что помимо ранее рассмотренных стандартных ошибок процессоры фирмы Intel поддерживают возможность работы с дополнительной ошибкой — "*денормализованный операнд*".

Значение флага, равное 1, означает появление в процессе вычислений ошибки соответствующего типа. Заметим, что нельзя узнать число возникших ошибок заданного типа, можно лишь проверить, что хотя бы одна такая ошибка произошла. Установка для соответствующей маски значения 0 приведет к тому, что при возникновении соответствующей ошибки будет осуществлено прерывание программы, и управление будет передано соответствующему обработчику. По умолчанию, для ошибок в операциях над числами с плавающей точкой такой обработчик проведет аварийное завершение программы.

При обсуждении второго варианта функции `sum` было отмечено, что после деления на нуль будет либо вычислен результат *Inf*, либо произойдет аварийное завершение программы. Первый случай соответствует значению 1 маски ZM, а второй случай — значению 0. В обоих случаях флагу ZM будет установлено значение 1. Получив в процессе вычислений значение *Inf*, можно проанализировать состояние флагов OE и ZE и понять, в результате какой ошибки ("переполнение" или "деление на нуль") было получено такое значение. Подобный анализ может существенно повысить эффективность отладки программы.

В табл. 3 приведены фрагменты программ, позволяющие установить режим прерывания для ошибки "неточный результат". Для маски PM устанавливается значение 0.

Таблица 2

Регистр MXCSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE

Таблица 3

Установка прерывания для ошибки "неточный результат"

Ассемблер	Си

`1. stmxcsr  [rsp - 8]`
`2. mov  rax,  [rsp - 8]`
`3. and  ah,  0efH`
`4. mov  [rsp - 8],  rax`
`5. ldmxcsr  [rsp - 8]`
	`1. #include <fenv.h>`
	`2. ...`
	`3. feenableexcept(FE _ INEXACT);`

Таблица 4

Установка режима округления "к нулю"

Ассемблер	Си
1. stmxcsr [rsp - 8] 2. mov rax, [rsp - 8] 3. and ah, 9fH 4. or ah, 60H 5. mov [rsp - 8], rax 6. ldmxcsr [rsp - 8]	1. #include <fenv.h> 2. ... 3. fesetround(FE _ TOWARDZERO);

В левом столбце приведен вариант на языке ассемблера YASM [3]. Следует обратить внимание на две инструкции `stmxcsr` и `ldmxcsr`. Первая инструкция позволяет получить текущее состояние регистра MXCSR, а вторая — загрузить в этот регистр новое значение. В первой и второй строчках состояние регистра MXCSR копируется в регистр RAX через промежуточную локальную переменную. В третьей строчке разряды регистра RAX с номерами 8—15 побитово перемножаются на число, имеющее двоичное представление `1110 1111`. Обнуляется разряд, соответствующий маске PM, а все остальные разряды остаются без изменения. В четвертой и пятой строках измененное состояние регистра RAX через локальную переменную загружается в регистр MXCSR.

В правом столбце приведен эквивалентный фрагмент программы на языке программирования Си. Он сводится к одному вызову стандартной функции `feenableexcept`, позволяющей установить режим прерывания для ошибки заданного типа. Отметим, что в заголовочном файле `fenv.h` определены прототипы для целого набора стандартных функций, позволяющих регулировать параметры и отслеживать состояние выполнения операций над числами с плавающей точкой. Приведем некоторые из них.

Функция `fedisableexcept` позволяет выключить режим прерывания для заданного типа ошибок, а функция `fetestexcept` проверяет, имели ли место ошибки заданного типа. С помощью функции `feclearexcept` можно очистить флаги ошибок, а с помощью функции `fesetround` установить режим округления.

За режим округления отвечают два разряда, обозначенные в табл. 2 как RC. Значение 00 этих разрядов означает округление в сторону ближайшего числа с плавающей точкой, значение 01 — округление в сторону отрицательной бесконечности, значение 10 — округление в сторону положительной бесконечности, значение 11 — округление к нулю.

В табл. 4 приведены фрагменты программ, позволяющие устанавливать режим округления к нулю. В левом столбце приведен вариант на языке ассемблера YASM. Как и в ранее рассмотренном примере, сначала в первой и второй строках с помощью инструкции `stmxcsr` текущее состояние управляющего регистра MXCSR копируется в регистр RAX через промежуточную локальную переменную. В третьей и четвертой строках модифицируется состояние регистра RAX. В разряды с номерами 14 и 13 записываются единицы. В пятой и шестой строках с помощью инструкции `ldmxcsr` измененное состояние регистра RAX загружается в управляющий регистр MXCSR через промежуточную локальную переменную.

В правом столбце приведен эквивалентный фрагмент программы на языке Си. Он сводится к одному вызову стандартной функции `fesetround`.

Заключение

Представлен принятый автором подход к обучению основам программирования численных методов. В рамках этого подхода учащийся получает необходимые теоретические знания и вырабатывает практические навыки работы в трех взаимосвязанных областях. Первая область — это теоретические основы машинной арифметики. Вторая область — это реализация машинной арифметики в целевой вычислительной системе. В качестве примера была рассмотрена архитектура SSE, реализованная в современных процессорах фирмы Intel. Третья область — инструментальные средства разработки программ. В качестве примера были рассмотрены соответствующие средства стандартной библиотеки языка программирования Си, ассемблер YASM, возможности компилятора GCC и отладчика GDB.

Список литературы

1. **IEEE Std 754—2008.** IEEE Standard for Floating-Point Arithmetic. IEEE, 29 August 2008, 58 p.
2. **Шундеев А. С.** Введение в стандарт IEEE 754 // Программная инженерия. 2013. № 3. С. 44—47.
3. **Seyfarth R.** Introduction to 64 Bit Intel Assembly Language Programming for Linux. CreateSpace Independent Publishing Platform, 2011. 252 p.
4. **Корнев А. А., Чижонков Е. В.** Упражнения по численным методам. Часть II / под ред. Н. С. Бахвалова. М.: Изд-во ЦПИ при механико-математическом факультете МГУ им. М. В. Ломоносова, 2003. 200 с.

The Programming's Basics for Numerical Methods

A. S. Shundeev, Institute of Mechanics, Lomonosov Moscow State University, 119991, Moscow, Russian Federation

Corresponding author:

Shundeev Aleksandr S., Leading Researcher, Institute of Mechanics, Lomonosov Moscow State University, 119991, Moscow, Russian Federation,
e-mail: alex.shundeev@gmail.com

Received on February 19, 2016
Accepted on March 02, 2016

This article describes an approach to teaching the programming basics for the software implementation of numerical methods. This approach was used by the author during the practical sessions in the training course "Working on a computer and programming" for the first-year Faculty of Mechanics and Mathematics students of Lomonosov Moscow State University.

In this approach, the student receives the necessary theoretical knowledge and develops practical skills in three interrelated areas. The first area is the theoretical foundations of computer arithmetics. The second area is the implementation of machine arithmetics in the target computer system. As an example of the architecture, SSE has been considered. This technology is implemented in modern Intel processors. The third area is the instrumental programming tools. The examples in the third area are based on the standard library of the C programming language, YASM assembler, GCC compiler and GDB debugger.

Keywords: *floating-point arithmetic, IEEE754, numerical methods, the standard C programming language library, YASM assembler, SSE*

For citation:

Shundeev A. S. The Programming's Basics for Numerical Methods, *Programmnaya Ingeneria*, 2016, vol. 7, no. 5, pp. 221—230

DOI: 10.17587/prin.7.221-230

References

1. **IEEE Std** 754—2008. IEEE Standard for Floating-Point Arithmetic, IEEE, 29 August 2008, 58 p.

2. **Shundeev A. S.** Vvedenie v standart IEEE 754 (Introduction to the IEEE 754), *Programmnaya Ingeneria*, 2013, no. 3, pp. 44—47 (in Russian).

3. **Seyfarth R.** *Introduction to 64 Bit Intel Assembly Language Programming for Linux*, CreateSpace Independent Publishing Platform, 2011, 252 p.

4. **Kornev A. A., Chizhonkov E. V.** *Uprazhnenija po chislennym metodam. Chast' II* (Exercises on numerical methods. Part II.), Moscow, Izd-vo CPI pri mehaniko-matematicheskom fakul'tete MGU im. M. V. Lomonosova, 2003, 200 p. (in Russian).

УДК 004.932.4 DOI: 10.17587/prin.7.231-237

С. З. Свердлов, канд. тех. наук, проф., e-mail: c3c@mail.ru,
Вологодский государственный университет

Алгоритм и программа для уменьшения цифровых изображений

Рассмотрены алгоритм и реализующая его программа, предназначенные для уменьшения растровых цифровых изображений. За счет использования колориметрически-корректных преобразований и эффективного метода масштабирования, имеющего естественные основания, обеспечивается высокое качество масштабирования: правильная передача тонов, корректная цветопередача, высокая детализация.

Ключевые слова: *обработка изображений, цифровая фотография, масштабирование, растровое изображение, программное обеспечение, колориметрия*

Введение

Средства масштабирования цифровых растровых изображений широко распространены. В частности, востребованным является уменьшение изображений перед их публикацией в сети Интернет. В статье представлены эффективный алгоритм масштабирования и реализующая его программа, которые обеспечивают высококачественное уменьшение изображений. При проведении исследований, результаты которых представлены в статье, приняты перечисленные далее подходы и предположения.

- Преобразования изображения выполняются в линейном цветовом пространстве. В противном случае при вычислении цвета пикселей будут нарушены фундаментальные законы физики и колориметрии, в частности, закон аддитивности Г. Грассмана, согласно которому цветовые координаты суммы излучений равны сумме координат его составляющих [1, 2]. В задаче уменьшения изображений это особенно актуально, поскольку цвет пикселей результирующего (уменьшенного) изображения формируется как смесь цветов исходных пикселей.
- Промежуточные вычисления выполняются в вещественных, а не в целых числах. Это обеспечивает отсутствие ошибок округления. В рассматриваемом случае это важно, в частности, потому что при линеаризации исходного гамма-корректированного изображения значения RGB-компонент уменьшаются, что при использовании целочисленного представления привело бы к существенным потерям.
- Использованный алгоритм масштабирования базируется на естественных основаниях и относится к категории алгоритмов, основанных на площадях. Идея этого алгоритма известна [3, 4]. Предлагаемая автором его эффективная реализация, использующая вещественное представление цветовых линеаризованных координат, позволяет получить результаты, которые зачастую дают лучшее качество масштабирования, чем широко используемые методы.
- После масштабирования предусматривается регулируемое усиление резкости и выравнивание контраста.

Линеаризация

Переход к линейным цветовым координатам выполняется по достаточно сложным в вычислительном отношении формулам. Например, для цветового пространства sRGB [5] они выглядят следующим образом:

$$C_{\text{лин}} = \begin{cases} \dfrac{C_{srgb}}{12,92}, & C_{srgb} \leqslant 0,04045 \\ \left(\dfrac{C_{srgb}}{1,055}\right)^{2,4}, & C_{srgb} > 0,04045. \end{cases} \quad (1)$$

Здесь C_{srgb} — исходные гамма-корректированные sRGB-значения цветовых координат R, G или B (формулы для преобразования R, G и B одинаковы); $C_{\text{лин}}$ — линеаризованные значения R, G или B. Предполагается, что цветовые координаты выражаются вещественными числами в диапазоне от 0,0 до 1,0.

Как видно, вычисления по формуле (1) связаны с возведением в дробную степень, что требует существенных ресурсозатрат. По этой причине преобразование табулируется. При использовании 8-битного представления изображения число возможных значений C_{srgb} равно 256, в случае 16-битного представления — не более 65536. Ни по затратам памяти, ни по быстродействию использование таблицы такого размера не создает трудностей реализации при ресурсах, которые доступны на современных компьютерах.

Одновременно с преобразованием линеаризации выполняется и переход к вещественному представлению данных. Кроме этого, при табулировании преобразования (1) применяются нормирующие множители, которые используются в алгоритме масштабирования.

Таким образом, преобразования, которые должны быть применены к каждому пикселю, выполняются в ходе обработки одним обращением к таблице, требующим минимальных затрат.

Масштабирование

На рис. 1 (см. вторую сторону обложки) показана схема, иллюстрирующая принцип работы алгоритма масштабирования.

Рассмотрим масштабирование, при котором сохраняются пропорции изображения. Для иллюстрации возьмем картинку квадратного формата, что не уменьшает общности рассмотрения. Пусть есть растровое изображение размером $N \times N$ пикселей. На рис. 1 $N = 7$. Его нужно уменьшить до размера $n \times n$, $n < N$. В примере $n = 4$. Будем считать, что пиксели изображения — цветные квадраты. Естественный способ получения изображения размером $n \times n$ из картинки размером $N \times N$ можно представить как фотосъемку или сканирование исходного изображения с помощью цветочувствительной матрицы размером $n \times n$, сенсоры которой (сенсели) также представляют собой квадраты. Цвет, зарегистрированный каждым сенселем, формируется как цвет суммы излучений тех частей исходных пикселей, которые покрываются данным сенселем. Цветовые координаты этой суммы вычисляются как сумма соответствующих цветовых координат пикселей, покрываемых данным сенселем, с весовыми коэффициентами, пропорциональными части площади пикселя, которая покрывается сенселем. На рис. 1, *б* (см. вторую сторону обложки) светлым контуром выделены те части пикселей, которые участвуют в формировании цвета одного из сенселей.

Реализация масштабирования. Рассмотренный метод масштабирования допускает эффективную реализацию. Алгоритм обладает свойством сепарабельности. Его суть в том, что обработка матрицы изображения может быть сведена к масштабированию сначала по строкам, а затем — по столбцам (или наоборот). Решение двумерной задачи таким образом сводится к решению двух одномерных. В связи с этим обстоятельством в дальнейшем будем рассматривать одномерный случай. На рис. 2 (см. вторую сторону обложки) показана строка из $N = 7$ пикселей, которая должна быть масштабирована до размера $n = 4$ пикселя (сенселя).

Разобьем исходную строку ($N = 7$) и строку, которую нужно получить ($n = 4$), на $N \times n$ частей. В примере это $7 \times 4 = 28$ частей. Тогда и в каждый пиксель, и в каждый сенсель укладывается целое число таких элементарных частей — в пиксель — n частей, а в сенсель — N (рис. 2). Масштабирование можно представить как распределение элементарных частей пикселей по сенселям. При этом RGB-координаты сенселей вычисляются как сумма RGB-координат элементарных частей, а RGB-координаты каждой элементарной части берутся равными $1/n$-й RGB-координат соответствующего пикселя. Поскольку, в отличие от показанной на рис. 2 схемы, действительный размер пикселя итогового изображения (сенселя) такой же, что и пикселя исходного изображения, получаемая сумма должна дополнительно масштабироваться множителем n/N (для одномерного случая).

Пусть значение G-координаты (зеленая компонента цвета) первого слева на рис. 2, *а* (см. сторую сторону обложки), пикселя исходной строки равно 28, а второго слева пикселя — 84 (имеются в виду линейные координаты из диапазона 0...255). На каждую элементарную часть пикселя приходится 1/4 часть значения координаты. Таким образом, элементарная часть первого пикселя несет $28/4 = 7$ "единиц зеленого", а второго — $84/4 = 21$ "единицу зеленого". Цвет первого слева сенселя (рис. 2, *а*) образуется из четырех элементарных частей первого пикселя и трех частей второго. Значение G для первого сенселя вычисляется так: $G = (7 \times 4 + 3 \times 21)4/7 = 52$. Нетрудно заметить, что вычисление можно сократить и воспользоваться формулой

$$C = \frac{1}{N}\sum_{i=1}^{k} C_i m_i, \qquad (2)$$

где C — вычисленная цветовая координата сенселя; k — число пикселей, влияющих на цвет сенселя; C_i — цветовая координата i-го пикселя; m_i — число элементарных частей i-го пикселя, формирующих цвет сенселя.

Программную реализацию масштабирования рассмотрим для одномерного случая и монохромного изображения. Обработка трех цветовых каналов (R, G и B) выполняется аналогично. Обработка каждой строки матрицы изображения выполняется одинаково, а обработка столбцов — аналогично обработке строк.

Каждый пиксель исходной строки может участвовать в формировании цвета одного или двух пикселей результирующей строки (сенселей). В первом случае он целиком "поглощается" одним сенселем, во втором — его значение делится между двумя соседними.

Обозначим `x[N]` — массив цветовых координат строки исходного (монохромного) изображения; `y[n]` — массив результирующей строки; `i` — индекс элемента исходной строки; `k` — индекс элемента результирующей строки. Перед формированием результирующей строки запишем в ее элементы нулевые значения:

```
for(int k = 0; k<n; k + +)
    y[k] = 0;
```

Рассмотрим каждый пиксель исходной строки. Он состоит из n элементарных частей, которые нужно

распределить по пикселям результирующей строки (сенселям). Обозначим d — разность между количеством "свободного места" в очередном, k-м сенселе, и значением *n* — числом элементарных частей, которые надо разместить. Первоначально k = 0; d = N – n.

Можно выделить три случая.

1. Разность d>0. Свободного места более чем достаточно. К значению сенселя добавляется *n* элементарных частей (цветовая координата i-го пикселя умножается на весовой коэффициент, равный *n*). Разность d уменьшается на *n*.

2. Свободного места ровно *n* элементарных ячеек, d = 0. К значению сенселя добавляется *n* частей, сенсель заполнен, готовится заполнение следущего сенселя (k увеличивается на единицу). Значение d становится таким же, как в самом начале.

3. Количество свободного места недостаточно для размещения *n* частей: d < 0. В этом случае *n* частей разделяются между текущим и следующим сенселями.

Получаем следующую программу:

```
d = N - n;
k = 0;
for(i = 0; i<N; i ++)
   if(d > 0) {
     y[k] + = x[i]*n;
     d -= n;
}
else if(d == 0) {
   y[k ++] += x[i]*n;
   d = N - n;
}
else {
   y[k] += (n + d)*x[i];
   y[++ k] -= d*x[i];
   d += N - n;
};
```

Эта программа вычисляет для каждого сенселя сумму, присутствующую в правой части формулы (2). Однако для получения значений цветовых координат сенселей необходимо еще в соответствии с формулой (2) применить множитель 1/*N*. Отдельных ресурсозатрат для этого не требуется, поскольку такую операцию можно совместить с линеаризацией исходных данных, которая выполняется по таблице, вычисленной заранее. Затраты при этом переносятся с момента обработки изображения на момент предварительного формирования таблицы, которая обеспечивает не только линеаризацию, но также учет множителя и переход от целочисленного представления цветовых координат к вещественному.

Свойства алгоритма масштабирования. Как видно из рассмотрения принципа масштабирования (рис. 1, см. вторую сторону обложки), в формировании результирующего изображения всегда участвуют все пиксели исходного изображения. Это гарантирует сохранение детализации на итоговой картинке в той мере, насколько это возможно при уменьшении размера. При билинейной интерполяции используются 4 пикселя исходного изображения для вычисления цвета одного результирующего, при бикубической — 16 пикселей. При уменьшении более чем вдвое, в случае билинейной интерполяции, и более чем вчетверо, в случае интерполяции бикубической, неизбежна потеря детализации. Однако и при меньшей кратности уменьшения рассматриваемый алгоритм может использовать больше информации, чем билинейная и бикубическая интерполяции. Так, при кратности уменьшения от 1 до 2 число пикселей исходного изображения, участвующих в формировании цвета одного результирующего пикселя, может быть равным 9, при масштабе от 2 до 3 — 16, при масштабе от 3 до 4 — 25.

При кратности уменьшения, не превышающей 2, алгоритм работает примерно с той же скоростью, что и билинейный. При большей кратности билинейный алгоритм может работать быстрей, но при этом теряет данные.

К недостаткам представленного алгоритма масштабирования можно отнести некоторое снижение визуальной резкости итогового изображения.

Усиление резкости

После масштабирования выполняется регулируемое усиление резкости. Поскольку основным назначением уменьшенного изображения считается его использование для просмотра на экране в масштабе 1:1, оказывается возможным использование простого и эффективного алгоритма. Использование линейного цветового пространства для преобразования позволяет избежать артефактов, характерных для некоторых алгоритмов усиления резкости. Расчет для строки (для каждого из трех RGB-каналов) выполняется по формуле

$$y_i = \alpha x_i + (1-\alpha)\frac{x_{i-1} + x_i + x_{i+1}}{3};\ i \in [1, n-2], \qquad (3)$$

где α — сила эффекта; α > 1 соответствует усилению резкости; α < 1 — ослаблению; x_{i-1}, x_i, x_{i+1} —значения (цветовые координаты) трех очередных пикселей исходной (до усиления резкости) строки; y_i — значение *i*-го пикселя после преобразования. Обработка по столбцам выполняется аналогично.

Выравнивание контраста

В использованном алгоритме масштабирования вычисление цвета пикселей результирующего изображения выполняется усреднением (с весами) цвета нескольких пикселей исходного. При этом максимальное значение цветовых координат пикселей итогового изображения окажется не больше максимального значения цветовых координат исходного изображения. Если оно будет строго меньше, то этот факт будет означать уменьшение максимальной яркости масштабированного изображения. Аналогично обстоит дело и с минимальными значениями. Таким образом, в результате масштабирования возможно уменьшение общего контраста изображения.

В связи с отмеченными выше обстоятельствами предусмотрено выравнивание контраста. Для этого в ходе обработки определяются минимальные и максимальные значения цветовых координат пикселей исходного изображения, а затем цветовые координаты уменьшенного изображения масштабируются так, чтобы минимальные и максимальные значения цветовых координат уменьшенного и исходного изображений совпадали.

Программа C3C Image Size

Рассмотренный алгоритм уменьшения изображений реализован в программе C3C Image Size (рис. 3) [6]. Программа представляет собой подключаемый модуль (плагин) для Adobe Photoshop. Существуют версии для ОС Windows и Mac OS (адаптация для Mac OS выполнена А. Данильченко).

Параметр Sharpness (резкость) задается по логарифмической шкале. Значение Sharpness, равное 0, соответствует отсутствию усиления резкости ($\alpha = 1$), значение Sharpness, равное 1, соответствует $\alpha = \sqrt{2}$.

Тестирование

Предъявление примеров масштабирования растровых изображений в печатном документе представляет определенные сложности. То, что мы видим на бумаге — результат работы программ и оборудования, используемых при печати. Эти программы в том числе выполняют и масштабирование. В силу этих обстоятельств приведенные иллюстрации могут дать лишь приблизительное представление о работе обсуждаемых алгоритмов. Для корректного сопоставления нужно рассматривать оригиналы изображений, как исходных, так и уменьшенных. Эти оригиналы доступны на сайте программы C3C Image Size [6].

Черно-белая мишень. Первый пример — масштабирование черно-белой мишени. Исходное изображение (рис. 4) имеет разрешение 1200×1200 пикселей. Оно масштабировалось до размера 200×200 программами C3C Image Size, Adobe Photoshop CC (выпуск 2015, алгоритм bicubic) и плагином Resize [7] к программе ImageJ 1.49v [8] (алгоритм Least-Squares).

Результаты масштабирования показаны на рис. 5 и 6. Чтобы улучшить качество воспроизведения масштабированных изображений в печатном документе, после масштабирования выполнено увеличение вдвое с использованием алгоритма ближайшего соседа, который в данном случае "учетверяет" каждый пиксель.

Обращает на себя внимание, что полученные ImageJ и Adobe Photoshop результаты дают затемнение по периферии мишени. По эффекту муара варианты масштабирования программами Adobe Photoshop и

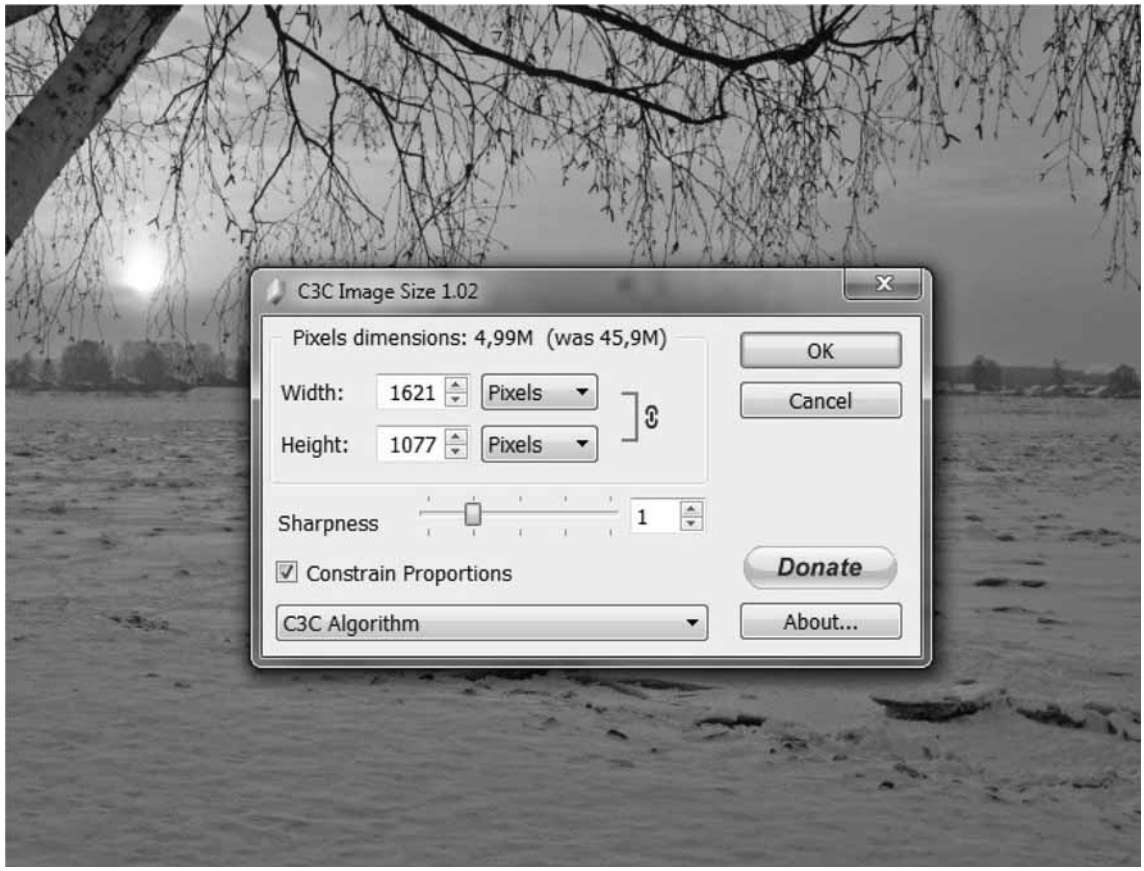


Рис. 3. Программа C3C Image Size. Версия для Windows

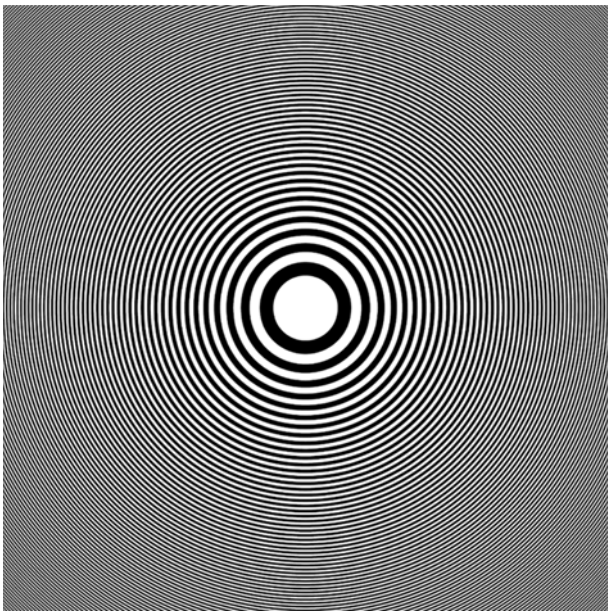


Рис. 4. Черно-белая мишень

C3C Image Size выглядят предпочтительней варианта масштабирования посредством ImageJ.

Фотография. В качестве примера для масштабирования взят снимок "Русский Север" (рис. 7, см. вторую сторону обложки), сделанный в Великом Устюге в январе 2016 г. Исходный размер фотографии 4912×3264 пикселя — оригинальный размер, получаемый с фотокамеры Sony NEX-6.

Фотография масштабировалась до "веб-размера" — 900 пикселей по длинной стороне (линейное уменьшение в 5,46 раза). Затем, чтобы можно было сравнивать результаты масштабирования с исходным изображением, а также для того, чтобы эти результаты лучше воспроизводились в печатном документе, уменьшенные изображения были снова приведены к исходному размеру с помощью алгоритма ближайшего соседа. Фрагменты этих изображений показаны на рис. 8—11 (см. третью сторону обложки).

Сравнивая детализацию разных вариантов масштабирования, можно обратить внимание на перечисленные далее объекты на исходном снимке (рис. 8).

- В нижней части слева видны два человека, переходящих по льду замерзшую реку Сухону. Сказать уверенно, что на каком-либо из вариантов при уменьшении пара (то, что их двое) осталась хорошо различима, вряд ли можно. Однако, по мнению автора, на рис. 9 что-то просматривается.
- На противоположном берегу стоит автомобиль с включенными фарами. Лучше всего наличие двух фар передано в варианте ImageJ (рис. 10). Две фары различимы в варианте C3C Image Size и совсем неразличимы в случае масштабирования с помощью Adobe Photoshop.
- На исходном фрагменте хорошо видны пять крестов на церквях Дмитрия Солунского и Сергия Радонежского. При масштабировании программой C3C Image Size остаются различимы два креста; программой ImageJ — один; программой Adobe Photoshop — ни одного.
- Можно также обратить внимание на передачу деталей в изображении большого дома, расположенного справа от церквей. Предпочтительным представляется вариант, полученный программой C3C Image Size.

Специальный тест. Приведены результаты испытания программ на тестовом изображении (рис. 12,

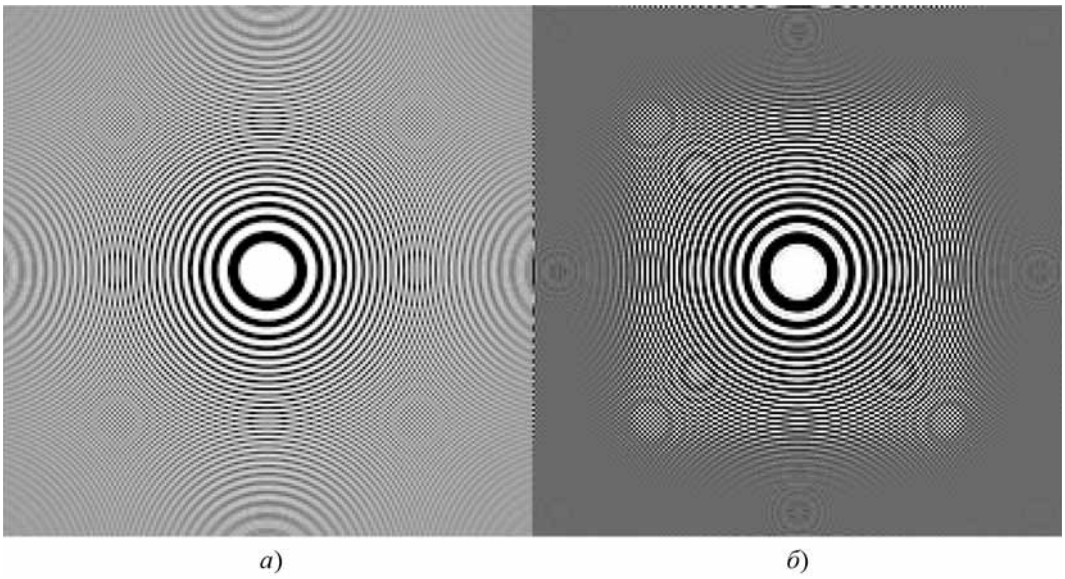


Рис. 5. Результаты масштабирования:
а — программой C3C Image Size; *б* — программой ImageJ

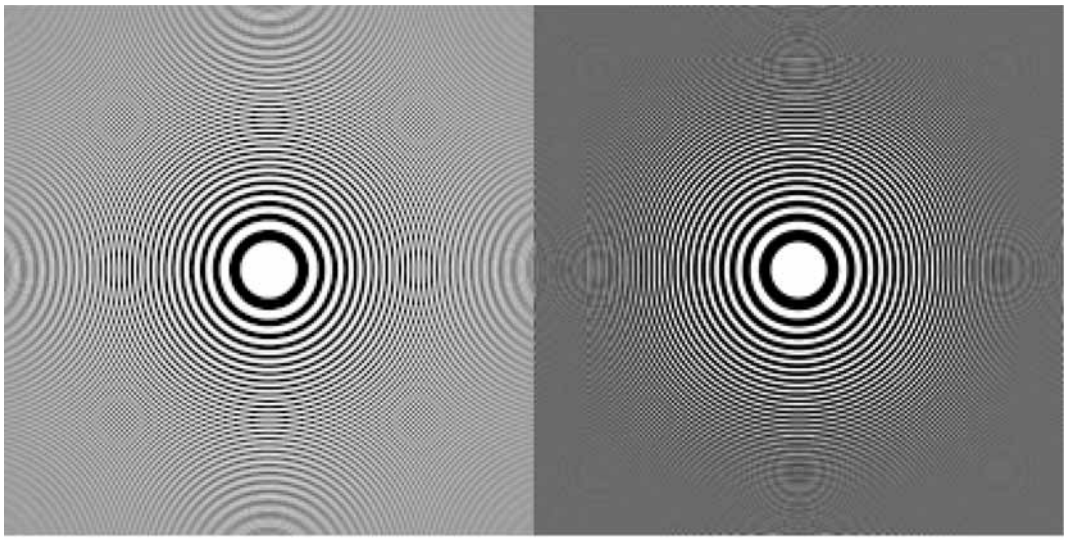


Рис. 6. Результаты масштабирования:
а — программой C3C Image Size; *б* — программой Adobe Photoshop

см. четвертую сторону обложки), специально созданном для того, чтобы продемонстрировать некорректное масштабирование программами, которые используют для преобразований нелинейное цветовое пространство.

На рис. 13 (см. четвертую сторону обложки) показан увеличенный фрагмент тестового изображения. В оригинале цветные квадраты имеют размер 4 × 4 пикселя, а общий размер изображения 512×512 пикселей.

Прежде чем обсуждать результаты тестирования, сформулируем критерий правильного преобразования: если преобразование, выполняемое с изображением, не имеет целью получение каких-либо специальных эффектов, то результат трансформации должен совпадать с тем, что видит человек в тех обстоятельствах, которые соответствуют такой трансформации. В нашем случае, когда речь идет об уменьшении изображения, результирующее изображение должно выглядеть также, как исходное изображение, которое наблюдатель рассматривает с большего расстояния. Например, уменьшенная вдвое картинка должна выглядеть при обычном расстоянии просмотра также, как исходная с расстояния, превышающего обычное расстояние просмотра в 2 раза.

Если рассматривать изображение на рис. 12 издалека, мы всегда, пока видимый размер изображения не достигнет предела разрешающей способности глаза, видим звезду бледно-красного цвета на сером фоне. Соответственно, при масштабировании этого изображения на уменьшенном варианте должна быть видна такая же звезда. Но это оказывается не всегда так.

При тестировании исходное изображение уменьшалось в 8 раз до размера 64×64 пикселя. Затем, для того чтобы результат можно было представить в печатной публикации, результирующие изображения были увеличены в 8 раз с помощью алгоритма ближайшего соседа. Результат показан на рис. 14, см. четвертую сторону обложку (масштабы на рис. 12 и рис. 14 различны).

Полученный результат, разумеется, обусловлен тем, что тестовое изображение специально построено таким образом, чтобы некорректное поведение программ, использующих нелинейное цветовое пространство для преобразований, было очевидно. На результат масштабирования реальных фотографий подобное некорректное поведение тоже влияет, но это влияние не так бросается в глаза.

Заключение

Представленный алгоритм и реализующая его программа C3C Image Size обеспечивают высококачественное уменьшение растровых изображений. Программа также предоставляет возможность регулировки резкости. Это помогает компенсировать возможный недостаток резкости уменьшенного изображения. Использование линейного цветового пространства гарантирует корректное вычисление цветов уменьшенного изображения. Тесты демонстрируют правильную тонопередачу без затемнения или осветления участков изображения, корректную цветопередачу, высокую детализацию уменьшенного изображения. Использованный алгоритм допускает эффективную реализацию.

Программа C3C Image Size [6] получила распространение среди фотографов и доступна для свободного использования.

Список литературы

1. **Wyszecki G., Stiles W. S.** Color Science. Concepts and Methods, Quantitative Data and Formulae. Second Edition, Wiley-Interscience Publication, 2000.
2. **Юстова Е. Н.** Цветовые измерения (Колориметрия). Спб.: Из-во С.-Петер. ун-та, 2000. 397 с.
3. **Chun-ho Kim, Si-mun Seong, Jin-aeon Lee, Lee-sup Kim** Winscale: An Image-Scaling Algorithm Using an Area Pixel Model// IEEE Transactions on Circuits and Systems for Video Technology. 2003. Vol. 13, N. 6. P. 549—553.
4. **Summers D.** Pixel Mixing. URL: http://entropymine.com/imageworsener/pixelmixing
5. **International** Standard IEC 61966-2-1 ed1.0. Multimedia systems and equipment — Colour measurement and management — Part 2-1: Colour management — Default RGB colour space — sRGB.
6. **Свердлов С.** C3C Image Size. Высококачественное уменьшение изображений. URL: http://www.uni-vologda.ac.ru/~c3c/plug-ins/c3cimagesize.htm
7. **An ImageJ** plugin to resize an image using high-quality interpolation. URL: http://bigwww.epfl.ch/algorithms/ijplugins/resize/
8. **ImageJ.** Image Processing and Analysis in Java. URL: http://imagej.nih.gov/ij/

Algorithm and Program to Downsizing the Digital Images

S. Z. Sverdlov, c3c@uni-vologda.ac.ru, Vologda State University, Vologda, 160600, Russian Federation

Corresponding author:

Sverdlov Sergey Z., Professor, Vologda State University, Vologda, 160600, Russian Federation, e-mail: c3c@uni-vologda.ac.ru

Received on February 05,2016
Accepted on March 02, 2016

The scaling algorithm, which provides high-quality digital images downsizing, and C3C Image Size program, that implements it, are described. They are developed based on the following principles:

1. Images are transformed in a linear color space. Otherwise, the calculation of the pixels color would violate fundamental laws of physics and colorimetry, in particular, H. Grassmann's law of additivity: the color coordinates of the sum of radiations are equal to the sum of the components' coordinates. It's especially relevant for the images size reduction, because the color of pixels of the resulting (downsized) image is formed as a mixture of color of source pixels.

2. Intermediate calculations are performed in real numbers rather than integers, that ensures the absence of rounding errors. As RGB component values are reduced during the linearization of original gamma-corrected image, using integer representation would lead to substantial loss of accuracy, that's why it's especially important to avoid rounding errors.

3. Scaling is performed using an algorithm, which has a natural physical interpretation.

4. Controlled sharpness gain and contrast leveling are applied after scaling.

Tests show proper transmission of tones without darkened or lightened areas on the picture, correct color rendition, high quality of the reduced image. The algorithm allows effective implementation, that provides high operating speed of the program. C3C Image Size program has spread among photographers and is available for free use.

Keywords: *Image processing, digital photo, resize, raster graphics, software, plug-in, colorimetry*

For citation:

Sverdlov S. Z. Algorithm and Program to Downsizing the Digital Images, *Programmnaya Ingeneria*, 2016, vol. 7, no. 5, pp. 231—237.

DOI: 10.17587/prin.7.231-237

References

1. **Wyszecki G., Stiles W. S.** *Color Science. Concepts and Methods, Quantitative Data and Formulae*. Second Edition, Wiley-Interscience Publication, 2000.
2. **Justova E. N.** *Cvetovye izmerenija (Kolorimetrija)* (The color measurement (Colorimetry)), Saint Petersburg, S-PbU Publishing, 2000. 397 p. (in Russian).
3. **Chun-ho Kim, Si-mun Seong, Jin-aeon Lee, Lee-sup Kim** Winscale: An Image-Scaling Algorithm Using an Area Pixel Model// IEEE Transactions on Circuits and Systems for Video Technology. 2003. Vol. 13, no. 6, pp. 549—553.
4. **Summers D.** Pixel Mixing. URL: http://entropymine.com/imageworsener/pixelmixing
5. **International** Standard IEC 61966-2-1 ed1.0. Multimedia systems and equipment — Colour measurement and management — Part 2-1: Colour management — Default RGB colour space — sRGB.
6. **Sverdlov S.** C3C Image Size, available at: http://www.uni-vologda.ac.ru/~c3c/plug-ins/c3cimagesize.htm
7. **An ImageJ.** Plugin to resize an image using high-quality interpolation, available at: http://bigwww.epfl.ch/algorithms/ijplugins/resize/
8. **ImageJ.** Image Processing and Analysis in Java, available at: http://imagej.nih.gov/ij/

VII Всероссийская конференция "Проблемы разработки перспективных микро- и наноэлектронных систем" (серия "МЭС")
пройдет 03–07 октября 2016 г.
в Зеленограде

МЭС-2016

Среди учредителей конференции – Российская академия наук, Российский фонд фундаментальных исследований, ФАНО России, Национальный исследовательский университет "МИЭТ", Южный федеральный университет, Префектура Зеленоградского АО г. Москвы, ОАО "НИИМЭ и Микрон". В число официальных партнеров и спонсоров конференции входят Фонд образовательных и инфраструктурных программ Роснано, Фонд "Сколково", фирма Intel, ПАО "ИНЭУМ им. И. С. Брука", ЗАО "ПКК Миландр", ОАО НПЦ "ЭЛВИС", ЗАО "Мегратек" и многие другие предприятия. Организатор конференции Федеральное государственное бюджетное учреждение науки Институт проблем проектирования в микроэлектронике Российской академии наук (ИППМ РАН), соорганизаторы конференции Казенное предприятие г. Москвы "Корпорация развития Зеленограда" и Московское научно-техническое общество радиотехники, электроники и связи имени А. С. Попова.

Рейтинг конференций серии "МЭС" неуклонно растет. Увеличивается число участников (в 2014 г. — более 230), к организации конференции привлекаются новые предприятия. Труды конференции, в том числе англоязычный реферативный сборник статей, включены в Перечень рецензируемых научных изданий, в которых должны быть опубликованы основные научные результаты диссертаций на соискание ученой степени кандидата наук, на соискание ученой степени доктора наук (Перечень ВАК). В текущем году планируется включение Трудов конференции в раздел Russian Science Citation Index (RSCI) базы научных публикаций Web of Science.

Как и прежде, конференция "МЭС" этого года посвящена актуальным вопросам автоматизации проектирования микроэлектронных схем, систем на кристалле, IP-блоков и новой элементной базы по следующим научным направлениям:

- Теоретические аспекты проектирования микро- и наноэлектронных систем (МЭС);
- Методы и средства автоматизации проектирования микро- и наноэлектронных схем и систем (САПР СБИС);
- Опыт разработки цифровых, аналоговых, цифроаналоговых, радиотехнических функциональных блоков СБИС;
- Особенности проектирования СБИС для нанометровых технологий;
- Системы на кристалле перспективной РЭА.

Также особое внимание будет уделено вопросам разработки интегральных схем для космического применения.

В рамках конференции пройдет сессия-презентация научно-технических достижений российских и зарубежных компаний, а также организаций, способствующих развитию микроэлектроники и информационных технологий в России. Запланировано проведение выставки программных продуктов и оборудования, используемого в микроэлектронной отрасли.

Более подробную информацию о конференции можно получить на web-сайте
http://www.mes-conference.ru

Уважаемые коллеги!

Приглашаем Вас принять авторское участие в журнале "Программная инженерия".

К опубликованию принимаются статьи, содержание которых соответствует тематике журнала и включает новые результаты исследований, материалы обзорного и методического характера, не опубликованные ранее и не предназначенные к публикации в других печатных или электронных изданиях.

Все статьи проходят обязательное рецензирование в редакции.

Для опубликования статьи в редакцию журнала направляются следующие материалы:

- рукопись статьи в DOC- и PDF-форматах;
- таблицы, иллюстрации и перечень подрисуночных подписей;
- сведения об авторах, содержащие (согласно регламенту РИНЦ) фамилию, имя, отчество, ученые степень и звание, должность, место работы, служебный и домашний адреса, телефоны и E-mail;
- экспертное заключение о возможности публикации статьи в открытой печати;
- англоязычная информация, включающая краткие сведения об авторах, о содержании статьи (расширенная аннотация) и список литературы, которые необходимы для индексирования журнала в международных наукометрических базах данных.

Объем рукописи статьи, предлагаемой к публикации, должен быть не менее 10 и не более 25 страниц машинописного текста, напечатанного на одной стороне белого листа бумаги формата А4 с полями со всех сторон не менее 2 см, с абзацным отступом 1 см, с полуторным межстрочным интервалом, с использованием текстового редактора Microsoft Word (любая версия) с шрифтом Times New Roman размером 14 pt. В указанный объем статьи входят: текст, приложения, иллюстрации, таблицы, список литературы. В отдельных случаях по решению редколлегии объем статьи может быть увеличен. Страницы рукописи должны быть пронумерованы, начиная с первой.

Более подробную информацию о правилах оформления материала см. на сайте http://novtex.ru/prin/rus/for_authors.html

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

Сдано в набор 10.03.2016 г. Подписано в печать 25.04.2016 г. Формат 60×88 1/8. Заказ PI516
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**