ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

# Программная инженерия

ПрИн 9 2013

Учредитель: Издательство "НОВЫЕ ТЕХНОЛОГИИ"

Издается с сентября 2010 г.



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

## 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 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.

УДК 004.432.2

**Н. И. Вьюкова**, стар. науч. сотр., **В. А. Галатенко**, д-р физ.-мат. наук, стар. науч. сотр.,
**С. В. Самборский**, стар. науч. сотр.,
НИИ системных исследований РАН, г. Москва,
e-mail: niva@niisi.msk.ru

# Поддержка многопоточности в стандарте C11

*Представлены базовые средства поддержки многопоточных программ, вошедшие в стандарт языка C ISO/IEC 9899:2011, который был принят в декабре 2011 г. Рассматривается библиотека управления потоками, атомарные типы и простые атомарные операции, а также некоторые аспекты генерации кода для многопоточных программ.*

***Ключевые слова:*** *язык программирования C, C11, многопоточность, атомарные типы данных, последовательная консистентность*

## Введение

В декабре 2011 г. вышел новый стандарт языка C [1], получивший неформальное название C11. В статье [2] представлены почти все основные нововведения, вошедшие в C11. В настоящей статье, которая является продолжением работы [2], рассмотрены средства поддержки многопоточности — библиотека управления потоками, атомарные типы данных и базовые атомарные операции, а также специфика генерации кода для многопоточных программ. Еще одну статью авторы планируют посвятить рассмотрению прочих атомарных операций, а также модели памяти многопоточных программ.

Несмотря на то что в прежних стандартах языка C отсутствовали такие понятия, как поток и многопоточная программа, существует давняя практика написания многопоточных программ на C. Обычно для этого используют какую-либо библиотеку, предоставляющую средства работы с потоками и средства синхронизации, например, библиотеку Pthreads, реализующую потоки в соответствии со стандартом POSIX [3]. Со временем в программистском сообществе сложилось понимание того факта, что подобный подход некорректен. Наиболее полно эта точка зрения выражена в статье Ханса Боэма [4], вышедшей в 2004 г. Автор рассматривает спецификации Pthreads, регламентирующие упорядочение операций над памятью, и указывает на их явную недостаточность для разработчиков как прикладных программ, так и компиляторов. В результате отлаженная и работающая программа может оказаться неработоспособной при переходе на другую версию компилятора или аппаратуры.

В статье [4] представлены примеры методов оптимизации и генерации кода, которые могут приводить к проблемным вопросам при выполнении многопоточных программ. В частности, рассматривается вопрос неатомарности доступа к данным. Например, для считывания или записи 64-битного значения на 32-битном процессоре требуется выполнить две команды. Поэтому, если два потока одновременно модифицируют объект, то в память может быть записано некорректное значение, состоящее из "половинок" разных значений. Или, если один поток модифицирует объект, а другой читает его, может быть считано частично модифицированное значение.

В статье Боэма рассмотрены также вопросы эффективности программ, использующих традиционные средства синхронизации (блокировки). Отмечается, что во многих ситуациях более эффективными оказываются альтернативные подходы, основанные на неблокирующих (*lock-free*) алгоритмах и алгоритмах без ожидания (*wait-free*).

Основываясь на представленных в статье [4] соображениях, инициативная группа специалистов во главе с Боэмом выработала спецификации средств поддержки многопоточности и соответствующей модели памяти, которые первоначально были включены в очередной стандарт языка C++ [5]. За основу была взята модель памяти языка Java, адаптированная с учетом особенностей языка C++. В дальнейшем эти спецификации были перенесены с соответствующими изменениями в стандарт C11 языка C.

Заметим, что механизм транзакций памяти [6] не вошел в стандарты C++ и C. Объясняется это, видимо, тем обстоятельством, что на настоящее время отсутствует опыт промышленного применения этого средства.

## 1. Потоки выполнения

Стандарт C11 (см. [1], п. 5.1.2.4) явно вводит понятие многопоточной программы и определяет выполнение программы как выполнение всех ее потоков. Вводится также понятие разделяемой памяти. Утверждается, что значение объекта, видимое для потока *T* в некоторой точке программы, это либо начальное значение объекта, либо значение, записанное потоком *T* или другим потоком той же программы.

Стандарт вводит новый класс хранения **_Thread_local**, который используется для описания локальных объектов данных потока. Переменная с классом хранения **_Thread_local** не является разделяемой, для каждого потока создается отдельный экземпляр такой переменной.

Ниже будут описаны два основных инструментальных средства разработки многопоточных приложений — библиотека управления потоками **<threads.h>** и библиотека для работы с атомарными объектами **<stdatomic.h>**. Обе эти библиотеки относятся к числу необязательных возможностей.

## 2. Управление потоками и средства синхронизации. Библиотека <threads.h>

Библиотека управления потоками **<threads.h>** позволяет оперировать с объектами следующих типов:

- потоки (тип **thrd_t**);
- мьютексы (тип **mtx_t**);
- условные переменные (тип **cnd_t**);
- локальные данные потоков (тип **tss_t**);
- объекты, предназначенные для однократного выполнения инициализирующих действий (тип **once_flag**).

В ней также описаны вспомогательные типы, такие как типы стартовых функций потоков (**thrd_start_t**) и деструкторов (**tss_dtor_t**) для локальных данных потоков. Описание этих понятий, а также методы и примеры их использования можно найти в учебном курсе [7]. В целом набор функций этой библиотеки можно оценить как обобщение функциональных возможностей, наиболее распространенных в практике многопоточного программирования библиотек потоков. Ниже перечислены функции библиотеки **<threads.h>**.

### Функции управления потоками

Создание потока:

```
int thrd_create(thrd_t *thr, thrd_start_t
    func, void *arg);
```

Завершение потока:

```
_Noreturn void thrd_exit(int res);
```

Получение идентификатора потока:

```
thrd_t thrd_current(void);
```

Сравнение двух идентификаторов потоков на равенство:

```
int thrd_equal(thrd_t thr0, thrd_t thr1);
```

Обособление потока:

```
int thrd_detach(thrd_t thr);
```

Присоединение потока:

```
int thrd_join(thrd_t thr, int *res);
```

Приостановка потока:

```
int thrd_sleep(const struct timespec *duration,
    struct timespec *remaining);
```

Передача процессора другому потоку:

```
void thrd_yield(void);
```

По сравнению с библиотеками потоков стандарта POSIX, в **<thread.h>** отсутствует понятие атрибутов потока. В частности, не предоставляются средства для управления параметрами планирования потоков.

### Функции управления мьютексами

Инициализация мьютекса (значения **type**: **mtx_plain** — обычный мьютекс, **mtx_timed** — мьютекс с ограниченным временем ожидания, **mtx_recursive** — мьютекс, допускающий рекурсивные блокировки):

```
int mtx_init(mtx_t *mtx, int type);
```

Высвобождение ресурсов, занимаемых мьютексом:

```
void mtx_destroy(mtx_t *mtx);
```

Захват мьютекса с неограниченным ожиданием:

```
int mtx_lock(mtx_t *mtx);
```

Захват мьютекса с ограниченным ожиданием:

```
int mtx_timedlock(mtx_t *restrict mtx,
    const struct timespec *restrict ts);
```

Захват мьютекса без ожидания:

```
int mtx_trylock(mtx_t *mtx);
```

### Управление условными переменными

Создание переменной:

```
int cnd_init(cnd_t *cond);
```

Уничтожение переменной:

```
void cnd_destroy(cnd_t *cond);
```

Разблокирование одного ждущего потока:

```
int cnd_signal(cnd_t *cond);
```

Разблокирование всех ждущих потоков:

```
int cnd_broadcast(cnd_t *cond);
```

Ожидание условия:

```
int cnd_wait(cnd_t *cond, mtx_t *mtx);
```

Ожидание условия, ограниченное по времени:

```
int cnd_timedwait(cnd_t *restrict cond,
     mtx_t *restrict mtx,
      const struct timespec *restrict ts);
```

#### Функции управления локальными данными потоков

Создание ключа данных:

```
int tss_create(tss_t *key, tss_dtor_t dtor);
```

Освобождение ресурсов, занимаемых данными:

```
void tss_delete(tss_t key);
```

Установка значения данных:

```
int tss_set(tss_t key, void *val);
```

Получение данных:

```
void *tss_get(tss_t key);
```

#### Функция инициализации

Позволяет однократно вызвать заданную функцию; используется, например, для создания ключа локальных данных потока:

```
void call_once(once_flag *flag, void (*func)
   (void));
```

## 3. Атомарные типы данных и операции над ними. Библиотека <stdatomic.h>

В этом разделе представлены атомарные типы данных и основные атомарные операции. Введено понятие неблокируемости атомарных типов. В заключение приведены примеры, иллюстрирующие применение атомарных операций.

### 3.1. Атомарные типы данных

Для описания атомарных типов данных стандарт C11 вводит новый спецификатор **_Atomic**. Он допускается в декларациях любых объектов и типов данных, за исключением функций и массивов. Примеры:

```
_Atomic int a; // atomic int

int* _Atomic b; // atomic pointer

_Atomic int* c; // pointer to atomic int

_Atomic int* _Atomic c; // atomic pointer to
   //atomic int

_Atomic struct { int d; } e; // atomic struct
```

Представления атомарного типа и соответствующего неатомарного могут различаться, в том числе по таким характеристикам, как размер и выравнивание. Это различие объясняется тем обстоятельством, что представление атомарного типа может, например, содержать мьютекс.

### 3.2. Предопределенные атомарные типы

В заголовочном файле **<stdatomic.h>** вводятся определения атомарных типов, соответствующих всем стандартным целочисленным типам, и типам, описанным в файле **<stdint.h>**. Имена атомарных типов образованы путем добавления префикса **atomic_** к имени соответствующего неатомарного типа, например: **atomic_bool**, **atomic_char**, **atomic_uchar**, **atomic_int**, **atomic_uint**, **atomic_size_t**, **atomic_intptr_t** и т. д.

Стандарт требует, чтобы эти предопределенные атомарные типы имели такое же выравнивание, что и соответствующие неатомарные, однако размер при этом может отличаться.

### 3.3. Инициализация атомарных объектов

В библиотеке **<stdatomic.h>** вводятся два средства инициализации атомарных объектов.

Макрос **ATOMIC_VAR_INIT** (*C value*) используется в декларациях атомарных объектов для их инициализации, например:

```
atomic_int guide = ATOMIC_VAR_INIT(42);
```

Функция **void atomic_init(volatile** *A* ***obj,** *C* **value)**; инициализирует атомарный объект по указателю **obj** значением **value**. Здесь *A* — атомарный тип, *C* — соответствующий неатомарный тип.

### 3.4. Атомарные операции

В этом подразд. описаны содержащиеся в библиотеке **<stdatomic.h>** операции над объектами атомарных типов, обеспечивающие функционирование многопоточных программ в режиме *последовательной консистентности*. Выполнение многопоточной программы в данном режиме может рассматриваться как перемежающееся последовательное выполнение вычислений из составляющих ее потоков. Стандарт поддерживает и другие режимы синхронизации, которые будут рассмотрены в статье, посвященной модели памяти многопоточных программ.

Описываемые далее функции являются обобщенными (перегруженными), они могут применяться к объектам любых стандартных атомарных типов. Используя обобщающие селекторы **_Generic** [2], пользователь может описывать также свои обобщенные функции для работы с данными разных атомарных типов.

В представленных далее описаниях использованы следующие обозначения типов:

*A* — атомарный тип;

*C* — соответствующий неатомарный тип;

*M* — тип данных для операндов арифметических операций.

Значение, на которое указывает указатель **object**, атомарно замещается значением **desired**:

**void atomic_store(volatile** *A* ***object,** *C* **desired);**

Функция атомарно считывает и возвращает значение, на которое указывает указатель **object:**

*C* **atomic_load(volatile** *A* ***object);**

Функция атомарно замещает значение ***object** на **desired**. Возвращает атомарно считанное прежнее значение по указателю **object**:

```
C atomic_exchange(volatile A *object,C desired);
```

Следующая функция атомарно выполняет такие действия: сравнивает на равенство ***object** и ***expected**, в случае равенства замещает значение по указателю **object** на **desired**, в противном случае замещает значение по указателю **expected** на ***object**; возвращает результат сравнения:

```
_Bool atomic_compare_exchange_strong
  (volatile A *object, C *expected,C desired);
```

Функция

```
_Bool atomic_compare_exchange_weak
  (volatile A *object, C *expected,C desired);
```

отличается от предыдущей функции тем, что может давать ложные отказы. Этот вариант позволяет обеспечить эффективность выполнения на RISC-процессорах, где для реализации атомарных операций вида RMW (*Read-Modify-Write* — чтение-модификация-запись) используют команды LL (*Load Linked*) и SC (*Store Conditional*). Команда LL считывает значение из памяти и инициализирует выполнение RMW-операции. Команда SC завершает RMW-операцию, выполняя запись входного регистра по указанному адресу при условии, что по нему не было записи другим потоком с момента выполнения LL. Команда SC записывает в выходной регистр 1 при успешной записи, иначе 0.

Особенность реализации SC такова, что она может давать ложный отказ, например, если с момента выполнения LL другой процессор обратился к слову памяти в той же кэш-линии или если произошло прерывание по переключению контекста. Ложные отказы могут происходить и по другим причинам, зависящим от реализации. Поэтому для реализации "сильного" варианта необходимо выполнять команды LL—SC в цикле. В результате, если пользовательская программа сама использует **atomic_compare_exchange** в цикле (пока не произойдет перезапись объекта), то будет сгенерирован неэффективный код в виде гнезда из пары вложенных циклов. В таком случае рекомендуется использовать "слабый" вариант функции.

Ниже приведен пример использования **atomic_compare_exchange_weak:**

```
int atomic_fetch_mul(atomic_int *val, int
multiplier)
{
   int old_val = atomic_load (val);
   int new_val;
   do {
     new_val = old_val * multiplier;
   } while (!atomic_compare_exchange_weak (
        val, &old_val, new_val));
   return old_val;
}
```

Представленные ниже функции замещают значение по указателю **object** результатом выполнения соответствующей операции над значениями ***object** и **operand** и возвращают атомарно считанное прежнее значение ***object**:

```
C atomic_fetch_key(volatile A *object,
  M operand);
```

где *key* может быть:

**add +** (сложение);
**sub -** (вычитание);
**or |** (побитовое включающее "или");
**xor ^** (побитовое исключающее "или");
**and &** (побитовое "и").

Следующие две функции применимы к объектам типа **atomic_flag**. Тип **atomic_flag** реализует классическую функцию "проверить-и-установить" (*test-and-set*). Объекты этого типа могут находиться в одном из двух состояний: **true** — занят (*set*), **false** — свободен (**clear**).

Первая из этих двух функций

```
_Bool atomic_flag_test_and_set(
   volatile atomic_flag *object);
```

атомарно устанавливает значение указуемого объекта равным **true** и возвращает его предшествующее значение.

Вторая функция

```
void atomic_flag_clear(volatile atomic_flag
   *object);
```

атомарно устанавливает значение указуемого объекта равным **false**.

Функции **atomic_exchange, atomic_compare_exchange_*, atomic_fetch_*, atomic_flag_test_and_set** являются функциями вида RMW.

### 3.5. Особенности выполнения других операций над объектами атомарных типов

Эти особенности касаются постфиксных операторов **++**, **--** и составных операторов присваивания, а также операций доступа (**.**, **->**) к элементам атомарных структур и объединений.

Постфиксные операторы **++**, **--** и операторы присваивания (такие как **+=** ) над объектами атомарных типов определяются стандартом C11 как атомарные операции вида RMW.

Согласно стандарту, непосредственный доступ к элементам атомарных структур или объединений рассматривается как неопределенное поведение. Для таких действий следует использовать объект соответствующего неатомарного типа и операции копирования.

### 3.6. Неблокирующие атомарные типы

Тип данных называется неблокирующим, если все определенные для него атомарные операции — неблокирующие. Реализация неблокирующих атомарных

операций основана на использовании аппаратных примитивов. Стандарт не требует, чтобы все атомарные операции для всех типов данных были неблокирующими. Единственный тип, для которого это требование обязательно — **atomic_flag**, т. е. операции **atomic_flag_test_and_set, atomic_flag_clear** должны быть реализованы аппаратными средствами.

Если целевая архитектура не поддерживает аппаратной реализации некоторой атомарной операции, то ее программная реализация может, например, использовать мьютекс, т. е. данная операция будет блокирующей. Некоторые алгоритмы могут работать менее эффективно, если реализации используемых в них атомарных операций блокирующие. Перечисленные далее макросы позволяют программе опрашивать свойство неблокируемости для целочисленных и указательных типов.

- **ATOMIC_BOOL_LOCK_FREE;**
- **ATOMIC_CHAR_LOCK_FREE;**
- **ATOMIC_CHAR16_T_LOCK_FREE;**
- **ATOMIC_CHAR32_T_LOCK_FREE;**
- **ATOMIC_WCHAR_T_LOCK_FREE;**
- **ATOMIC_SHORT_LOCK_FREE;**
- **ATOMIC_INT_LOCK_FREE;**
- **ATOMIC_LONG_LOCK_FREE;**
- **ATOMIC_LLONG_LOCK_FREE;**
- **ATOMIC_POINTER_LOCK_FREE.**

Перечисленные макросы могут принимать значения 0, 1 или 2. Значение 0 указывает, что операции над объектами соответствующего типа всегда блокирующие; значение 1 означает, что операции могут быть блокирующими или неблокирующими. Значение 2 соответствует неблокирующим типам. Каждый из перечисленных макросов задает свойства для соответствующего знакового и беззнакового типов.

### 3.7. Примеры

В этом подразделе приведены примеры, иллюстрирующие различия между использованием обычных и атомарных операций чтения и записи данных в многопоточной программе. Рассмотрим пример "наивной" реализации синхронизации между двумя потоками, один из которых записывает данные, а второй обрабатывает.

**Пример 1.** Многопоточная программа, использующая обычные операции доступа к объектам в памяти.

```
int data;
int flag = 0;
// Thread 1:
data = 123; // запись данных
flag = 1;
// Thread 2:
if (flag) // обработка данных
  printf ("%d", data);
```

Второй поток (**Thread 2**) предполагает, что, если флаг установлен (**flag==1**), то данные (переменная **data**) уже записаны и их можно печатать. Такая программа может работать неправильно в силу того, что операции записи в первом потоке (**Thread 1**) могли быть переупорядочены компилятором или процессором. Забегая вперед отметим, что с точки зрения стандарта C11 такая программа некорректна, поскольку в ней имеет место гонка данных. Более подробно вопросы синхронизации и понятие гонки данных будут рассмотрены в статье, посвященной модели памяти многопоточных программ.

**Пример 2.** Использование атомарных операций.

```
int data;
atomic_int flag = ATOMIC_VAR_INIT (0);
// Thread 1:
data = 123; // запись данных
atomic_store (&flag, 1)
// Thread 2:
if (atomic_load (&flag)) // обработка данных
  printf ("%d", data);
```

Такая программа будет работать корректно, использование описанных в подразд. 3.4 атомарных операций обеспечивает выполнение многопоточной программы в режиме последовательной консистентности. Это означает, что видимое поведение программы должно быть таким, как при последовательном перемежающемся выполнении вычислений из обоих потоков. В данном случае реализация должна гарантировать, что если второй поток прочитал значение переменной **flag**, равное **1**, в первом потоке уже было выполнено присваивание переменной **data**, и второй поток может начать обработку (печать) данных. Для этого, во-первых, при генерации кода операции записи в переменные **data** и **flag** не должны переупорядочиваться; во-вторых, если процессор поддерживает аппаратное планирование, то может потребоваться вставка инструкции, реализующей барьер памяти, чтобы эти операции не были переупорядочены во время выполнения.

Таким образом, атомарные операции не только обеспечивают атомарное выполнение операций над данными, но и являются инструментами синхронизации доступа к данным.

## 4. Функция быстрого завершения программы

Еще одно новшество C11, введенное для поддержки многопоточности — функция **quick_exit**:

```
#include <stdlib.h>
Noreturn void quick_exit(int status);
```

Она позволяет осуществить выход из программы с минимальной деинициализацией в тех случаях, когда нельзя использовать **exit()**, т. е. когда согласованное завершение потоков невозможно. Она выполняет обработчики, зарегистрированные при помощи функ-

ции `at_quick_exit` (в порядке, обратном регистрации), а затем вызывает `_Exit(status)`, которая, в отличие от `exit()`, не сбрасывает буферы файлов.

## 5. Вопросы оптимизации и генерации кода

Рассмотрим примеры принятых в современных компиляторах методов генерации кода (см. работу [4]), которые могут привести к ошибкам в работе многопоточных программ.

**Пример 1.** Запись в смежные поля структур. Пусть существует структура

```
struct {
  char a;
  int b:9;
  int c:7;
  char d; } s;
```

и поток 1 выполняет присваивание битовому полю `b: s.b = 0x12,` а поток 2 записывает значение в поле `a: s.a = 'n'`. Как правило, присваивание битовому полю реализуется как последовательность следующих операций:

1) считать слово, содержащее поле `b`, в регистр `t`;

2) при помощи битовых операций над регистром `t` сформировать требуемое значение в поле `b`;

3) записать в память значение регистра `t`.

Запись в память затрагивает не только поле `b`, но и значения соседних полей, хранящихся в том же слове. По этой причине, если запись `s.a = 'n'` произошла между шагами 1 и 3 в представленной выше последовательности, то ее результат будет потерян. В связи с этим фактом стандарт вводит понятие *позиции в памяти* (*memory location*) — скалярный объект или последовательность соседних битовых полей. Утверждается, что два потока могут изменять различные позиции в памяти, не создавая конфликтов.

Битовое поле и соседнее небитовое поле структуры всегда находятся в разных позициях памяти. Два битовых поля находятся в разных позициях памяти, если

- одно из них лежит во вложенной структуре, а другое нет;
- или между ними есть не битовое поле;
- или между ними есть битовое поле нулевого размера.

Например, в следующей структуре

```
struct {
char a;
int b:5, c:11, :0, d:8;
struct { int ee:8; } e;
}
```

есть четыре позиции памяти: каждое из полей `a, d, e, ee` составляет отдельную позицию в памяти, а поля `b`, `c` находятся в одной позиции. Соответственно, разные потоки не должны одновременно модифицировать поля `b` и `c`.

Компилятор для модификации полей структур должен использовать команды записи подходящих единиц памяти (байтов, полуслов) так, чтобы не затрагивать соседние позиции памяти.

**Пример 2.** Оптимизирующие преобразования программ. Этот пример показывает некорректность использования типовых механизмов оптимизации, которые применяют для экономии операций в условных операторах или переключателях. Пусть в программе, выполняемой потоком 1, есть переключатель

```
switch (y) {
   case 0: x = 17; w = 1; break;
   case 1: x = 17; w = 3; break;
   case 2: w = 9; break;
   case 3: x = 17; w = 1; break;
   case 4: x = 17; w = 3; break;
   case 5: x = 17; w = 9; break;
   default: x = 17; w = 42; break;}
```

Компилятор может "вынести за скобки" присваивание `x = 17`, сгенерировав код, эквивалентный следующему:

```
tmp = x; x = 17;
    switch (y) {
    case 0: w = 1; break;
    case 1: w = 3; break;
    case 2: x = tmp; w = 9; break;
    case 3: w = 1; break;
    case 4: w = 3; break;
    case 5: w = 9; break;
    default: w = 42; break;}
```

Пусть `y == 2` и `x! = 0`. Тогда, если поток 2 выполняет присваивание `x = 0` (между операциями `tmp = x` и `x = tmp` в потоке 1), то результат присваивания `x = 0` будет утерян.

Стандарт C11 запрещает применение подобных методов, если они могут привести к видимым изменениям в поведении программы. Например, если компилятору известно, что переменная `x` используется только в одном потоке, то данная оптимизация допустима.

Сообществами разработчиков компиляторов GCC [8] и Clang/LLVM [9, 10] проведен анализ проиллюстрированных выше проблемных вопросов, и в настоящее время основная работа по пересмотру методов оптимизации и генерации кода с учетом требований стандартов C11 и C++11 уже проделана.

## Заключение

В стандарте C11 впервые введено понятие многопоточной программы, оперирующей разделяемой памятью. Стандарт включает традиционные средства управления потоками и средства синхронизации — мьютексы и условные переменные. Кроме этого в стандарт вошли новые атомарные типы данных и операции над ними. В настоящей статье рассмотрены только простые атомарные операции, реализующие выполнение программы в режиме последовательной консистентности. Такой режим обеспечивает надеж-

ность и простоту понимания программы, но может серьезно ограничивать возможности оптимизации. Поэтому стандарт допускает менее строгие механизмы синхронизации вычислений, а также атомарные операции, не участвующие в синхронизации. Этим вопросам авторы планируют посвятить еще одну статью, где будет рассмотрена модель памяти многопоточной программы, а также некоторые аспекты поддержки многопоточности в компиляторах.

В настоящее время активно ведется работа по реализации этих средств в свободно-распространяемых компиляторах GCC и Clang/LLVM. Хотя на уровне синтаксиса входного языка и стандартных библиотек поддержка атомарных типов для языка C пока отсутствует, семантика атомарных операций уже реализована в виде встроенных функций, которые могут быть использованы для разработки приложений, а также для реализации стандартной библиотеки **<stdatomic.h>**.

**Список литературы**


1. **ISO/IEC 9899:2011** — Information technology — Programming Languages — C.

2. **Вьюкова Н. И., Галатенко В. А., Самборский С. В.** О стандарте C11 // Программная инженерия. 2013. № 8. С. 2—9.

3. **Информационная технология** — интерфейс мобильной операционной системы (POSIX). Часть 1: Интерфейс прикладных программ (API) [Язык программирования C]. М.: НИИСИ РАН, 1999.

4. **Boehm H.-J.** Threads Cannot be Implemented as a Library // Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation. Chicago, Illinois, USA. June 11—15, 2005. P. 261—268. URL: http://www.stanford.edu/class/cs240/readings/p261-boehm.pdf

5. **ISO/IEC 14882:2011** — Information technology — Programming Languages — C++.

6. **Shavit N. and Touitou D.** Software Transactional Memory // Proceedings of the 14th ACM Symposium on Principles of Distributed Computing, Ottawa, Ontario, Canada. August 20—23, 1995. P. 204—213. URL: http://www.cse.ohio-state.edu/~agrawal/ -788-su08/Papers/week4/shavit95software.pdf

7. **Галатенко В. А.** Программирование в стандарте POSIX. Курс лекций. М.: Интернет-университет информационных технологий, 2004.

8. **GCC**, the GNU Compiler Collection. URL: http://gcc.gnu.org.

9. **Clang:** a C language family frontend for LLVM. URL: http://clang.llvm.org.

10. **The LLVM** Compiler Infrastructure. URL: http://llvm.org.

УДК 004.272.44, 004.4.414, 004.81

**С. М. Салибекян**, канд. техн. наук, стар. препод., Московский институт электроники и математики научно-исследовательского университета "Высшая школа экономики", e-mail: salibek@yandex.ru,
**П. Б. Панфилов**, канд. техн. наук, доц., консультант, Министерство экономического развития Российской Федерации

# Анализ языка с помощью объектно-атрибутного подхода к организации вычислений

*Описывается применение объектно-атрибутного подхода к организации вычислений в вычислительной системе с управлением потоком данных (вычислительные системы dataflow-архитектуры) для создания систем анализа языка (компиляция, интерпретация, смысловой анализ) на примере создания языка программирования для программной модели суперкомпьютерной системы с управлением потоком данных. Показывается, что предложенный подход имеет преимущества перед анализом текста с помощью конечного автомата: бóльший функционал; возможность распараллеливания вычислений и реализации на распределенных вычислительных системах; отсутствие семантического разрыва между языком программирования и машинным языком, свойственного современным вычислительным системах с управлением потоком команд (вычислительные системы controlflow-архитектуры). Описывается также возможное применение объектно-атрибутного подхода для смыслового анализа естественного языка.*

***Ключевые слова:*** *объектно-атрибутная архитектура, синтаксическая диаграмма, программа разбора языка, компиляция, интерпретация, семантический анализ языка, обработка естественного языка*

## Актуальность исследования

Обработка языковой информации, включая анализ (распознавание) и синтез (генерацию) языка, составляет объект исследований одного из актуальных научных направлений в современной компьютерной науке, а именно вычислительной лингвистики (англ. *computational linguistics*). Это направление искусственного интеллекта (ИИ) занимается изучением вычислительных аспектов анализа/синтеза языка, разработкой алгоритмов и прикладных программных систем для обработки языковой информации. Данное научное направление в последнее время развивается весьма активно. В частности, подобластями вычислительной лингвистики являются: моделирование естественного языка на основе теории автоматов с применением контекстно-зависимых грамматик и линейно-ограниченных конечных автоматов (машин Тьюринга) [1]; вычислительная семантика, включающая в себя определение подходящих логик для представления лингвистического смысла; автоматизированная корпусная лингвистика; разбор и частеречная разметка естественного языка и генерация языка, включающие разборку языка на части и его сборку из частей; машинный перевод; нейронные сети [2]; теория фреймов [3], концепция "Смысл-текст" Игоря Мельчука [4]. В настоящей статье предлагается новый подход к распознаванию языка, а именно обработка и анализ языка на основе объектно-атрибутного подхода (ОАП), разработанного в МИЭМ НИУ ВШЭ [5—8]. Объектно-атрибутный подход был первоначально предложен для создания распределенных вычислительных систем (ВС)

и приложений, работающих в парадигме организации вычислений с управлением потоком данных (*dataflow*-парадигме [9]), с возможностью его использования как на программном, так и на аппаратном уровнях ВС. Подход, в частности, использовался для реализации компилятора среды программирования и моделирования суперкомпьютерной системы объектно-атрибутной архитектуры (ОА-архитектуры) [5], созданной в рамках НИР "Исследование и разработка архитектуры и среды программирования перспективной суперкомпьютерной системы на основе динамической модели вычислений с управлением потоком данных" по федеральной целевой программе "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007—2013 годы" (далее для краткости — НИР).

Задача разработки нового языка программирования и его компилятора для параллельной ВС ОА-архитектуры возникла в связи с тем, что, по мнению авторов, старые парадигмы программирования для описания работы такой системы оказались или неудобны, или неприменимы. Классическая императивная парадигма несовместима с парадигмой *dataflow*. Функциональные языки и акторная модель, в основном используемые для описания *dataflow*-вычислений, имеют ограниченные возможности абстракции данных и программы. Объектно-ориентированную и агентную парадигмы весьма непросто реализовывать на распределенных и параллельных ВС. Принимая во внимание отмеченные обстоятельства, в рамках НИР было принято решение реализовать и язык программирования *dataflow*-ВС, и компилятор, используемый для его распознавания, по единым принципам объектно-ориентированного подхода. При этом подход ОАП оказался применимым не только к аппаратной, но и к программной части ВС [5]. В результате аппаратура, язык программирования и компилятор, работающие по одним и тем же принципам, делают ВС ОА-архитектуры целостной и устраняют известный "семантический разрыв". Его суть в том, что различие между классическим языком программирования высокого уровня и машинным языком фон-неймановского компьютера приводит к большим вычислительным затратам на компиляцию и делает скомпилированные программы неоптимальными.

В ходе выполнения работ по НИР созданы "параллельный" объектно-атрибутный (ОА-язык) программирования и компилятор для него, обладающий следующими качествами: быстрый запуск среды программирования; распределенный режим работы компилятора; большое разнообразие конструкций ОА-языка; возможность добавления в ОА-язык конструкций языка Си для облегчения написания последовательных фрагментов программы. Был разработан синтаксис ОА-языка программирования, алгоритмы работы виртуальных функциональных устройств (ФУ), применяемых в процессе компиляции. Выработаны основные приемы создания систем разбора и трансляции для машинного языка высокого уровня ("контекстно-свободный язык" по иерархии Хомского). Следует отметить, что объектно-атрибутный подход к компиляции применим не только для работы с языками программирования высокого уровня, но и для распознавания естественного языка, где необходима развитая абстракция данных и программ.

## Использование объектно-атрибутного подхода к созданию компилятора

В соответствии с ОАП распределенная ВС *dataflow*-архитектуры представляет собой совокупность ФУ, обменивающихся между собой данными, оформленными в виде информационных пар (ИП). Такая пара представляет собой совокупность атрибута $a$ и нагрузки $l$, т. е. двойку:

$$c = \{a, l\},$$

где атрибут $a \in A$ ($A$ — множество атрибутов, по которым ФУ идентифицируют нагрузку $l$), а нагрузка $l \in L = \{\Re \cup \text{nil} \cup S \cup \Omega\}$ ($L$ — множество данных в нагрузке; $\Re$ — множество рациональных чисел; nil — обозначение пустой нагрузки; $\Omega$ — множество адресов глобальной памяти; $S \in \Sigma^*$ — множество цепочек символов, принадлежащих алфавиту $\Sigma$). Функциональные устройства по приходе к ним ИП записывают данные или адрес глобальной памяти из ее нагрузки в свой контекст (совокупность внутренних регистров, определяющих состояние ФУ). Нагрузка идентифицируется ФУ по ее атрибуту. При накоплении в контексте всех необходимых данных для осуществления операции ФУ проводит вычисления и выдачу ИП с результатом другим ФУ.

Информационные пары могут быть следующих двух видов:

- милликоманда — служит для обмена данными между ФУ (с помощью милликоманды для ФУ передается только один операнд);
- характеристическая ИП — служит для описания признака какого-либо объекта; атрибут ИП представляет собой индекс (уникальный идентификатор) признака, а в нагрузке хранится значение (характеристика) признака.

Информационные пары объединяются в капсулы (цепочки ИП) [5], которые либо используются для описания признаков какого-либо объекта (содержат характеристические ИП), либо они хранят последовательность милликоманд (миллипрограмму). Форматы милликоманды и характеристической ИП совпадают, что позволяет объединять данные и миллипрограмму в одной информационной конструкции. Такое решение напоминает принцип инкапсуляции (объединения в одной информационной конструкции данных и

программ), применяемый в объектно-ориентированном подходе к программированию (ООП). Однако информационные конструкции ОАП в отличие от ООП синтезируются не на стадии компиляции, а непосредственно во время вычислительного процесса [5], что делает вычислительную систему более гибкой.

Для пояснения предложенного объектно-атрибутного принципа распознавания текста рассмотрим небольшой пример реализации объектно-атрибутной системы (ОА-системы) для анализа языка программирования высокого уровня (компилятора ОА-языка, ОА-компилятора). Задача ОА-компилятора состоит в том, чтобы перевести программу на ОА-языке [5, 7, 8] в объектно-атрибутный образ (ОА-образ, совокупность информационных капсул и миллипрограмм, задающих алгоритм решения какой-либо задачи путем описания обмена данными между ФУ). Для этого осуществляется формирование индексов милликоманд, индексов атрибутов, индексов ячеек памяти ИП, капсул, графов/деревьев абстракций (совокупности капсул, связанных между собой ссылками). Объектно-атрибутный образ несколько напоминает байт-код для JAVA-машины, где программа также переводится в коды для управления виртуальным компьютером.

Для распознавания текста в ОА-системе, как и в любом ОА-приложении, применяют различные типы ФУ, сведения о которых представлены далее.

Функциональное устройство "Поиск" (*Find*) предназначено для поиска заданных ИП в одной капсуле и запуска миллипрограмм исходя из результата поиска: "удача" (*success*), когда выполняется условие поиска, и "неудача" (*fail*), когда условие поиска не выполняется. Это ФУ реализует следующие милликоманды (в скобках приведена мнемоника милликоманды):

- установка указателя на миллипрограмму, запускаемую при выполнении условия поиска (SuccessProgSet);
- установка указателя на миллипрограмму, запускаемую при невыполнении условия поиска (FailProgSet);
- установка указателя на капсулу, по которой будет осуществляться поиск (Set);
- поиск по капсуле одной информационной пары (FindIc), которая пришла в качестве нагрузки к милликоманде.

Таким образом, миллипрограмма, на которую указывает SuccessProg, выполняет роль блока "*then*" программы условного оператора, а FailProg — блока "*else*" условного оператора в классической конструкции языка программирования высокого уровня "*if-then-else*".

Для реализации множественного выбора (аналог конструкции "*switch*" в языке Си или "*case*" в языке Pascal) предназначено ФУ "Список" (*List*). В контексте этого ФУ находится ссылка на список капсул, где хранятся данные для осуществления поиска (рис. 1). В объектно-атрибутный список (ОА-список) кроме

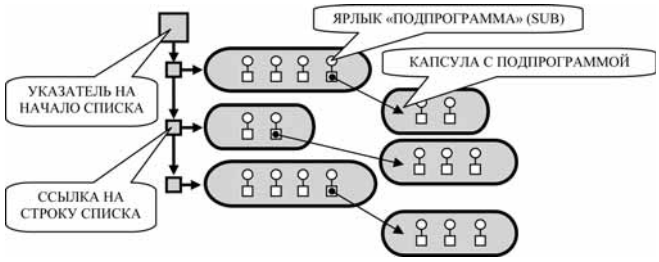


**Рис. 1. Информационная конструкция "Список"**

данных для поиска могут быть добавлены миллипрограммы обработки данных и осуществления перенаправления информационного потока. Объектно-атрибутный список может также использоваться в качестве таблицы переменных, где хранятся сведения обо всех мнемониках, что встречаются в распознаваемой программе высокого уровня (переменные, константы, указатели и т. д.). Следует отметить, что ОА-список дает программисту гораздо больше возможностей, чем оператор "*switch*" или "*case*". Например, в качестве критерия выбора одной из ветвей алгоритма используют не только целое число (как в операторе "*switch*"), но и любые константы, интервалы значений, "выколотые точки" в интервале значений. Возможен поиск в капсуле по нескольким критериям, поиск нескольких строк в ОА-списке и ряд других возможностей.

Источником данных для ОА-системы распознавания языка является ФУ лексического разбора, которое разбивает поступающий к нему текст на лексемы. В состав данного ФУ входит регистр, где хранится атрибут милликоманды, который прикрепляется к генерируемой лексеме. На начальной стадии разбора в этот регистр ФУ заносится милликоманда для первого ФУ, которое будет осуществлять синтаксический разбор лексемы в самом начале процесса компиляции. Затем ФУ, получившее лексему, перенастраивает лексический анализатор, чтобы тот выдавал лексему на ФУ, ответственное за следующую стадию синтаксического разбора. Функциональное устройство лексического разбора строится на основе классического конечного автомата, так как лексический анализ довольно прост. На выходе ФУ лексического разбора язык будет представлен в виде потока милликоманд с ИП, описывающими лексемы:

$$<M_k, <a, M>>,$$

где $M_k \in A$ — милликоманда для ФУ; $M$ — нагрузка, представляющая собой указатель на ячейку памяти, где хранится информационная конструкция, передаваемая в качестве операнда; <...> — обозначение ИП, $A$ — множество атрибутов милликоманд.

В процессе распознавания могут быть использованы не только перечисленные выше типы ФУ, но и другие ФУ, осуществляющие различные вычислитель-

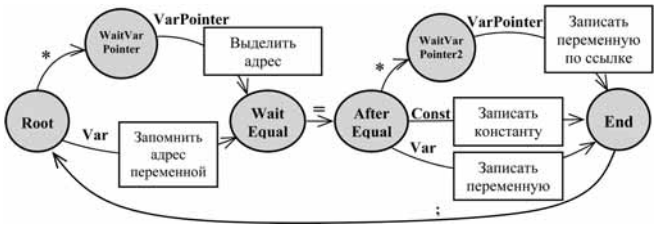


**Рис. 2. Синтаксическая диаграмма разбора оператора присваивания языка Си**

ные операции. К их числу относятся операции ввода-вывода, хранение информации и ряд других операций, что существенно повышает возможности ОА-системы компиляции по сравнению с классическим конечным автоматом, который "умеет" лишь записывать символы выходного алфавита на выходную ленту.

Создание ОА-компилятора можно разделить на следующие стадии. В начале создания ОА-компилятора необходимо составить синтаксическую диаграмму для распознаваемого языка. Затем для каждого узла диаграммы в ОА-системе программирования создается свое виртуальное ФУ: для тех вершин, из которых выходит только одна дуга, применяется ФУ "Поиск", для остальных — ФУ "Список". Создается также и ФУ лексического разбора, которое будет источником данных в ОА-системе. Далее происходит настройка созданных в среде ОА-программирования ФУ — и компилятор готов к разбору текста программы на языке программирования высокого уровня.

Для примера рассмотрим синтаксическую диаграмму (рис. 2) простейшего оператора присваивания языка Си:

$$(\text{Var} \mid (* \text{VarPointer})) = \\ = (\text{Const} \mid \text{Var} \mid (* \text{VarPointer})),$$

где Var — мнемоника переменной; Const — числовая константа; VarPointer — мнемоника указателя на переменную; "*" — символ, используемый для обозначения ссылки (указателя) в языке Си.

На рис. 2 видно, что на стадию синтаксического разбора приходят не отдельные символы, а ИП, обозначающие лексемы (поставщиком ИП с лексемами является ФУ лексического разбора). К дугам графа приписываются лексемы, по которым ОА-система переходит из одного состояния в другое. Однако в отличие от классической синтаксической диаграммы распознающего автомата, к дугам диаграммы также приписываются и действия, которые ОА-система выполняет, реагируя на приходящую лексему (а не символы, выдаваемые на выходную ленту, как у классического автомата-транслятора). Такое решение дает программисту намного больше возможностей при создании распознающей системы, чем конечный автомат. Как уже отмечалось ранее, каждому узлу диаграммы соответствует свое ФУ, и обозначение (мнемоника) этого ФУ приписано к вершине графа.

Работа ОА-системы распознавания происходит в соответствии со следующим алгоритмом. Перед началом работы осуществляется настройка всех ФУ. В том числе и настройка ФУ лексического анализа, чтобы оно выдавало первую лексему на ФУ, отвечающее за начало анализа (в нашем случае Root, рис. 2). Далее ФУ, которому от ФУ лексического разбора поступила ИП, анализирует пришедшую лексему, и в зависимости от результата анализа перенастраивает лексический анализатор таким образом, чтобы тот "выбрасывал" очередную лексему для ФУ, ответственного за следующую стадию синтаксического анализа.

Приведем фрагмент ОА-программы (рис. 3), соответствующий представленной выше синтаксической диаграмме (синтаксис ОА-языка описан в работе [7]). Как видно из листинга на рис. 3, который содержит описание инициализации только первых трех ФУ ОА-системы, при выполнении условия поиска ФУ Root с помощью милликоманды Lex.ReceiverMkSet перенастраивает ФУ лексического разбора таким образом, чтобы оно выдавало очередную лексему на следующую стадию синтаксического разбора (в нашем случае ФУ WaitEqual, см. рис. 2).

В листинге присутствуют следующие обозначения:

Root.Set — милликоманда установки списка для ФУ "Список" (Root);

> — знак начала новой капсулы, входящей в ОА-список;

Separator — атрибут символа-разделителя;

Lex.ReceiverMkSet — установка милликоманды, прикрепляемой к генерируемой лексеме;

nil — обозначает, что в качестве нагрузки ИП может выступать любая константа или указатель;

Var — атрибут переменной;

```
Root.Set=
>{Separator="*" Lex.ReceiverMkSet=WaitVarPointer.FindIc}
>{Var=nil Root.InObjPopMk=VarManager.Set  Lex.ReceiverMkSet=WaitEqual.FindIc}
>{0=nil Console.Out="Wrong variable describeing" Lex.Stop}

WaitVarPointer.Set={VarPointer=nil}
WaitVarPointer.SuccessProgSet=
     {WaitVarPointer.InObjPointPop=VarManager.Read  Lex.ReceiverMkSet=WaitEqual.FindIc}
WaitVarPointer.FailProgSet={Console.Out="Pointer not finded" Lex.Stop}

WaitEqual.Set={Separator="="}
WaitEqual.SuccessProgSet={Lex.ReceiverMkSet=AfterEgual.FindIc}
WaitEqual.FailProgSet={Console.Out=" '=' not finded"   Lex.Stop}
……
Lex.Lexing="x=10; y=*Uk" \*Запуск генератора лексем*\
```

**Рис. 3. Листинг ОА-программы синтаксического разбора**

FindIc — милликоманда поиска одной ИП в капсуле или списке;

VarManager — ФУ, ответственное за работу с переменными;

VarManager.Set — милликоманда установки указателя на переменную;

VarManager.Read — милликоманда чтения переменной из указанного в нагрузке адреса;

Console.Out — милликоманда вывода сообщения на консоль;

Lex — обозначение ФУ лексического разбора;

Lex.Stop —милликоманда остановки ФУ лексического разбора;

Lex.Lexing — запуск лексического разбора (в нагрузке милликоманды символьная строка для разбора).

Из приведенного примера видно, что программа на ОА-языке хорошо приспособлена для реализации алгоритма разбора языка, представленного в виде синтаксической диаграммы. Заметим, что этого нельзя сказать об императивном (процедурном), объектно-ориентированном или функциональном языках программирования. В ОА-языке каждой вершине синтаксического графа однозначно соответствует свое ФУ, а каждой дуге — миллипрограмма. Таким образом, ФУ работают асинхронно, и вычисления активизируются с помощью милликоманд, которыми ФУ обмениваются между собой. Такая организация вычислительного процесса позволяет обеспечить параллельность вычислений и работу компилятора на распределенных ВС. К тому же у ОА-системы существует возможность изменять информационные связи непосредственно во время вычислительного процесса, что значительно увеличивает гибкость организации вычислений. Информация, которая приписывается к дугам графа, — это милликоманды (токены), которыми обмениваются между собой ФУ (такой принцип вычислений как раз соответствует парадигме *dataflow*). Принцип *dataflow*, по которому работает ОА-система, обеспечивает распознающей системе большие возможности, а именно:

- распараллеливание и самораспараллеливание вычислений;
- реализация на распределенной и гетерогенной (состоящей из вычислительных узлов различной архитектуры) ВС;
- удобная декомпозиция задачи, когда процесс распознавания можно разделить на несколько уровней и несколько автономных подзадач.

Напомним, что конечный автомат, активно используемый в настоящее время для анализа языка, по определению является последовательным. В один момент времени автомат может находиться только в одном состоянии $q \in Q$, где $Q$ — множество всех возможных состояний автомата. Заметим, что недетерминированный автомат в плане параллелизма преимуществ не дает, так как он также работает в последовательном режиме, обрабатывая один символ с входной ленты в один момент времени. Параллелизм в классической системе распознавания языка может быть обеспечен лишь благодаря использованию нескольких уровней (проходов) компиляции. Например, лексический и синтаксический уровни анализа, где лексический автомат выделяет из входного текста лексемы и передает сформированный поток лексем на автомат синтаксического разбора для дальнейшего анализа. Как правило, больше двух-трех уровней анализа языка в современных компиляторах не применяется. В отличие от классического подхода, ОА-система компиляции состоит из совокупности параллельно работающих ФУ, которые синхронизируются приходом данных. А для более сложного смыслового распознавания текста (далее семантического уровня языка) конечный автомат неприменим. Он не может работать с абстракцией данных, где уже требуется "интеллектуальная" система. Объектно-атрибутный подход способен справиться с решением такой задачи, так как обладает следующими преимуществами, связанными с удобством абстракции данных.

- Высокий уровень абстракции данных и программ [8].
- Возможность совмещать в одной информационной структуре как данные, так и программу их обработки.
- Возможность динамического синтеза и перестройки абстрактных данных непосредственно во время вычислительного процесса. Это свойство особенно важно для смыслового анализа текста: смыслы (объекты), заложенные в тексте, в подавляющем большинстве случаев не вписываются в рамки стандартных шаблонов (фреймов или объектов). И именно динамический синтез абстракций позволит по заложенным в ОА-систему правилам создавать описания объектов, заранее неизвестных программисту.
- Изоморфизм ОА-системы на уровнях программы и данных. Изоморфизм — это свойство программы, когда изменение ее части не влечет нарушения целостности всей программы. Данное свойство необходимо для обучающихся интеллектуальных систем (и систем смыслового распознавания текста в том числе), так как им приходится постоянно пополнять базу знаний новыми данными, и это не должно нарушать целостности всей системы.

## Объектно-атрибутный подход к смысловому анализу языка

Объектно-атрибутный подход оказался весьма удобным для решения задач из области обработки естественного языка (*natural language processing, NLP*), включая синтаксический, морфологический, семантический анализы текста. Обработка естественного текста NLP — это общее направление ИИ и вычислитель-

ной лингвистики, которое изучает вопросы компьютерного анализа и синтеза естественных языков. Наиболее близкой к ОА-способу анализа текста является концепция советского лингвиста Игоря Мельчука "Смысл-текст" [4], где синтез смысловых конструкций осуществляется не напрямую, а в несколько этапов (уровней) на основании принципа "от простого — к сложному". Мельчук предлагает шесть уровней: семантический (уровень смысла); глубинно-морфологический; поверхностно-морфологический; глубинно-синтаксический; поверхностно-синтаксический; фонологический (уровень текста). Мельчук, правда, делает оговорку — число уровней может быть и другим (на усмотрение разработчика модели анализа языка). Смысл же, заложенный в тексте, Мельчук представляет в виде графа, вершинами которого являются так называемые "семы" — элементарные (неразложимые) смысловые единицы. В виде графов представляются и отдельные смысловые элементы на глубинно-морфологическом и поверхностно-морфологическом уровнях. Однако концепция "Смысл-текст" не получила широкого распространения ввиду своей сложности и, самое главное, негибкости. Так, семы в концепции Мельчука представляли собой несколько видов предикатов с фиксированным числом аргументов. Это существенно затрудняло описание алгоритма распознавания текста и лишало его свойства изоморфизма.

В предлагаемой ОА-концепции смыслового анализа текста разбор также осуществляется от простого к сложному (рис. 4). Процесс синтеза смысловых конструкций начинается с информационных атомов (элементарных смыслов). В случае анализа текста атомами являются символы исходного теста. На первом этапе осуществляется лексический анализ, т. е. выделение из последовательности символов лексем. Этой работой в ОА-системе занимается ФУ лексического разбора. Далее лексемы поступают на этап синтаксического анализа, где происходит анализ взаимосвязи лексем. На этом этапе формируемые абстракции довольно сложны и представляют собой информационные капсулы или объектно-атрибутные информационные деревья (смысловой граф) [5, 8]. Синтезированные абстракции передаются для дальнейшей обработки на следующий уровень анализа, где другие ФУ синтезируют из них еще более сложные абстракции. Так происходит до тех пор, пока не будет получена абстракция, описывающая смысл всего распознаваемого текста (ключевая абстракция). Основная нагрузка по синтезу абстракций на уровнях от синтаксического и выше ложится на ФУ типов "Поиск" (*Find*) и "Список" (*List*), которые проводят анализ поступивших к ним информационных конструкций и осуществляют вызов соответствующих миллипрограмм для их обработки. Процесс анализа текста представляет собой так называемый "конус абстракций". В основании конуса располагаются информационные атомы (символы текста), в вершине — ключевая абстракция. Если кроме смыслового распознавания необходимо осуществить перевод текста на другой язык, то прямой конус абстракций дополняется конусом обратным, который осуществляет синтез текста на другом языке из ключевой абстракции по принципу "от сложного — к простому" (рис. 4).

Синтезированный объектно-атрибутный граф (ОА-граф) можно использовать, в частности, для семантического поиска. Технология семантического поиска следующая: пользователь вводит запрос на естественном языке, запрос преобразуется в семантический ОА-граф, а далее осуществляется поиск вхождения ОА-графа запроса в ОА-граф, описывающий текст. Для поиска информации в семантическом графе в ФУ "Поиск" и ФУ "Список" добавляются милликоманды поиска по ОА-графу, чтобы была возможность сравнения описаний сложно структурированных объектов.

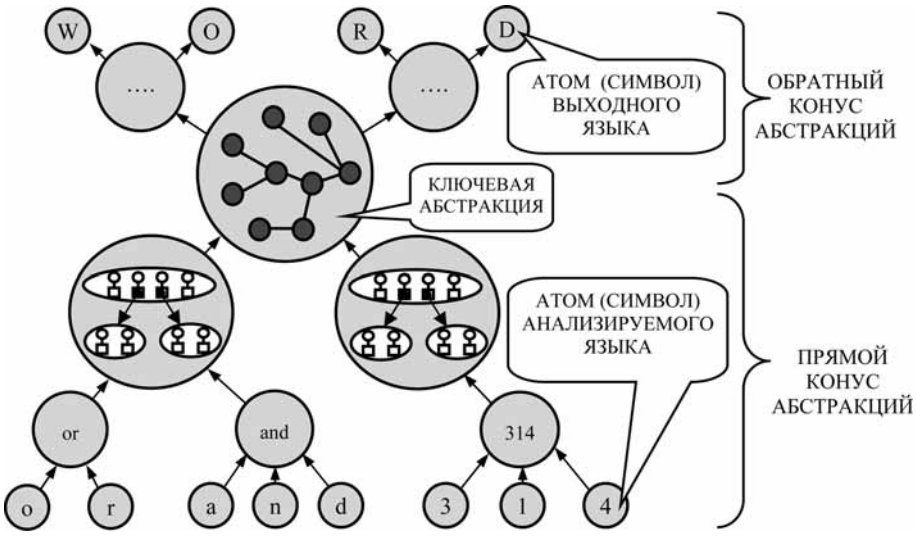


**Рис. 4. Объектно-атрибутный конус абстракций**

Для описания алгоритмов обработки при сравнении ОА-графов была разработана объектно-атрибутная алгебра, в которой даны следующие определения.

1. Эквивалентность ИП: $c_1 = \{a_1, l_1\}$ и $c_2 = \{a_2, l_2\}$ эквивалентны ($c_1 = c_2$) если $a_1 = a_2$ и $l_1 = l_2$.

2. Цепочка ИП — это последовательность ИП, объединенных в информационную капсулу ($\omega$ — обозначение пустой цепочки ИП).

3. Эквивалентность цепочек ИП $\eta_1$ и $\eta_2$: цепочки эквивалентны, если $|\eta_1| = |\eta_2|$, где $|\eta_1|$ и $|\eta_2|$ — длины цепочек $\eta_1$ и $\eta_2$ соответственно, и $\eta_1 =$

$= \eta_2 = \omega$ или $\eta_{1,i} = \eta_{2,i}$, $i = 1, 2, ..., |\eta_1|$, где индекс обозначает порядковый номер ИП в цепочке.

4. Операции над цепочками ИП $\eta_1$ и $\eta_2$:

- конкатенация (объединение): $\eta_1 \cup \eta_2$,
- пересечение: $\eta_1 \cap \eta_2$.

Цепочка $\gamma$ является пересечением цепочек ИП $\alpha$ и $\beta$ ($\alpha \cap \beta$) в том случае, если она содержит в себе ИП, удовлетворяющие следующему условию: $\gamma = \{a_i \in \alpha: \exists\ b_j \in \beta$, такое что, $a_i = b_j\}$, где $i = 1, 2, ..., |\alpha|$, $j = 1, 2, ..., |\beta|$.

5. Связка цепочек ИП $W$ — это взаимосвязанная структура данных, состоящая из цепочек ИП. Информационная пара $W = \{w_1, w_2, ..., w_{|W|}\}$, где $|W|$ — длина связки цепочек символов. Для индексации конкретной ИП в связке цепочек ИП можно пользоваться двойным индексом: $W_{ij}$ — это ИП, которая содержится в $j$-ой ИП $i$-ой цепочки ИП из связки $W$. Связка цепочек ИП $W$ используется в качестве аналога общей памяти компьютера, в которой располагаются данные, к которым имеют доступ все вычислительные устройства.

6. Объектно-атрибутный граф — набор цепочек ИП (капсул), объединенных ссылками, расположенными в нагрузках ИП (рис. 5). Для удобства обработки все цепочки ИП, входящие в ОА-граф, объединяются в связку $V$ цепочек ИП. Тогда в качестве множества адресов, располагающихся в нагрузках ИП, можно использовать номера (индексы) цепочек ИП в связке $V$.

7. Переиндексация графа — изменение последовательности ИП ОА-графа (т. е. изменение индексов у ИП, входящих в ОА-граф).

8. Эквивалентности ОА-графов: ОА-графы $G_1$ и $G_2$ эквивалентны, если $|G_1| = |G_2|$ и $G_{1i} = G_{2i}$, где $i = 1, 2, ..., |G_1|$.

9. Изоморфность ОА-графов: ОА-графы изоморфны, если можно подобрать такую переиндексацию ОА-графов, что они будут эквивалентны.

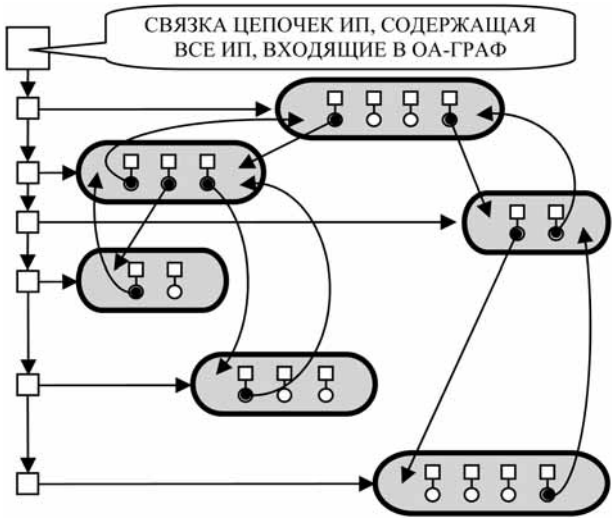


**Рис. 5. Объектно-атрибутный граф**

10. Частичная изоморфность ОА-графов: ОА-графы частично изоморфны, если можно подобрать такую переиндексацию $G_1$ и $G_2$, что $G_{1i} \cap G_{2i} = G_{1i}$, где $i = 1, ..., |G_1|$. Частичная изоморфность $G_1$ и $G_2$ означает, что $G_1$ является подграфом в $G_2$.

В рамках НИР[1] по созданию вычислительных систем для семантического поиска текстовой информации был реализован уровень синтаксического анализа текста, в который на обработку поступают ОА-капсулы с описанием лексем, а на выходе выдается семантический граф, описывающий смысл текста. Объектно-атрибутная алгебра была реализована программно в ФУ "Поиск" и ФУ "Список", что позволило с помощью ОА-языка описывать сложные алгоритмы семантического разбора языка. В экспериментальную базу знаний, созданную для распознавания русского языка, помещались описания лексем (существительные, глаголы, прилагательные, наречия). Для каждой лексемы были заданы как синтаксические свойства, так и свойства семантические (т. е. описание объекта, с которым ассоциируется лексема).

Синтез ОА-графа на синтаксическом уровне осуществляется следующим образом.

1. Поиск поступающей на анализ лексемы в синтаксическом словаре (в словарь помещают синтаксические и семантические свойства лексем).

2. Переписывание найденной капсулы в ОА-список лексем (список лексем рационально формировать не более чем для одного предложения).

3. Свертка наречий с глаголами и прилагательными (свертка заключается в том, что семантические свойства из наречий переписываются в капсулы связанных с ними глаголов или прилагательных, а капсулы с описанием наречий удаляются из списка лексем).

4. Свертка прилагательных и существительных.

5. Установка связей (заданных глаголами) между объектами (заданными именами существительными). В результате происходит формирование семантического ОА-графа.

6. Разбор более сложных синтаксических конструкций языка.

Экспериментальная база знаний по анализу русского языка проводит проверку согласования лексем по времени, роду и числу, а также проверку правильности синтаксических конструкций. База знаний насчитывает порядка 100 лексем и реализует более 30 синтаксических конструкций русского языка.

[1] В данной научной работе использованы результаты проекта "Разработка и исследование новых методов создания распределенных энергосберегающих вычислительных систем для семантического поиска текстовой информации на основе модели вычислений с управлением потоком данных", выполняемого в рамках Программы фундаментальных исследований НИУ ВШЭ в 2013 г.

## Выводы

В области распознавания текста с применением объектно-атрибутного подхода решены следующие задачи:

- сформулированы концептуальные положения процесса разбора языка (компиляции, трансляции, смыслового анализа) с помощью объектно-атрибутной вычислительной среды *dataflow*-архитектуры;
- отработаны основные типы ФУ, используемые для языкового анализа;
- реализованы виртуальные ФУ, участвующие в процессе распознавания языка;
- с помощью ОА-среды программирования создан ОА-компилятор, который осуществляет перевод программы, написанной на специальном ОА-языке программирования высокого уровня в ОА-образ (совокупность информационных капсул и миллипрограмм, описывающих алгоритм решения какой-либо вычислительной задачи);
- создана экспериментальная ОА-система для семантического распознавания текста на русском языке.

Объектно-атрибутный подход, обладающий большой гибкостью при построении смысловых конструкций и возможностью работы с абстрактными данными, эффективен не только при анализе машинных языков высокого уровня, относящихся к классу контекстно-независимых по иерархии Хомского. Он применим и для анализа "естественных" языков, относящихся к контексто-зависимым и свободным. Так, в ОА-модель языка можно, например, добавлять признаки стилистической окраски текста, ассоциативные связи и другие механизмы.

Семантический разбор языка, основанный на ОАП, обладает целым набором положительных качеств, которых не может обеспечить никакой другой способ. К числу таких качеств относятся: объектный принцип обработки данных, обеспечивающий высокий уровень абстракции данных и программы; синтез семантического графа от простого к сложному; соединение в одной информационной конструкции как данных, так и программного кода для его обработки; возможность распараллеливания вычислений и реализации на распределенной гетерогенной ВС; изоморфизм системы распознания текста; гибкость организации анализа языка (ОА-граф легко модифицируется путем добавления и удаления ИП из него), что позволяет генерировать описания объектов любой сложности (в том числе и описания объектов, которые не были заранее предусмотрены программистом).

**Список литературы**


1. **Ахо А., Ульман Дж.** Теория синтаксического анализа, перевода и компиляции. М.: Мир, 1978.
2. **Круглов В. В., Борисов В. В.** Искусственные нейронные сети. Теория и практика. 2-е изд., стереотип. М.: Горячая линия — Телеком, 2002.
3. **Минский М.** Фреймы для представления знаний. Пер. с англ. — М.: Энергия, 1979.
4. **Мельчук И. А.** Опыт теории лингвистических моделей "СМЫСЛ <—>ТЕКСТ". М.: Школа "Языки русской литературы", 1999.
5. **Салибекян С. М., Панфилов П. Б.** Объектно-атрибутная архитектура — новый подход к созданию объектных систем // Информационные технологии. 2012. № 2. С. 8—14.
6. **Салибекян С. М., Панфилов П. Б.** Объектно-атрибутный подход к построению интеллектуальных систем // Нейрокомпьютеры: разработка и применение. 2011. № 11. С. 9—17.
7. **Salibekyan S. M., Panfilov P. B.** Object-attribute architecture for design and modeling of distribute automation system // Automation and remote control. 2012. Vol. 73, N 3. P. 587—595, DOI: 10.1134/S0005117912030174.
8. **Салибекян С. М., Панфилов П. Б.** Моделирование суперкомпьютерной вычислительной системы объектно-атрибутной архитектуры с управлением потоком данных // Информационные технологии и вычислительные системы. 2013. № 1. С. 3—10.
9. **Silk J., Robic B. and Ungerer T.** Asynchrony in parallel computing: From dataflow to multithreading. Institut Jozef Stefan, Technical Report CDS-97-4, September 1997.

УДК 004.4'23

**М. А. Беляев**, диспетчер отдела системного администрирования и технической поддержки, Учебный научно-практический вычислительный центр Донецкого национального университета, e-mail: belyaev.mark@gmail.com,
**А. В. Беляев,** канд физ.-мат. наук., доц., Донецкий государственный университет управления, e-mail: nika@vnet.dn.ua

# О проектировании расширяемых и отказоустойчивых систем, реализующих паттерн MVVM в рамках инфраструктуры Microsoft.NET

*Анализируется принципиальная возможность применения идей автоматного программирования и базовых концепций паттерна MVVM для проектирования отказоустойчивых и легко расширяемых приложений с использованием средств, предоставляемых платформой Microsoft.NET, на примере программы, автоматизирующей некоторые функции работы с графами.*

***Ключевые слова:*** *паттерн "модель—представление—модель представления" (MVVM), конечные автоматы, паттерны проектирования, технология Microsoft.NET*

Задача организации взаимодействия интерфейса и логики программы, которое бы позволяло легко расширять, изменять и отлаживать уже написанные ее части, возникла не сразу после появления интуитивно понятных пользовательских интерфейсов. Однако сегодня и в веб-программировании, и в программировании "десктопных" систем данная задача стала рассматриваться все чаще. Такое положение дел вполне объяснимо. Необходимо отметить, что сам термин "веб-программирование" возник не сразу в силу того, что на этапе создания сети Интернет и протокола http в браузерах пользователям показывались только статичные html-страницы. С течением времени стали появляться технологии динамической генерации html-страниц на сервере (Perl, ASP, PHP, ASP.Net) либо на клиентской машине пользователя (JavaScript, Flash, Silverlight). Динамическая генерация при этом, конечно, подразумевала наличие минимальной бизнес-логики при демонстрации веб-страницы пользователю. Вместе с тем вопросы написания расширяемых и отказоустойчивых веб-систем еще не стояли остро в силу незавершенности самих этих технологий. Что касается "десктопных" систем, естественным является факт наличия в них бизнес-логики даже с момента их возникновения. Тем не менее, на этапе своего появления они были настолько сложны (win API), что решению проблемы взаимодействия представления и бизнес-логики программы должно было предшествовать хотя бы какое-нибудь упрощение этих технологий с точки зрения программирования прикладных программ. При этом следует заметить, что в рамках данной технологии представление и логика приложения в общем случае еще даже не были разъединены, так как не существовало явных механизмов, препятствующих смешиванию кода, отвечающего за прорисовку пользовательского интерфейса и за бизнес-логику приложения. Упрощением сложного win API для написания визуальных приложений стало, как известно, создание технологии MFC, а затем и технологии windows Forms. Параллельно с этим создавались системы RAD, скрывающие от прикладного программиста все сложности организации базовых элементов пользовательского интерфейса (например, Borland C++ Builder, Borland Delphi, Microsoft Visual Studio и т. д.). Таким образом, было достигнуто базовое разделение между кодами представления и бизнес-логики. В веб-программирование такое базовое разделение пришло под видом технологии Microsoft ASP.Net Web Forms (представление — aspx-страница, code behind — бизнес-логика). Тем не менее, код бизнес-логики все

еще мог быть связанным с кодом представления в силу того, что элементы представления были доступны в коде бизнес-логики в виде специальных объектов. Такая половинчатость в разделении представления и логики приложения, конечно, упрощала процесс создания, отладки и поддержки программ, но, как оказалось, могла быть в дальнейшем преодолена.

Отметим, что параллельно с развитием технологий разработки программного обеспечения развивалась и методология программирования. Несмотря на постоянное совершенствование средств разработки, вопросы "разбухания", повторения одних и тех же частей кода и другие вопросы качества кода вынуждали самих программистов задумываться о создании неких общепринятых способов организации архитектуры создаваемых ими программных систем. Безусловно, весьма значимым шагом в этом направлении был выход в 1995 г. книги "Design Patterns" [1], описывающей наиболее удачные способы (паттерны) использования средств объектно-ориентированного программирования для создания прикладных приложений с определенными ограничениями на предметную область. Кроме того, из всего множества существовавших на тот момент паттернов проектирования, наконец, выделились паттерны, разделяющие архитектуру приложения на несколько независимых частей и оформляющие взаимоотношения между этими частями. Появление таких паттернов дало, хотя и не сразу, новый импульс развитию сложных программных решений и даже новых технологий разработки приложений. В свое время, например, незамеченными широкой общественностью остались идеи Трюгве Реенскаугома, а затем и труд Стива Бурбека "Applications Programming in Smalltalk-80: How to use Model-View-Controller" [2], которые фактически дали рождение таким серьезным платформенным проектам, как Ruby on Rails (2004 г.) или Microsoft MVC (2007 г.). На волне возрождения идей Т. Реенскаугома возникли также и другие паттерны, повторяющие в целом идеи MVC, но вносящие в него определенные коррективы согласно требованиям каждой отдельно взятой предметной области. Таким паттерном был MVP (Model-View-Presenter), описанный Майком Потелом в его статье "MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java" [3]. Model-View-ViewModel (MVVM) — еще один шаблон проектирования архитектуры приложения, представленный Мартином Фоулером как модификация шаблона Presentation Model [4]. Паттерн MVVM ориентирован на современные платформы разработки, такие как Windows Presentation Foundation, Silverlight от компании Microsoft, ZK framework, Knockout.js и др. Таким образом, средства разработки программного обеспечения стали развиваться с учетом выработанных и оправдавших себя паттернов проектирования (WPF для "десктопных систем" — паттерн MVVM, ASP.Net MVC и др. — в веб-программировании). У программистов появилась возможность окончательно разделить представление и бизнес-логику и самим решать, как организовывать взаимодействие между ними. В настоящее время все три шаблона проектирования (MVC, MVP, MVVM) активно используют в проектах, в которых надежность написанного кода, расширяемость функциональности приложения и возможность модульного тестирования существенны.

Если говорить о ближнем зарубежье, то в контексте разработки отказоустойчивых приложений необходимо упомянуть имя российского исследователя А. А. Шалыто, разработавшего так называемую "switch-технологию" [5]. Ее суть заключается в том, что поведение некоторых программных комплексов в его отдельных частях может быть смоделировано поведением специально построенного конечного автомата.

В настоящей статье рассматривается принципиальная возможность и способ применения идей А. А. Шалыто при проектировании программ в соответствии с шаблоном MVVM в рамках средств, предоставляемых разработчику платформой Microsoft.NET на примере приложения, автоматизирующего некоторые функции работы с графами.

Традиционно паттерн MVVM (модель — представление — модель представления) применяют при проектировании приложений в среде, в которой реализована возможность двухсторонней синхронизации данных пользовательского интерфейса с данными некоторого экземпляра определенного класса. Именно такую среду разработки предоставляет программисту технология Windows Presentation Foundation (WPF), представляющая собой подмножество инфраструктуры Microsoft.NET в области создания программ для "десктопных" систем. Отметим, что указанная выше возможность связывания данных объектов с данными интерфейса не ограничивается лишь синхронизацией значений некоторых переменных. В контексте рассматриваемой в данной статье задачи особенный интерес представляет система команд, логически продолжающая концепцию связывания данных. По сути, в рамках WPF к интерфейсу можно привязывать не только данные, но и методы определенного класса через упомянутую выше систему команд. Более того, возможность выполнения каждой отдельно взятой команды может динамически регламентироваться специального вида предикатом. Такие средства в совокупности с базовыми принципами паттерна MVVM открывают выход на идеи так называемого "автоматного" программирования, которые последнее время весьма активно развивают профессиональные программисты [6].

В самом деле, формализм конечных автоматов достаточно универсален. Он позволяет решать самые разные прикладные задачи — этим обусловливаются его постоянная популярность и актуальность. Интересно, что свое применение данный формализм нашел и в области разработки паттернов проектирования, породив в этой области серию интересных архитектурных решений. В качестве примера можно привести паттерн проектирования State, в основу которого легли принципы работы конечного автомата [7].

Впрочем следует признать, что сфера применения данного паттерна весьма ограничена.

Совсем иначе дело обстоит с уже упомянутыми выше идеями "автоматного" программирования, на которых следует остановиться отдельно. Мысль о том, что событийная модель любого приложения гармонично проецируется на конечный автомат, весьма перспективна. Такой подход к проектированию программных комплексов вносит некоторую определенность в недетерминированную по своей природе систему взаимодействий частей более или менее сложной программы.

В то же время надо признать, что отмеченная выше мысль вовсе не является единственно возможным способом применения идеи конечного автомата в решении задачи поиска отказоустойчивых и расширяемых архитектурных форм. Данное утверждение подтверждается статьей А. А. Шалыто "Применение конечных автоматов при программировании мобильных устройств" [8]. В ней конструктивным способом доказывается возможность моделирования работы некоторых компьютерных программ с помощью конечного автомата. Идеи, представленные в данной статье, закрывают вопрос поиска паттернов создания отказоустойчивых приложений для класса задач, которые могут быть решены с использованием регулярных языков (которые определяются конечными автоматами). Имея возможность моделировать работу программы с помощью конечного автомата, можно еще на этапе разработки до конца отладить проектируемое программное решение и точно описать поведение этого решения в любой ситуации. Принципиальная возможность сделать это обусловливается детерминированной природой конечного автомата. Таким образом, в итоге получаем способ проектирования приложения, который уже на этапе создания алгоритма полностью, в случае корректного моделирования работы программы, исключает возможность возникновения каких-либо непредвиденных ошибок в работе. При этом вся логика, ответственная за реакцию программы на управляющее воздействие (событие элемента управления, нажатие кнопки и т. д.), выносится в отдельную структуру данных. Работа этой структуры данных обеспечивается минимальным числом ресурсов системы (в оперативной памяти хранится только таблица переходов и алфавит автомата, элементы которых можно хранить в переменных целого типа, а проход по таблице переходов в общем случае может осуществляться напрямую по адресам состояний). При этом из процесса реагирования программы на управляющее воздействие исключается этап поиска действий, которые должны быть впоследствии проведены, в силу того, что набор этих действий был строго определен на этапе проектирования автомата. Исключение указанного этапа из общего цикла работы программы, очевидно, сокращает время отклика системы и тем самым повышает ее общую производительность. Вместе с тем такой подход к разработке приложений не лишен определенных недостатков, а именно:

- Создание конечного автомата или совокупности автоматов, моделирующих работу программного комплекса, является достаточно трудоемкой задачей. Следует признать, однако, что результатом проделанной работы становится существенное сокращение времени, потраченного на тестирование приложения и на выявление/исправление непредвиденных ошибок в его работе.
- Серьезные проблемы с расширяемостью программного комплекса. Необходимо признать, что данный недостаток наиболее существенен и ничем не компенсируется. Более того, необходимость практически заново создавать модель приложения при добавлении в него новой функциональной возможности обостряет первый недостаток данного метода тем больше, чем чаще вносятся изменения в уже написанную программу.

Указанные выше недостатки, безусловно, затрудняют использование формализма конечных автоматов во время проектирования. В то же время сложно отказываться от преимуществ, возникающих при таком их использовании. Очевидно, что и слабая расширяемость, и сложность применения конечных автоматов в программировании связаны с необходимостью явно задавать некоторый конечный автомат. В уже упомянутой статье [8] А. А. Шалыто замечает, что неявно реализованный конечный автомат делает потенциально возможным появление непредвиденных ошибок, в то время как явно реализованный позволяет этих ошибок избежать. В свете изложенного выше становится понятно, что обе крайности едва ли могут претендовать на роль универсального решения. Логично предполагать, что оптимальная структура программы должна представлять собой некоторый компромисс между формализацией и удобством использования.

Рассмотрим схему устройства программы, представленную на рис. 1.

Данная схема реализует некоторое стабилизирующее уточнение паттерна MVVM. Заключение программной системы, построенной согласно концепциям паттерна MVVM, в такие жесткие рамки превращает ее в конечный автомат, функционирующий по следующему принципу.

- Текущее состояние автомата определяется как совокупность команд, которые могут быть выполнены в данный момент времени.
- Начальное состояние автомата определяется начальным состоянием модели данных и состоянием модели представления программной системы.
- Любое воздействие пользователя на интерфейс программы в произвольный момент времени инициирует вызов той или иной команды из модели представления, причем любая вызванная пользователем команда потенциально оказывает воздействие на модель данных программы. С этой точки зрения выполнение любой команды модели представления эквива-

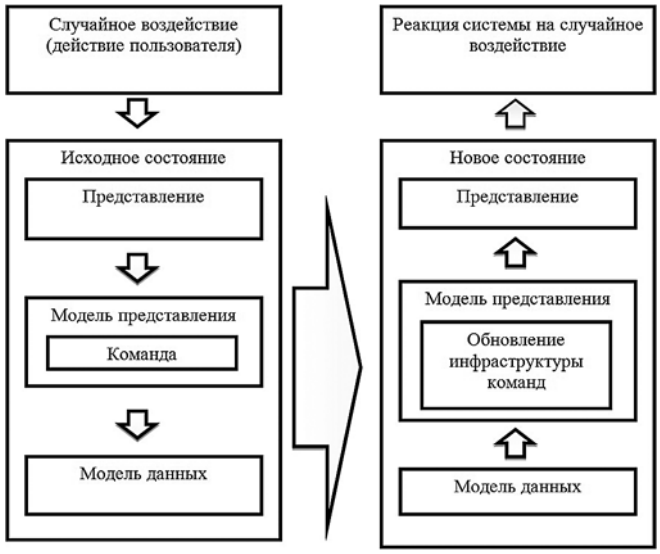


**Рис. 1. Архитектура программы, реализующей паттерн MVVM с его стабилизирующим уточнением**

лентно поступлению на вход конечного автомата некоторого входного символа. Таким образом, входной алфавит автомата определяется как совокупность всех команд модели представления, каждая из которых соответствует некоторому входному символу конечного автомата. Представление не может вызвать ничего, кроме команды, определенной в модели представления (на вход автомата не может поступать ничего, кроме символа входного алфавита). Вызов команды может быть осуществлен лишь после того, как была проведена проверка возможности такого вызова на этапе последнего воздействия на данные программной системы (состав входного алфавита сужается либо обратно расширяется в соответствии с состоянием программной системы). Все случайные воздействия на представление фиксируются лишь фактом вызова какой-нибудь команды модели представления и игнорируются в случае отсутствия такого вызова.

- Воздействие, оказанное на модель данных в результате выполнения определенной команды модели представления, в общем случае переводит модель данных программы в новое состояние. Поэтому выполнение каждой команды модели представления должно завершаться обновлением всей инфраструктуры команд данной модели представления (под обновлением инфраструктуры команд понимается последовательный вызов всех предикатов, определяющих возможность выполнения той или иной команды). Обновление инфраструктуры команд либо переводит систему в новое состояние (если для выполнения становятся доступными или блокируются новые команды), либо оставляет ее в том же состоянии, в котором она находилась до выполнения данной команды.

- Функция переходов рассматриваемого конечного автомата определяется самой программной системой согласно набору предикатов команд, составленных программистом. В общем случае такая функция в рассматриваемой ситуации не является однозначной, так как новое состояние системы не определяется лишь новым входным символом и текущим состоянием. Однако эта недетерминированность совершенно допустима с точки зрения проектируемой программы. Причина в том, что несмотря на недетерминированность функции перехода, программа ведет себя вполне определенно, а именно в любой момент времени она находится в каком-нибудь из состояний автомата. На практике это означает следующее: что бы ни происходило с данными системы во время выполнения, программа, как система управления, анализируя эти данные будет адекватно реагировать на их изменение, и такая реакция будет выражена в изменениях, осуществляемых ею в представлении. Концепция последовательного выполнения программы неустойчива в том смысле, что коллапс данных в такой программе заводит ее выполнение в тупик. Динамический поиск правильного способа регулировки сложившейся исключительной ситуации далеко не всегда завершается успешно. В общем случае такая программа останавливается. В то же время конечно-автоматная схема функционирования, описанная выше, очень гладко обрабатывает такие ситуации в силу уже упоминавшихся ранее причин, не останавливая общий ход выполнения программы.

Итак, конструктивно было показано, что при соответствующих ограничениях на модель, представление и модель представления в рамках паттерна MVVM программная система функционирует как недетерминированный конечный автомат определенного вида.

Резюмируя все изложенное выше заметим, что функционирование программной системы по принципу конечного автомата в значительной мере ее стабилизирует, однако такие системы, конечно, трудно расширять. Предложенное выше архитектурное решение показывает принципиальную возможность решения задачи расширяемости программы, использующей конечные автоматы в своей структуре. Расширяемость в данном случае достигается за счет недетерминированности конечного автомата, лежащего в основе функционирования системы. По сути, при задании автомата не нужно выписывать в явном виде ни его функции переходов, ни всех его состояний. Однако их число конечно, и достаточно того, чтобы этот автомат просто существовал. В таком случае подконтрольная система не будет делать ничего кроме реагирования на то или иное состояние своих данных изменением своего представления. Таким образом, добавление новых команд в модель представления никак не будет отражаться на уже существующей схеме работы программы, если, конечно, добавление такой команды не должно изменить работу уже существующей части программы. В таком случае будут бессильны любые паттерны проектирования. И все же приходим к некоторому компромиссу между детерминированностью поведения программы и возможностью ее расширяемости.

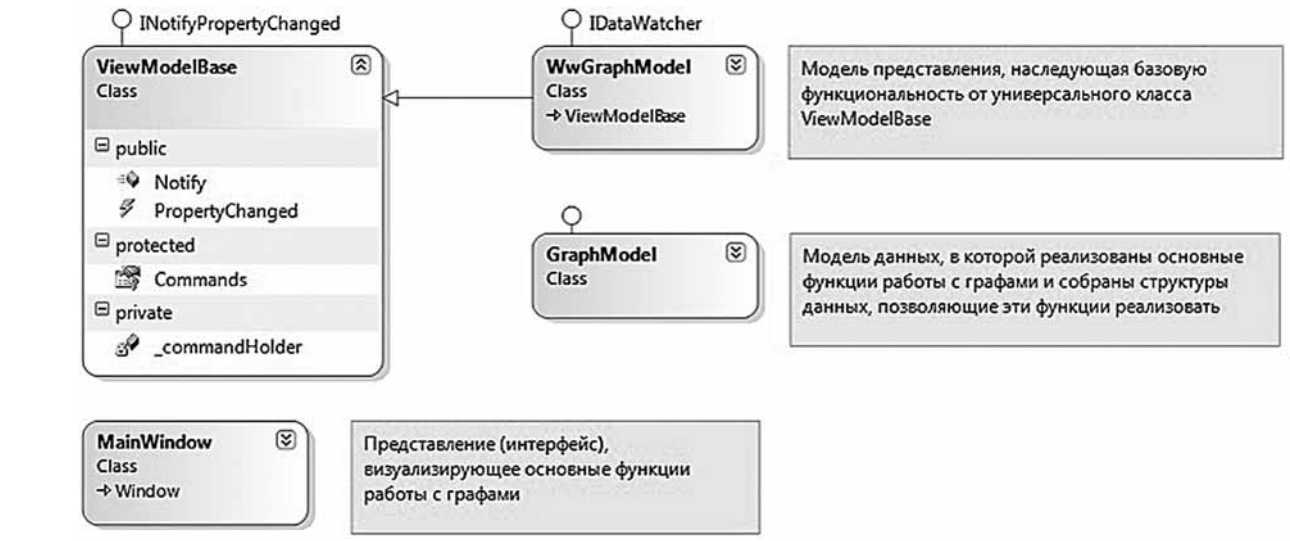


**Рис. 2. Диаграмма основных классов, использовавшихся в программе**

Далее рассмотрим реализацию идей, изложенных выше, на примере программы, автоматизирующей некоторые функции работы с графами.

Вначале определим классы модели, представления и модели представления. Назовем их соответственно GraphModel, MainWindow и WwGraphModel. Класс GraphModel будет реализовывать основную логику работы программы, предоставляя некоторый интерфейс вызовов внешней среде (рис. 2).

Класс MainWindow будет являться представлением в проектируемой программе. В его конструкторе будет создаваться экземпляр модели представления, назначаемый контекстом данных рассматриваемого представления. В рамках данного класса будут описываться все визуальные компоненты программы (рис. 3).

Класс WwGraphModel будет обрабатывать воздействия пользователя на интерфейс системы, собирая данные из представления и инициируя выполнение требуемых операций класса GraphModel. Заметим, что данный класс будет наследником класса ViewModelBase, созданного S. Jack [9] для работы в рамках паттерна MVVM. Данная версия общего класса модели представления кажется авторам наиболее удобной с точки зрения работы с инфраструктурой команд приложения в WPF и с точки зрения удобства реализации описанного в данной статье подхода к со-

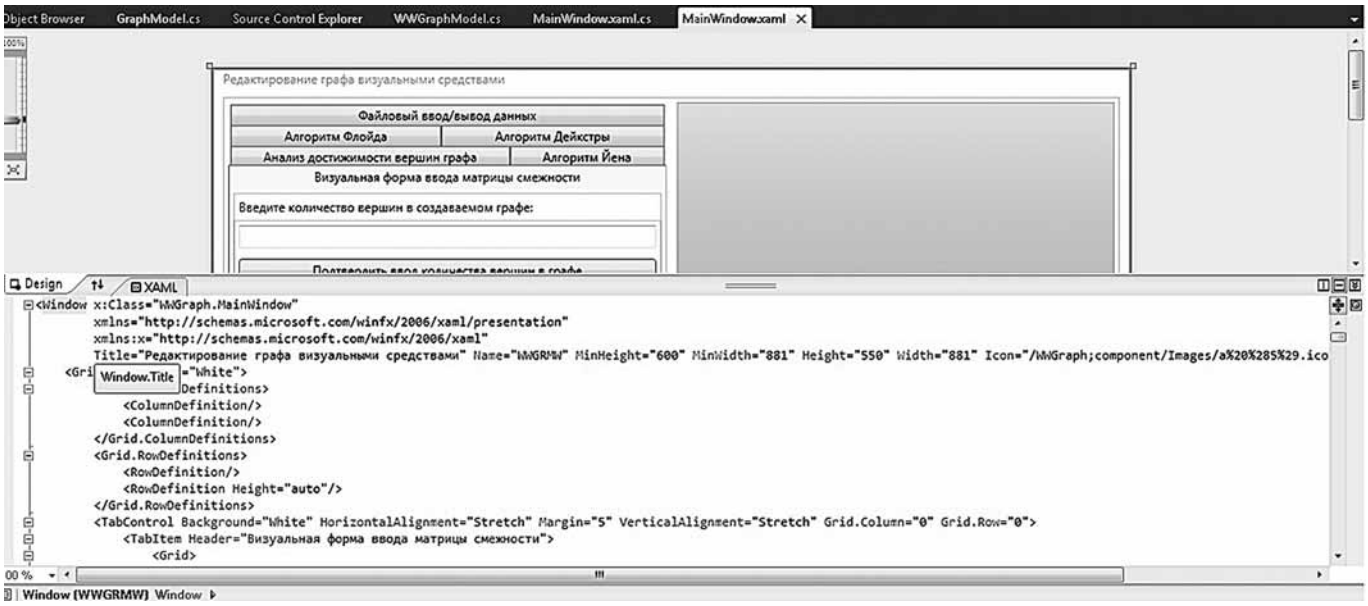


**Рис. 3. Реализация визуальных компонентов программы (XAML-разметка)**

зданию отказоустойчивых и расширяемых приложений. В классе WwGraphModel нас будет интересовать метод, отвечающий за анализ состояния данных программы (листинг 1) и пример реализации отдельно взятой команды модели представления (листинг 2).

Листинг 1:

```
private void UpdateCommandsInfrastucture()
   {
     foreach (var command in Commands)
     {
       KeyValuePair<string, ICommand> kvp;
       if (command is KeyValuePair<string,
         ICommand>)
         kvp = (KeyValuePair<string, ICommand>)
            command;
       else
            continue;
         var dc = kvp.Value as DelegateCommand;
         if (dc == null)
            continue;
         dc.InvokeCanExecuteChanged();
     }
   }
```

Листинг 2:

```
#region LoadWeightsCommand
   public ICommand LoadWeightsCommand
   {
      get { return
      CommandsGetOrCreateCommand(() =>
      LoadWeightsCommand, LoadWeights,
      CanExecuteLoadWeights); }
   }
   private void LoadWeights()
   {
      _myGraph.LoadWeights(FileNameWeight);
      UpdateCommandsInfrastucture();
   }
   private bool CanExecuteLoadWeights()
   {
      if (_myGraph == null)
      {
         return false;
      }
      return (_myGraph.ConMatrix != null)
&& (_myGraph.NumberNodes != 0)&&
(!string.IsNullOrEmpty(FileNameWeight));
      }
      #endregion LoadWeightsCommand
```

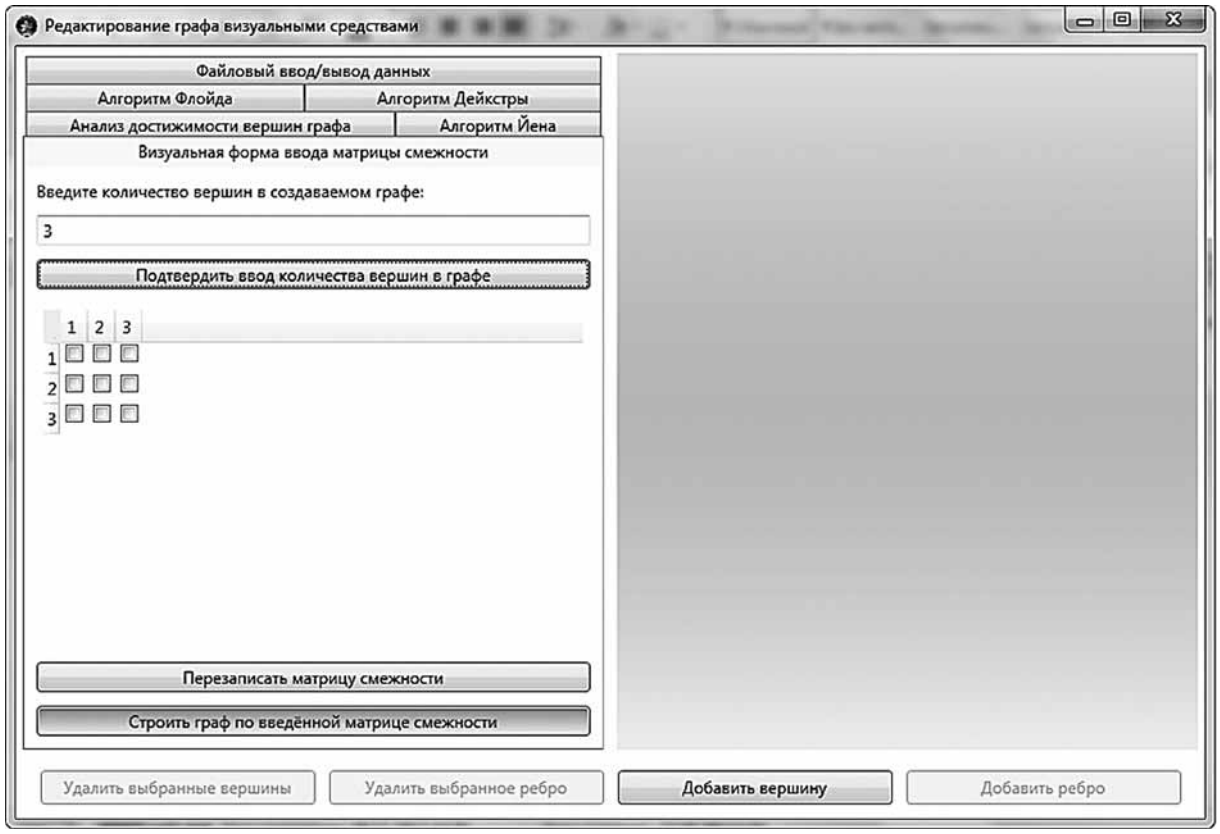


**Рис. 4. Интерфейс программы, автоматизирующей некоторые функции работы с графами**

Как видно из листинга 1, метод UpdateCommandsInfrastucture(), осуществляющий перевод системы в новое состояние, сигнализирует системе команд о том, что ей необходимо проверить возможность выполнения всех команд, содержащихся в модели представления.

В листинге 2 приводится пример описания команды модели представления, состоящего из трех частей: свойства, возвращающего объект класса, реализующего интерфейс ICommand (свойство LoadWeightsCommand); предиката, определяющего возможность выполнения команды (функция CanExecuteLoadWeights()); самой команды непосредственно (функция LoadWeights()).

Результатом взаимодействия описанных выше классов является программа, скриншот которой приведен на рис. 4.

Как видно на рис. 4, не все кнопки на форме активны — это результат ответа системы на состояние данных программы в текущий момент времени. Очевидно, что на данном этапе выполнения программы невозможно добавить ребро графа, удалить выбранные вершины или ребро. Вместе с тем возможно внести изменения во введенную матрицу смежности графа, нарисовать граф или добавить в него вершину. Данная программа реализует еще много функций, однако приводить все скриншоты в данной статье не представляется целесообразным. Тем не менее, требуемое поведение системы получено. Испытание подхода в рамках рассматриваемой предметной области можно считать успешным.

**Список литературы**


1. **Gamma E., Helm R., Johnson R., Vlissides J.** Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley. 1994. 395 p.

2. **Burbeck S.** Applications Programming in Smalltalk-80: How to use Model-View-Controller. URL: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

3. **Potel M.** MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java.URL: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

4. **Fowler M.** Presentation Model. URL: http://martinfowler.com/eaaDev/PresentationModel.html

5. **Туккель Н. И., Шалыто А. А.** SWITCH-технология — автоматный подход к созданию программного обеспечения "реактивных" систем // Программирование. 2001. № 5. С. 45—62.

6. **Бабаев А.** Автоматный интерфейс, новый метод создания логики интерфейса // RSDN Magazine. 2005. № 5. URL: http://rsdn.ru/article/UI/AutoUI.xml

7. **Новиков М.** Паттерн проектирования State // RSDN Magazine. 2005. № 4. URL: http://rsdn.ru/article/patterns/State.xml

8. **Шалыто А. А.** Применение конечных автоматов при программировании мобильных устройств. URL: http://is.ifmo.ru/progeny/mobdev/

9. **Jack S.** WPF MVVM Commands — new reduced-boilerplate recipe. URL: http://blog.functionalfun.net/2010/06/wpf-mvvm-commands-new-reduced.html, 2010.


## *ИНФОРМАЦИЯ*

**26—29 ноября 2013 г.**
**Переславль-Залесский, ИПС имени А. К. Айламазяна РАН**

# Национальный Суперкомпьютерный Форум (НСКФ-2013)

Форум посвящен вопросам создания и практики применения суперкомпьютерных технологий, проводится при поддержке Отделения нанотехнологий и информационных технологий Российской академии наук.

**НСКФ-2013 приглашает к участию:**
- разработчиков суперЭВМ;
- разработчиков системного и прикладного программного обеспечения для суперЭВМ;
- разработчиков грид-технологий и грид-сервисов;
- представителей российских суперкомпьютерных центров;
- аспирантов и молодых ученых, работающих в суперкомпьютерной отрасли;
- потребителей суперкомпьютерных технологий.

**В рамках Национального Суперкомпьютерного Форума проводятся следующие мероприятия:**
- научно-практическая конференция;
- выставка;
- серия мастер-классов и тренингов;
- серия круглых столов;
- пресс-тур.

**Рабочие языки Форума:** русский и английский. Синхронный перевод не обеспечивается.
Мероприятия пресс-тура проводятся на русском языке.

**Подробная информация:** http://www.nscf.ru/

УДК 517.977.1

**К. Ф. Иванова**, канд. техн. наук, докторант, Санкт-Петербургский государственный университет,
e-mail: klara.i2010@yandex.ru

# Чувствительность двойственной интервальной задачи линейного программирования

*Предлагается новая методика оценки чувствительности сопряженных задач линейного программирования с интервальным представлением входной информации. Решение интервальных двойственных задач сводится к решению точечных задач с односторонними пределами погрешностей, определяемых на основе "знаковой" методики, разработанной для систем линейных алгебраических уравнений. Основная идея знаковой методики состоит в процедуре поиска погрешности допустимого решения, исходя из знаков погрешности коэффициентов матрицы и правых частей ограничений. Оценка чувствительности решения задачи линейного программирования проводится параллельно для обеих систем при идентичных относительных погрешностях параметров. Определение оптимальных планов задачи и оптимизация целевой функции осуществляются на основе разработанного алгоритма модельного представления задачи с привлечением симплексного метода, реализованного в системе MATLAB.*

***Ключевые слова:*** *сопряженные интервальные задачи, задача линейного программирования, чувствительность, "знаковая" методика, оценка решения*

## Введение

Методы линейного программирования (ЛП) широко используют в задачах экономики, планирования, оперативного управления и во многих областях народного хозяйства. Решение задачи линейного программирования (ЗЛП) состоит в максимизации (минимизации) целевой функции (ЦФ), которая является линейной функцией от базисных переменных, на которые наложены ограничения в форме неравенств и уравнений.

Анализ чувствительности заключается в выявлении воздействия вариации параметров исходной модели на полученное ранее оптимальное решение ЦФ.

Учет всех факторов при моделировании сложных процессов приводит к серьезному усложнению модели и значительному увеличению ее размера, а пренебрежение большинством даже несущественных факторов — к потере корректности применения полученных результатов. Чем меньше измерений доступно в экономической системе и выше уровень ее неопределенности, тем менее точным является моделирование процесса. В условиях неопределенности, вызванной неточными измерениями и неучтенными факторами воздействия, ошибки моделирования можно компенсировать интервальным заданием параметров. В этом случае точное решение попадает в интервал предельных значений оптимальной ЦФ, гарантируя надежность ее оценки.

Анализ чувствительности ЗЛП, рассматриваемый в литературе [1, 2], как правило, проводится на основании изучения влияния пробных дискретных вариаций входных параметров на возмущение ЦФ [3]. Устойчивость находится из оценки близости возмущенного решения к субоптимальному по заданному $\varepsilon_0$ [4—6].

Предлагаемая в настоящей статье модель определения чувствительности построена на априорной оценке предельных отклонений решения, вызванных интервальными элементами технологической матрицы и правых частей неравенств, рассматриваемых для пары сопряженных задач. Алгоритм расчета ЗЛП, до-

пускающий изменение переменных оптимального плана в пределах наибольших интервальных отклонений, позволяет выявить предел абсолютного или относительного значения погрешности входных параметров, при которых сохраняется структура оптимального плана. Оценка чувствительности двойственных задач основана на "знаковой" методике [7, 8] с привлечением понятий интервальной алгебры [9], позволяющей обосновать и формализовать алгоритм решения. Совместность решения обеих задач проверяется их взаимным откликом на вариативность параметров и соблюдением основных теорем двойственности, устанавливающих связь между оптимальными планами [10].

Постановка и решение интервальных ЗЛП, методы их исследования, к числу которых относятся теоретические разработки условий существования решения интервальных задач, представлены в работах Ю. И. Шокина, С. П. Шарого, А. Н. Тихонова, И. Хансена, И. Рона, Б. Т. Поляка, Д. В. Давыдова и других авторов. В исследованиях проводится математическая оценка границ решения оптимальной задачи, которая в ряде случаев сводится к решению ЗЛП. Однако во всех методиках оценок влияния интервальности матриц или правых частей уравнений на решение задачи, сопутствующим неудобством является возникновение систем больших размерностей, что приводит к чрезмерным вычислительным трудностям.

## Интервальная модель двойственной задачи линейного программирования

Применение к оценке решения интервальной задачи линейного программирования (ИЗЛП) параметрического подхода, в рамках которого предполагается вариация интервальных входных параметров в заданных границах, допустимый разброс (интервал) значений ЦФ, ограничивающих точное значение решения, позволяет описать все возможные реализации оптимума.

В целях удобства изложения введены следующие замечания и обозначения:

- полужирным шрифтом размечены интервальные матрицы, векторы и их компоненты;
- прямым шрифтом обозначены все интервальные и вещественные матрицы;
- компоненты векторов и матриц обоих видов размечены курсивом;
- IR — множество интервальных чисел;
- $\mathbf{A} \in \mathrm{IR}^{m \times n}$ — интервальная матрица, имеющая $m$ строк, $n$ столбцов; $\mathrm{IR}^{m \times n}$ — интервальное вещественное пространство размерности $m \times n$; $\mathbf{b} \in \mathrm{IR}^{m}$ — интервальный вектор с $m$ компонентами;
- границы интервальных величин в общем случае снабжены нижним и верхним подчеркиваниями и заключены в квадратные скобки;
- границы интервалов векторов и матриц, полученные по "знаковой" методике, снабжены символами (+) и (−).

Наиболее распространенным типом ЗЛП является задача максимизации суммарной прибыли путем выпуска определенной номенклатуры продукции при ограниченных ресурсах [11]. Количество ресурсов $m$ обозначается вектором $\mathbf{b} = (\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_m)$. Технологические коэффициенты $\boldsymbol{a}_{ij}$ показывают норму расхода $i$-го ресурса на производство единицы $j$-й продукции. Эффективность выпуска единицы $j$-й продукции определяется прибылью $\boldsymbol{c}_j$, стоимостная оценка единицы ресурса — $\boldsymbol{y}_i$. Решение исходной задачи состоит в определении оптимального плана выпуска продукции $x = (x_1, x_2, ..., x_n)$, доставляющего максимум целевой функцией $F$ или минимум функции $Z$, выраженной стоимостной ценой расхода ресурсов $y = (y_1, y_2, ..., y_m)$.

Интервальная математическая модель задается параметрами, к числу которых относятся интервалы значений коэффициентов технологической матрицы и вектора правых частей, определяющие погрешность входных данных. Теорема двойственности в условиях интервального представления параметров, как и при их точечных значениях, устанавливает связь между оптимальными планами обеих сопряженных задач и критериев эффективности $F$ и $Z$. Целью исследования задачи в такой постановке является анализ влияния интервальных коэффициентов технологической матрицы и компонент вектора ограничений-неравенств на предельные границы оптимума ЦФ. Известно, что в точечных ЗЛП анализ вариаций объема ресурсов и стоимостных оценок определенного знака (в дальнейшем будем считать постоянного) может быть проведен заранее в целях определения их влияния на максимальное значение ЦФ. В то же время наиболее типичным видом вариации входных данных является их знаковая неопределенность. Использование "знаковой" методики в интервальном представлении коэффициентов технологической матрицы позволяет выбрать те из границ интервалов входных параметров, которые обеспечивают предельные отклонения оптимального базисного плана. Соответственно, оптимизируемая ЦФ реально получает разброс, который выражается наименьшим и наибольшим значениями, вычисленными по точечным значениям двух граничных оптимальных планов.

В интервальном виде постановка пары симметричных двойственных задач может быть представлена следующим образом:

$$\max(\min) Z = \sum_{j=1}^{n} c_j x_j,$$

$$\begin{cases} \sum\limits_{j=1}^{n} \boldsymbol{a}_{ij} x_j \leqslant \boldsymbol{b}_i, \\ x_j \geqslant 0, \; j = \overline{1, n}, \; i = \overline{1, m}, \end{cases} \qquad (1)$$

$$\min(\max) F = \sum_{j=1}^{m} b_i y_i,$$

$$\begin{cases} \sum\limits_{j=1}^{n} \boldsymbol{a}_{ij} y_i \geqslant \boldsymbol{c}_i,\ y_i \geqslant 0,\ j = \overline{1,\ n},\ i = \overline{1,\ m} \end{cases} \qquad (2)$$

с интервальной матрицей $\mathbf{A} \in \mathrm{IR}^{m \times n}$, $\mathbf{A} = \{\boldsymbol{a}_{ij}\}$, $i = \overline{1,\ m}$, $j = \overline{1,\ n}$, и интервальными правыми частями в исходной задаче $\mathbf{b} \in \mathrm{IR}^m$, $\mathbf{b} = \{\boldsymbol{b}_i\}$, $i = \overline{1,\ m}$ (1) и $\mathbf{c} \in \mathrm{IR}^n$, $\mathbf{c} = \{\boldsymbol{c}_j\}$, $j = \overline{1,\ n}$ — в двойственной задаче (2).

Допустимым оптимальным решением (базисом) сопряженных задач называют векторы $\mathrm{x} = \{x_j\}$, $j = \overline{1,\ n}$ и $\mathrm{y} = \{y_i\}$, $i = \overline{1,\ m}$, каждый из которых удовлетворяет соответствующей системе неравенств и реализующих экстремум ЦФ симметричных двойственных задач. По теореме двойственности $\max Z(x) = \min F(x)$ для невозмущенного варианта. Существенным допущением рассматриваемых оптимальных задач является условие попадания допустимых решений ИЗЛП в первый пространственный ортант положительных значений векторов x и y. Предлагаемый подход к оценке решения ИСЛАУ состоит в формализации интервальной системы как структуры точечных линейных систем с матрицами $\mathrm{A} \in \mathbf{A}$ и векторами $\mathrm{b} \in \mathbf{b}$ исходной задачи, с сопряженными матрицами $\mathrm{A}^{\mathrm{T}} \in \mathbf{A}^{\mathrm{T}}$ и правыми частями $\mathrm{c} \in \mathbf{c}$ — двойственной.

С позиции интервальной алгебры задача об отыскании внутреннего интервала для допускового множества решений известна как интервальная линейная задача о допусках [12]. Допусковые множества решений (*tol of tolerant*, допустимый)

$$\boldsymbol{\Sigma}_{tol} = \{\mathrm{x} \in \mathrm{R}^n | (\forall \mathrm{A} \in \mathbf{A})(\exists \mathrm{b} \in \mathbf{b})(\mathrm{Ax} = b)\}, \qquad (3)$$

$$\Sigma\forall'_{tol} = \{\mathrm{y} \in \mathrm{R}^m | (\forall \mathrm{A}' \in \mathbf{A}')(\exists \mathrm{c} \in \mathbf{c})(\mathrm{A}'\mathrm{y} = \mathrm{c})\} \qquad (4)$$

содержат всевозможные векторы x и y такие, что для всяких точечных матриц коэффициенты $\mathrm{A} \in \mathbf{A}$ и $\mathrm{A}^{\mathrm{T}} \in \mathbf{A}^{\mathrm{T}}$ преобразования $Ax$ и $\mathrm{A}^{\mathrm{T}}\mathrm{y}$ не выходят за границы интервальных векторов $\mathbf{b}$ ($\mathrm{Ax} = \mathbf{b}$, $\mathrm{A} \in \mathbf{A}$) и $\mathbf{c}$ ($\mathrm{A}^{\mathrm{T}}\mathrm{y} = \mathbf{c}$, $\mathrm{A}^{\mathrm{T}} \in \mathbf{A}$). Предельные допустимые точечные значения компонент вектора неизвестных $\mathrm{x} = \{x_j\}$ системы (1) и $\mathrm{y} = \{y_j\}$ системы (2) из множества решений определяют оценку оптимизируемых линейных функций.

Как и в точечном варианте, при решении исходной и двойственной интервальных задач линейного программирования, их приводят к каноническому виду с ограничениями в виде равенств и с положительными переменными.

## Интервальное представление двойственной интервальной задачи с позиции "знаковой" методики

Решение ИЗЛП сводится к решению двух точечных систем (см. ниже (5) и (6)) для каждой из сопряженных задач (1) и (2) выбором определенной границы каждого из интервалов коэффициентов матриц и векторов. Этот выбор осуществляется на основании "знаковой" методики и приводит к решению двух пар точечных систем ЗЛП:

$$\begin{cases} F1(x) = \mathrm{cx}^+ \to \max, \\ \mathrm{A}^- \boldsymbol{x}^+ \leqslant \mathrm{b}^+ \\ F2(x) = \mathrm{cx}^- \to \max, \\ \mathrm{A}^+ \boldsymbol{x}^- \geqslant \mathrm{b}^-, \\ \mathrm{b}^- = (b_1^-, b_2^-, ..., b_n^-)^{\mathrm{T}},\ \mathrm{b}^+ = (b_1^+, b_2^+, ..., b_n^+)^{\mathrm{T}}, \\ \mathbf{A} = [\mathrm{A}^-, \mathrm{A}^+],\ \mathbf{b} = [\mathrm{b}^-, \mathrm{b}^+],\ \mathrm{A}^+ \in \mathbf{A},\ \mathrm{A}^- \in \mathbf{A}. \end{cases} \qquad (5)$$

$$\begin{cases} Z1(y) = \mathrm{by}^+ \to \min, \\ \mathrm{A}^{\mathrm{T}-} \boldsymbol{y}^+ \leqslant \mathrm{c}^+ \\ Z2(y) = \mathrm{by}^- \to \min, \\ \mathrm{A}^{\mathrm{T}+} \boldsymbol{y}^- \leqslant \mathrm{c}^-, \\ c^- = (c_1^-, c_2^-, ..., c_n^-)^{\mathrm{T}},\ \mathrm{c}^+ = (c_1^+, c_2^+, ..., b_n^+)^{\mathrm{T}}, \\ \mathbf{A}^{\mathrm{T}} = [\mathrm{A}^{\mathrm{T}-}, \mathrm{A}^{\mathrm{T}+}],\ \mathbf{c} = [\mathrm{c}^-, \mathrm{c}^+],\ \mathrm{A}^{\mathrm{T}+} \in \mathbf{A}^{\mathrm{T}},\ \mathrm{A}^{\mathrm{T}-} \in \mathbf{A}^{\mathrm{T}}. \end{cases} \qquad (6)$$

Запись целевых функций с индексами 1 и 2 определяют ЦФ двух точечных задач с выбором определенной интервальной границы матриц и векторов $\mathbf{A}$, $\mathbf{A}^{\mathrm{T}}$, $\mathbf{b}$ и $\mathbf{c}$. Матрицы и векторы с верхними символами (+) и (−) конструируются формированием левой или правой границы интервалов, входящих в выражения (5), (6) по принципу возрастания определителя в положительном и отрицательном направлениях матрицы A и векторов b и c Матрицы $\mathrm{A}^-$ и $\mathrm{A}^+$ и компоненты векторов $\mathrm{b}^-$ и $\mathrm{b}^+$, $\mathrm{c}^-$ и $\mathrm{c}^+$ выбраны симметричными относительно их средних значений A, b и c в интервалах $[\mathrm{A}^-, \mathrm{A}^+]$, $[\mathrm{b}^-, \mathrm{b}^+]$ и $[\mathrm{c}^-, \mathrm{c}^+]$: $\mathrm{A}^- = \mathrm{A} - \Delta$, $\mathrm{A}^+ = \mathrm{A} + \Delta$, где $\Delta = \varepsilon\mathrm{A}$ — абсолютная погрешность, равная произведению относительной погрешности на среднее значение матрицы; $\mathrm{b}^+ = \mathrm{b} + \Delta 1$, $\mathrm{b}^- = \mathrm{b} - \Delta 1$ или $\mathrm{c}^+ = \mathrm{c} - \Delta 2$, $\mathrm{c}^- = \mathrm{c} - \Delta 2$, где $\Delta 1 = \delta\mathrm{b}$, $\Delta 2 = \delta\mathrm{c}$ — для векторов правых частей исходной и двойственной задач соответственно.

Тогда элементы матриц $\mathbf{A}$ и $\mathbf{A}^{\mathrm{T}}$ и компоненты векторов $\mathbf{b}$ и $\mathbf{c}$ записываются в интервальном виде следующим образом:

$$\boldsymbol{a}_{ij} = [a_{ij} - \Delta,\ a_{ij} + \Delta],\ i = \overline{1,\ m},\ j = \overline{1,\ n},$$

$$\boldsymbol{b}_i = [b_i - \Delta 1,\ b_i + \Delta 1],\ i = \overline{1,\ m},$$

$$\boldsymbol{c}_i = [c_i - \Delta 2,\ c_i + \Delta 2],\ i = \overline{1,\ m}$$

(для двойственной задачи).

Такой подход позволил полностью упростить процедуру решения NP-сложной ИЗЛП, заменив ее решением каждой из двух двойственных задач двумя точечными алгебраическими системами (5) и (6).

## Анализ решения задачи линейного программирования с помощью теории двойственности

Представленный подход позволяет сопоставить вариативность решений для ЦФ симметричных двойственных задач и проследить, как происходит параллельное изменение возмущенных векторов x и y в этом случае. При постановке ЗЛП приняты следующие допущения: линейные системы не вырожденные; число уравнений в системе ограничений задачи меньше числа входящих в них неизвестных, т. е. $m < n$; среди бесконечного множества решений можно выбрать оптимальное, доставляющее максимум целевой функции и такое, что в случае существования решения для одной из задач, оно существует и для двойственной.

Формализуя экономическую задачу можно говорить о выпуске продукции предприятием с ограниченным запасом ресурсов $m$, количество которого определяется вектором ресурсов $b = (b_1, b_2, ..., b_m)$ и количеством ассортимента продукции $n$, выраженным вектором $x = (x_1, x_2, ..., x_n)$. Технологические коэффициенты $\boldsymbol{a}_{ij}$ показывают норму расхода $i$-го ресурса на производство единицы $j$-й продукции. Эффективность выпуска единицы $j$-й продукции характеризуется прибылью $c_j$. Требуется определить план выпуска продукции x*, максимизирующий прибыль предприятия $\max F(x)$ при заданных ресурсах b.

Экономическое содержание сопряженных задач состоит в совместных оценках максимального выпуска продукции (5) и определения оценок ресурсов (6) при условии разрешимости обеих задач. План производства x и вектор оценок ресурсов y являются оптимальными тогда и только тогда, когда цена произведенной продукции $\boldsymbol{cx}$ и суммарная оценка ресурсов $\boldsymbol{by}$ совпадают. Оценки выступают как инструмент балансирования затрат и результатов и гарантируют рентабельность оптимального плана. В противном случае, продавать сырье не имеет смысла — целесообразнее изготовить товар и получить прибыль от его реализации. Отсюда следует основное положение теоремы двойственности относительно совпадения целевых функций на оптимальных решениях: $F1(x) = Z1(y)$; $F2(x) = Z2(y)$.

Теорема двойственности устанавливает связь между оптимальными планами пары двойственных задач, на основании которой должны выполняться оценки влияния свободных членов $b_j$ системы ограничений прямой задачи на значение $\Delta F(x) : \Delta F(x) = \Delta b_j \boldsymbol{y}_j$.

Предлагаемый подход к решению задачи позволяет найти знаки вариации относительных отклонений ε коэффициентов $\boldsymbol{a}_{ij}$ и погрешностей δ правых частей $\mathbf{b} = (\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_m)$ уравнения (5) исходной системы, которые приводят к максимальным вариациям ранее определенной ЦФ $F(x)$ и не нарушают структурного оптимального плана.

Одновременно в двойственной задаче выбор знака относительных погрешностей коэффициентов транспонированной матрицы $\boldsymbol{a}_{ij}$ и погрешности δ вектора $\mathbf{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_m)$ стоимостных оценок единицы продукции каждой номенклатуры определяет границы минимизируемой сопряженной функции $Z(y)$.

При выполнении условий (3) и (4) можно и не получить предельно максимальных отклонений для каждой компоненты продукции. Однако близость к оптимальному доходу следует из самой постановки задачи, ориентированной на выявление наибольшего диапазона отклонений в положительном $x_{\max}$ и отрицательном $x_{\min}$ направлениях максимально возможного числа компонент переменной $x$, определяющих диапазон интервальности ЦФ при вариациях переменных.

При таком подходе к анализу чувствительности решения к погрешностям внешних воздействий может быть проконтролирована и устойчивость оптимального решения, когда размер интервальных погрешностей Δ, Δ1 и Δ2 не приводит к отклонению возмущенной ЦФ от оптимального точечного решения на величину, бóльшую заданного значения $\varepsilon_0$.

## Примеры решения двойственных интервальных задач линейного программирования

Проиллюстрируем применение представленной выше методики для решения ИЗЛП.

**Пример 1.**

Рассмотрим план производства изделий двух видов. Математическая модель описывается канонической системой размерности 5 × 2 из пяти уравнений ($m = 5$) с двумя переменными $x_1$, $x_2$ [13]. Интервальный вариант прямой задачи заключается в определении максимума линейной функции

$$F(x) = x_1 + 2x_2 \to \max, \tag{7}$$

с ограничениями—неравенствами

$$\begin{cases} \boldsymbol{a}_{11}x_1 + \boldsymbol{a}_{12}x_2 <= \boldsymbol{b}_1 \\ \boldsymbol{a}_{21}x_1 + \boldsymbol{a}_{22}x_2 <= \boldsymbol{b}_2 \\ \boldsymbol{a}_{31}x_1 + \boldsymbol{a}_{32}x_2 <= \boldsymbol{b}_3 \\ \boldsymbol{a}_{41}x_1 + \boldsymbol{a}_{42}x_2 >= \boldsymbol{b}_4 \\ \boldsymbol{a}_{51}x_1 + \boldsymbol{a}_{52}x_2 >= \boldsymbol{b}_5, \end{cases} \tag{8}$$

где коэффициенты $\boldsymbol{a}_{ij}$ и векторы правой части $\boldsymbol{b}_i$ системы (7)—(8)

$$\boldsymbol{a}_{ij} = [a_{ij} - \varepsilon a_{ij},\ a_{ij} + \varepsilon a_{ij}]$$

$$\boldsymbol{b}_i = [b_i - \delta b_i,\ b_i + \delta b_i],\ i = \overline{1, 5},\ j = \overline{1, 2}$$

$$\begin{cases} a_{11} = -1,\ a_{12} = 3,\ a_{21} = 1,\ a_{22} = 1,\ a_{31} = 1, \\ a_{32} = -1,\ a_{41} = 1,\ a_{42} = 3,\ a_{51} = 2,\ a_{52} = 1, \\ b_1 = 10,\ b_2 = 6,\ b_3 = 2,\ b_4 = 6,\ b_5 = 4, \end{cases}$$

$$x_1,\ x_2 \geqslant 0,$$

$$\varepsilon <= |\varepsilon_{ij}|,\ \delta <= |\delta_i|.$$

Каноническая система получается введением дополнительных переменных $x_3$, $x_4$, $x_5$, $x_6$, $x_7$ и двух искусственных переменных $x_8$, $x_9$ и записывается как

$$F(x) = c_1x_1 + c_22x_2 + wx_8 + wx_9 \to \max; \qquad (9)$$

$$\begin{cases} \boldsymbol{a}_{11}x_1 + \boldsymbol{a}_{12}x_2 + x_3 = \mathbf{b}_1 \\ \boldsymbol{a}_{21}x_1 + \boldsymbol{a}_{22}x_2 + x_4 = \mathbf{b}_2 \\ \boldsymbol{a}_{31}x_1 + \boldsymbol{a}_{32}x_2 + x_5 = \mathbf{b}_3 \\ \boldsymbol{a}_{41}x_1 + \boldsymbol{a}_{42}x_2 - x_6 + x_8 = \mathbf{b}_4 \\ \boldsymbol{a}_{51}x_1 + \boldsymbol{a}_{52}x_2 - x_7 + x_9 = \mathbf{b}_5. \end{cases} \qquad (10)$$

Новая матрица канонизированной системы имеет размерность $5 \times 9$. В выражение для оптимизируемой функции (9) вводятся переменные $w$, представляющие бесконечную искусственную стоимость. Решение точечной задачи симплекс-методом обеспечивает максимальное значение функции цели $F_{\text{ср}} = 10$ при оптимальном допустимом значении базисного вектора с компонентами $\text{x}_{\max} = (2, 4, 0, 0, 4, 8, 4, 0, 0)$ и двумя первыми эффективными компонентами $x_1$, $x_2$. На базе знаковой методики сконструированы две системы с матрицами $\text{A}^-$ и $\text{A}^+$ и векторами $\text{b}^-$ и $\text{b}^+$ с интервальными коэффициентами для исходных переменных $x_1$, $x_2$ с относительными погрешностями. Такие системы позволяют определить два граничных оптимальных плана

$$\begin{cases} \text{x}_{\max} = (3{,}0000,\ 5{,}2500,\ 0,\ 0,\ 5{,}9000,\ 9{,}0000,\ 5{,}0000) \\ \text{x}_{\min} = (1{,}1816,\ 3{,}3182,\ 0,\ 0,\ 3{,}2364,\ 7{,}3636,\ 2{,}8183) \end{cases}$$

и вычислить соответствующий диапазон изменения для максимизируемой функций цели: $\boldsymbol{F}_{\min} < \boldsymbol{F}_{\text{ср}} < \boldsymbol{F}_{\max} \to$ $\to 7{,}82 < 10 < 13{,}50$.

Соответствующая двойственная задача в канонизированном виде представляется следующим образом:

$$\min Z(y) = b_1y_1 + b_2y_2 + b_3y_3 + b_4y_4 + b_5y_5 + 0y_6 + 0y_7 \to \min; \qquad (11)$$

$$\begin{cases} \boldsymbol{a}_{11}y_1 + \boldsymbol{a}_{21}y_2 + \boldsymbol{a}_{31}y_3 + \boldsymbol{a}_{41}y_4 + \boldsymbol{a}_{51}y_5 - y_6 = \mathbf{c}_1 \\ \boldsymbol{a}_{12}x_1 + \boldsymbol{a}_{22}x_2 + \boldsymbol{a}_{32}y_3 + \boldsymbol{a}_{42}y_4 + \boldsymbol{a}_{52}y_5 - y_7 = \mathbf{c}_2 \end{cases} \qquad (12)$$

$$y_1,\ y_2 \geqslant 0.$$

Канонизированная система имеет размерность $2 \times 7$. Решение точечной задачи симплекс-методом обеспечивает максимальное значение функции цели $F_{\text{ср}} = 10$, равное оптимальному значению ЦФ исходной задачи. Значение оптимального плана двойственной задачи получено равным $\text{y} = (0{,}25,\ 1{,}25,\ 0,\ 0,\ 0,\ 0,\ 0)$ с двумя первыми эффективными компонентами $y_1$, $y_2$. На базе знаковой методики сконструированы две системы с матрицами $\text{A}^{\text{T}-}$ и $\text{A}^{\text{T}+}$ с векторами $\text{c}^-$ и $\text{c}^+$, с интервальными коэффициентами для двойственных переменных $y_1$, $y_2$. На основании этих систем определяются два граничных оптимальных плана

$$\begin{cases} \text{y}_{\max} = (0{,}3611,\ 1{,}6667,\ 0,\ 0,\ 0,\ 0,\ 0) \\ \text{y}_{\min} = (0{,}1591,\ 1{,}0227,\ 0,\ 0,\ 0,\ 0,\ 0) \end{cases}$$

с интервальным диапазоном для ЦФ, равным $\boldsymbol{F}_{\min} < \boldsymbol{F}_{\text{ср}} < \boldsymbol{F}_{\max} \to 7{,}73 < 10 < 13{,}61$. Различие между оптимальными целевыми функциями исходной и двойственной задач составляют десятые доли процента. Эти отклонения объясняются исключительно разной размерностью основной и сопряженной систем, а именно числом столбцов матрицы и строк вектора правой части, компоненты которых подвергаются знаковый выборке границы интервала. Интервальное представление имеют только коэффициенты при эффективных переменных, число которых отличается от числа уравнений $m$. Так, в рассматриваемом примере в системе (8) присутствуют две эффективные переменные $x_1$, $x_2$ в пяти уравнениях. Однако в этом случае в интервальном представлении различаются компоненты правых частей $\boldsymbol{b}$ в прямой и $\boldsymbol{c}$ в двойственной задачах.

Между исходными и двойственными переменными соблюдаются соотношения, подчиненные теореме о нежесткости [3], аналогичные точечной двойственной задаче. Так, эффективным переменным $x_1$, $x_2$, вошедшим в оптимальный план, соответствуют дополнительные переменные $y_6$, $y_7$ двойственной задачи, равные нулю. Нулевым дополнительным переменным исходной задачи $x_3$, $x_4$ соответствуют ненулевые двойственные переменные $y_1$, $y_2$. Нулевым дополнительным переменным $x_5$, $x_6$, $x_7$, не равным нулям, соответствуют нулевые дополнительные переменные двойственной задачи $y_5$, $y_6$, $y_7$, равные нулю. В точечной задаче при увеличении ресурса первого продукта $b_1$ на 1, ЦФ увеличится на 0,25 для первой переменной $x_1 = 2$. Для граничных значений диапазона интервальной задачи эти значения составляют $y_1^- = 0{,}1591$ и $y_1^+ = 0{,}3611$ для переменных $x_1^- = 1{,}1816$ и $x_1^+ = 3{,}0000$ соответственно. Из сравнения этих показаний следует, что ЦФ может больше увеличиться при возрастании объема продукции $b_1$ на 1, чем понизиться при уменьшении $b_1$ на 1. Та же картина наблюдается и для второго продукта $b_2$. Равенство нулю дополнительных переменных исходной задачи $x_3$, $x_4$ характеризует полностью реализованный спрос.

На рис. 1, 2 слева представлены решения для базисов оптимальных планов точечной $x_{\text{ср}}$ (обозначенные $x_{\text{opt}}$) и исходной интервальной ($x_{\min}$ и $x_{\max}$) задач. Справа показаны расчеты оптимальной точечной функции $F_{\text{цели}}$ и граничные значения интервальной ($F_{\min}$ и $F_{\max}$). Ниже показаны абсолютные и относительные отклонения граничных значений оптимальных ЦФ от ее точечного значения. Из сравнения полученных результатов следует, что относительная

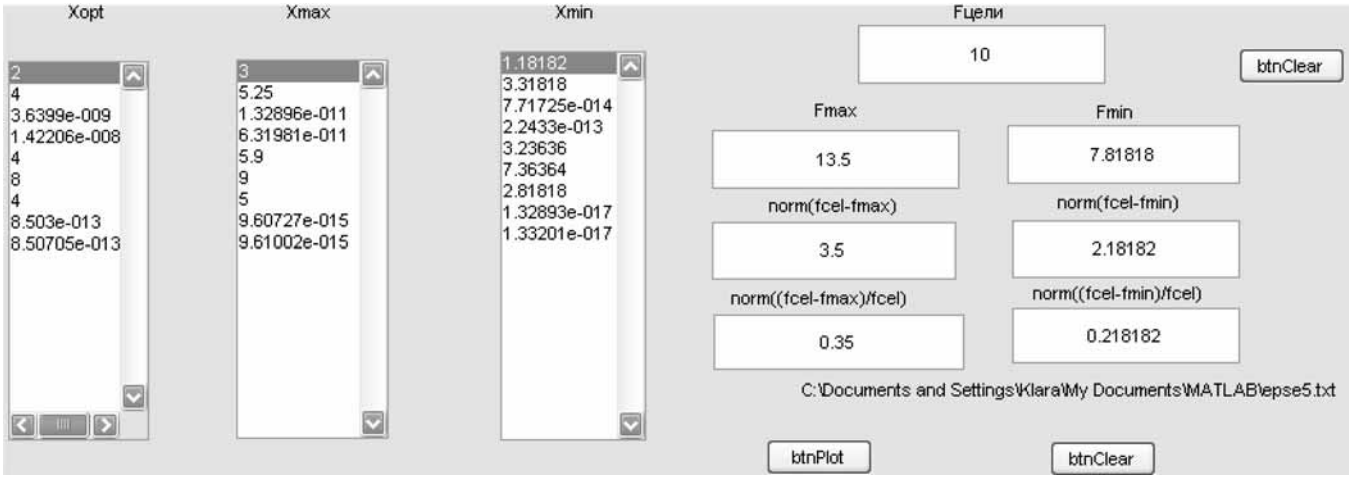


**Рис. 1. Расчет оптимальных планов и целевых функций с оценками абсолютных и относительных погрешностей ЦФ для прямой задачи ($\varepsilon = 0{,}2$; $\delta = 0{,}1$)**

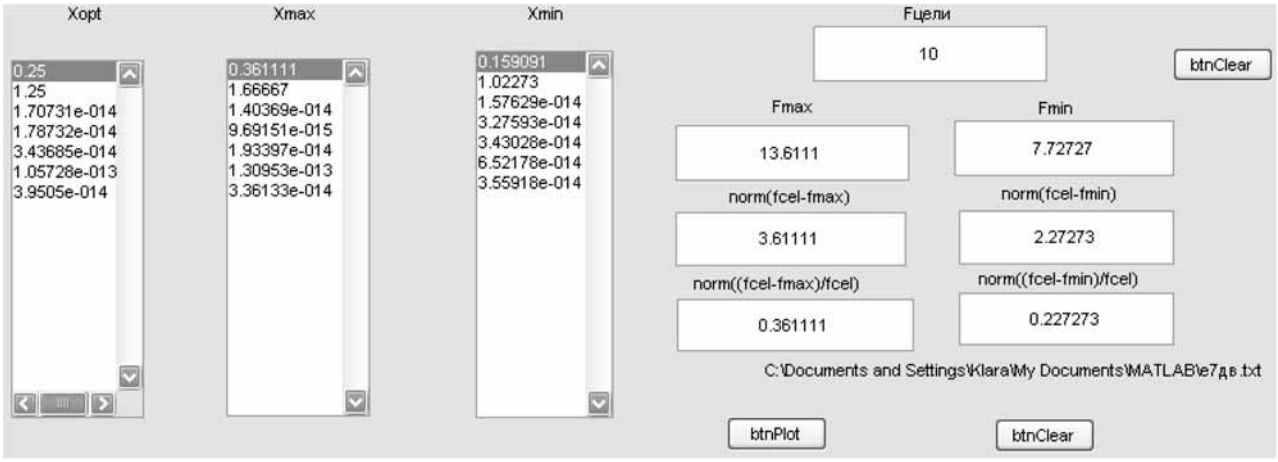


**Рис. 2. Расчет оптимальных планов и целевых функций с оценками абсолютных и относительных погрешностей ЦФ для двойственной задачи ($\varepsilon = 0{,}2$; $\delta = 0{,}1$)**

погрешность предельных отклонений от точечного значения достигает 21...22 % и 35...36 %. При этом структура оптимальных интервальных планов в соответствии с результатами решения точечной задачи остается прежней.

На рис. 3 (см. вторую сторону обложки) показаны области допустимых значений в переменных $x_1$, $x_2$ и $y_1$, $y_2$ и точки оптимальных планов, обозначенные кружками, включая оптимальный план для точечного решения, как точки пересечения ограничений-равенств (8) с линией уровня (7) для случая погрешностей коэффициентов матрицы $\boldsymbol{a}_{ij}$, равных $\varepsilon = 0{,}2$ и компонент вектора правых частей $\boldsymbol{b}_i$, равных $\delta = 0{,}1$. На рис. 4 (см. вторую сторону обложки) погрешности коэффициентов матрицы $\boldsymbol{a}_{ij}$ приняты равными нулю, погрешности компонент $\boldsymbol{b}_i$ — равными прежнему значению $\delta = 0{,}1$. Сравнение графиков на обоих рисунках демонстрирут заметное сужение допустимой области решения обеих сопряженных задач, определяющих граничные значения оптимальных целевых функций $F(x)$ и $Z(y)$.

На рис. 5 (см. вторую сторону обложки) показаны графики зависимостей компонент оптимальных планов от номера переменной для точечной и интервальной систем с постоянными знаками погрешностей и знакопеременными погрешностями для исходной и двойственной задач. Вектор базиса переменных, полученного для точечной задачи — $x_{\text{ср}}$, для интервального варианта при постоянных погрешностях коэффициентов и правых частей — $\bar{x}_{\min}$, $\bar{x}_{\max}$. Предельные отклонения переменных оптимального плана, полученные при выборе знаковых границ погрешности входных параметров — $x_{\min}$, $x_{\max}$. Для двойственной задачи приняты обозначения, аналогичные прямой задаче: $y_{\text{ср}}$ — оптимальный план точечной задачи, $\bar{y}_{\min}$, $\bar{y}_{\max}$ — базисный план для задачи с постоянными по знаку отклонениями входных параметров, $y_{\min}$, $y_{\max}$ — оптимальный план со знаковыми погрешностями входных параметров.

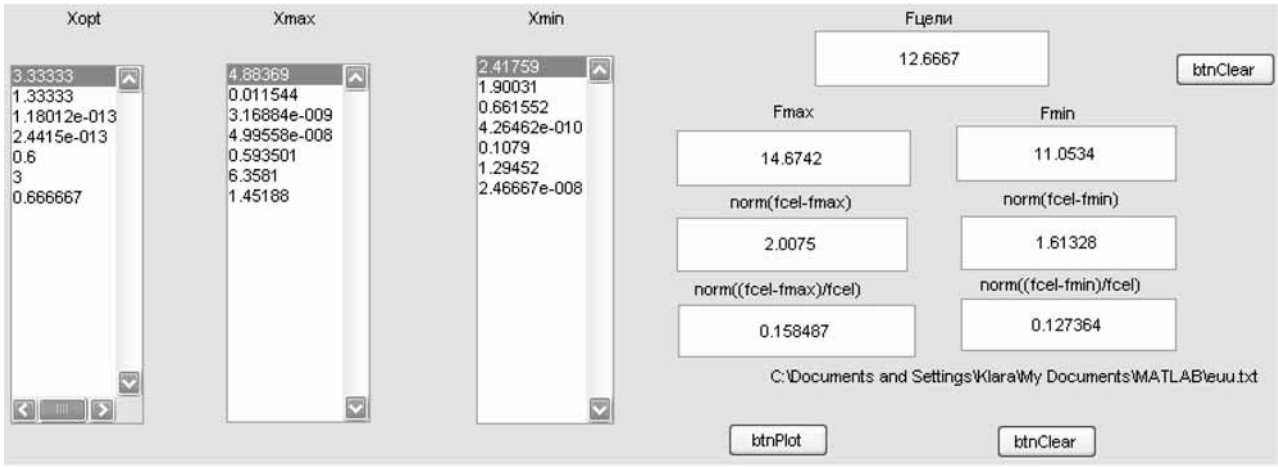


**Рис. 7. Расчет оптимальных планов точечной и интервальной исходных задач (ε = 0,1; δ = 0,1)**

На графиках рис. 6 (см. третью сторону обложки) показаны сравнительные интервальные значения оптимальных ЦФ для исходной и двойственной задач в зависимости от первой компоненты оптимальных планов.

Рассмотренный пример двойственной задачи оказался сравнительно устойчивым к большим погрешностям параметров. На следующем примере просматривается более заметная чувствительность решения к исходным данным.

**Пример 2.**

Рассмотрим план производства изделий двух видов. Математическая модель описывается канонической системой размерности 5 × 7 из пяти уравнений ($m = 5$) с двумя эффективными переменными $x_1$, $x_2$ и пятью дополнительными переменными $x_3$, $x_4$, $x_5$, $x_6$, $x_7$. Система представляется в каноническом виде со средними значениями коэффициентов. Интервальный вариант прямой задачи заключается в определении максимума линейной функции с ограничениями-равенствами

$$F(x) = 3x_1 + 2x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 \to \max;$$

$$\begin{cases} x_1 + 2x_2 + x_3 = 6 \\ 2x_1 + x_2 + x_4 = 8 \\ x_1 + 0{,}8x_2 + x_5 = 5 \\ -x_1 + x_2 + x_6 = 1 \\ x_2 + x_7 = 2 \end{cases}$$

$$x_j >= 0,\ j = \overline{1, 7}.$$

Двойственная система в каноническом виде:

$$Z(y) = 6y_1 + 8y_2 + 5y_3 + y_4 + 2y_5 + 0y_6 + 0y_7 \to \min;$$

$$\begin{cases} y_1 + 2y_2 + y_3 - y_4 - y_6 = 3 \\ 2y_1 + y_2 + y_4 + y_5 - y_7 = 2 \end{cases}$$

$$y_i >= 0,\ i = \overline{1, 7}.$$

На рис. 7 показана часть листинга решения задачи (11)—(12) при ε = 0,1, δ = 0,1.

На рис. 8 (см. третью сторону обложки) показана зависимость переменных оптимальных планов от номера компоненты $x$. Как следует из расчетов, проиллюстрированных на рис. 7 (третья компонента в столбце $x_{\min}$), и графиков на рис. 8, компонента $x_3$ оптимального плана левой границы интервала не обнуляется по сравнению с точечным вариантом. Наоборот, $x_2$ становится нулевой (вторая компонента в столбце $x_{\max}$), из чего следует вывод о нарушении конструкции базисного плана при заданных относительных погрешностях коэффициентов матрицы и правых частей ε = 0,1, δ = 0,1. Следует отметить, что нарушение конструкции базисного плана инициировано именно знакопеременными погрешностями. Это обстоятельство подтверждает актуальность знакового подхода к исследованию чувствительности решения оптимальной задачи (11)—(12).

При уменьшении погрешности ε до 0,08 и δ = 0 компонента $x_3$ уменьшается, что отображено на рис. 9, см. третью сторону обложки.

При приближении ε к 0,03 при δ = 0 переменная $x_3$ значительно уменьшается и устремляется к 0 (рис. 10, см. третью сторону обложки), и конструкция базисного плана восстанавливается.

**Пример 3.**

Рассмотрим план x = ($x_1$, $x_2$, ..., $x_3$) производства изделий трех видов, $x_4$, $x_5$, $x_6$, $x_7$ — дополнительные переменные. Математическая модель в каноническом виде для средних значений коэффициентов записывается следующим образом:

$$F(x) = 3x_1 + 4x_2 + 2x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 \to \max; \quad (13)$$

$$\begin{cases} x_1 + 2x_2 + x_3 + x_4 = 18 \\ 2x_1 + x_2 + x_5 = 16 \\ x_1 + x_2 + x_6 = 8 \\ x_2 + x_3 + x_7 = 6 \end{cases} \quad (14)$$

$$x_j >= 0,\ j = \overline{1, 7}.$$

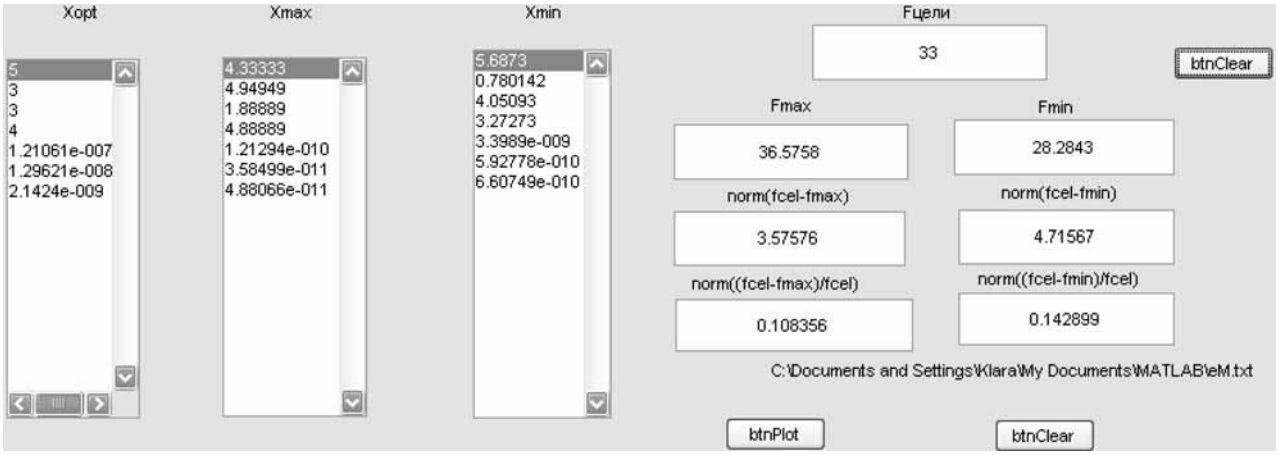


**Рис. 11. Расчет оптимальных точечного и интервального планов и ЦФ исходной задачи при условии ε = 0,1; δ = 0,1**

Двойственная задача формулируется как

$$Z(y) = 3y_1 + 4y_2 + 2y_3 + 0y_4 + 0y_5 + 0y_6 + 0y_7 \to \min; \quad (15)$$

$$\begin{cases} y_1 + 2y_2 + y_3 - y_5 = 3 \\ 2y_1 + y_2 + y_3 - y_6 = 4 \\ y_1 + y_2 - y_7 = 2 \\ y_1 = 0 \end{cases} \quad (16)$$

$$y_j >= 0,\ j = \overline{1,\ 7}.$$

На рис. 11 показана часть программы расчета точечного и интервального планов и ЦФ исходной задачи (13)—(14) с интервалами для ЦФ $F_{\min} < F_{\text{ср}} < F_{\max} \to$ $\to 28{,}28 < 10 < 36{,}58$.

На рис. 12 (см. четвертую сторону обложки) показаны сравнительные кривые базисных планов точечной и интервальных исходной (13)—(14) (рис. 12, *а*) и сопряженной (15)—(16) (рис. 12, *б*) задач. Первая двойственная компонента оптимального плана равна нулю как в точечных задачах, так и в интервальных вариантах.

Для проверки совпадения интервальных решений сопряженных задач проведена оптимизация ЦФ с погрешностями коэффициентов матрицы ε = 0,1; δ = 0. В результате равенства размерностей исходной и транспонированной матриц и коэффициентов систем, интервальные границы оптимизируемой ЦФ совпадают. Наборы оптимальных базисов обеспечивают один и тот же интервальный диапазон изменения для ЦФ, равный $\boldsymbol{F}_{\min} < \boldsymbol{F}_{\text{ср}} < \boldsymbol{F}_{\max} \to 30{,}23 < 10 < 36{,}50$, нижняя граница которого в отсутствие погрешности правой части значительно возросла, а верхняя почти не изменилась.

$$\begin{cases} x_{\max} = (5{,}1369,\ 3{,}7500,\ 2{,}9167,\ 4{,}0000,\ 0,\ 0,\ 0) \\ y_{\max} = (0,\ 0{,}6250,\ 2{,}0833,\ 1{,}5972,\ 0,\ 0,\ 0) \\ F_{\max} = 36{,}2500 \end{cases}$$

$$\begin{cases} x_{\min} = (4{,}7727,\ 2{,}5000,\ 2{,}9545,\ 4{,}0000,\ 0,\ 0,\ 0) \\ y_{\min} = (0,\ 0{,}4167,\ 1{,}8939,\ 1{,}4015,\ 0,\ 0,\ 0) \\ F_{\min} = 30{,}2273. \end{cases}$$

Графически эти результаты отражены на рис. 13 и 14 (см. четвертую сторону обложки). Зависимость переменных оптимального плана исходной и двойственной задач от номера компоненты показана на рис. 13. На рис. 14 обозначены точки интервальных границ ЦФ обеих сопряженных задач. На оси абсцисс отложены значения вторых компонент оптимальных планов *x* и *y*.

На рис. 14 видно, как идентичны обе картинки для изображения ЦФ в зависимости от свой координаты по оси абсцисс для сопряженных задач.

## Заключение

1. Предложена методика оценки чувствительности решения сопряженных задач линейного программирования с позиции интервальной алгебры и "знаковой" методики. Такой подход позволяет оценить диапазон предельных отклонений ЦФ в результате произвольных по знаку погрешностей коэффициентов технологических матриц и векторов правых частей систем.

2. Интервальные двойственные симметричные задачи переформулированы как две пары точечных систем, позволяющие получить допусковый интервальный диапазон изменения оптимизируемой функции, внутри которого находится точное значение ЦФ.

3. Соблюдены правила и постулаты основных теорем двойственности. Получено совпадение экстремальных значений ЦФ для обеих сопряженных задач в случае одинаковых размерностей основной и транспонированной технологических матриц.

4. Выявлены различия результатов оптимизации интервальной функции цели при одинаковых по знаку погрешностях коэффициентов матрицы и правых частей ограничений, и погрешностях, одинаковых с ними по модулю, но имеющих знаки, формализованные по "знаковой" методике.

5. Результаты оценки чувствительности ЗЛП подтверждают обоснованность применения выбранной методики как способа моделирования задач линейного программирования с допусковыми оценками, без которого, в силу неопределенности входных воздействий, нельзя гарантировать ни одного прогнозного решения задачи оптимизации.

6. Представленные в статье расчеты выполнены на основании разработанных автором программ, зарегистрированных ФСИС [14, 15].

**Список литературы**


1. **Карманов В. Г.** Математическое программирование. М.: Наука. 1975. 272 с.

2. **Еремин И. И.** Общая теория устойчивости в линейном программировании // Известия академии наук. 1999. № 12 (451). С. 43—51.

3. **Мастяева И. Н., Горбовцов Г. Я., Семенихина О. Н.** Исследование операций в экономике. М.: Московская финансово-промышленная академия, 2004. 110 с.

4. **Ашманов А. С.** Условия устойчивости задач линейного программирования // Журнал вычислительной математики и математической физики. 1981. Том. 21. № 6. С. 1402—1414.

5. **Ершов А. Г.** Гарантированные субоптимальные решения задач линейной оптимизации // PSI'03. ИМРО — Интервальная математика и распространение ограничений. Новосибирск, 8—9 июля 2003 г. С. 119—127. URL: http://www.sbras.nsc.ru/interval/Conferences/IMRO_03/Ershov.pdf

6. **Ащепков Л. Т., Давыдов Д. В.** Универсальные решения интервальных задач оптимизации и управления. М.: Наука, 2006. 151 с.

7. **Петров Ю. П.** Как обеспечить надежность решения систем уравнений. СПб.: БХВ-СПб, 2009. 172 с.

8. **Иванова К. Ф.** Точечно-интервальный метод оценки численного решения уравнений эллиптического типа // Программная инженерия. 2013. № 4. С. 21—28.

9. **Шарый С. П.** Конечномерный интервальный анализ. Новосибирск: XYZ, 2007. 700 с.

10. **Астафуров В. Г., Колодникова Н.** Компьютерное учебное пособие, раздел "Анализ на чувствительность с помощью двойственной задачи". Томск: Томский государственный университет, 2002.

11. **Шарая И. А., Шарый. С. П.** Допусковое множество решений для интервальных систем уравнений со связанными коэффициентами // Вычислительные технологии. 2009. Том 14, № 3. С. 104—123.

12. **Кремер Н. Ш.** Исследование операций в экономике: учеб. пособие для вузов. М.: ЮНИТИ, 2005. 407 с.

13. **Гаас С.** Путешествие в страну линейного программирования. М.: Мир. 1973. 176 с.

14. **Иванова К. Ф.** Свидетельство РФ о государственной регистрации программы для ЭВМ № 2011617669 "Программа для оценки целевой функции при решении задачи линейного программирования" (PZLP). 30.09.2011.

15. **Иванова К. Ф.** Свидетельство РФ о государственной регистрации программы для ЭВМ № 201261281. "Программа оценки решения моделей физических и технических задач на основе расчета интервальных линейных систем" (POPRIZ). 21.03.2012.


## *ИНФОРМАЦИЯ*

**20—22 февраля 2014 г. Минск, БГУИР**

### IV Международная научно-техническая конференция "Открытые семантические технологии проектирования интеллектуальных систем" (OSTIS—2014)

Основной целью ежегодных конференций OSTIS (Open Semantic Technology for Intelligent Systems) является создание условий для расширения сотрудничества различных научных школ, вузов и коммерческих организаций, направленного на разработку комплексной массовой технологии компонентного проектирования (модульного, сборочного проектирования) интеллектуальных систем.

Тематика конференции**:**

- Проблема независимости технологий проектирования интеллектуальных систем от различных платформ и вариантов их реализации
- Проблема интеграции интеллектуальных систем и их компонентов
- Принципы, лежащие в основе массовой технологии проектирования интеллектуальных систем на основе семантических сетей
-

**Рабочие языки конференции:** русский, белорусский, английский.

**Подробности:** http://conf.**ostis**.net

УДК 004.912

**Г. М. Ганкин,** аспирант, МГУ им. М. В. Ломоносова,
e-mail: gregorian21@yandex.ru

# Определение эмоциональной окраски коротких текстовых сообщений

*Представлены результаты исследований, направленных на реализацию инструментальных средств для определения эмоциональной окраски текстовых сообщений на русском языке. В частности, рассматриваются методы и средства классификации коротких текстов по шкале "положительная-отрицательная". Представлен обзор существующих решений и результатов в данной области, а также их классификация. Предложены решения поставленной задачи, использующие достаточно распространенный алгоритм машинного обучения — Support Vector Machine (метод опорных векторов). В заключительной части работы представлены результаты сравнительного анализа предложенного решения и уже существующих решений в рамках одного корпуса текстов.*

***Ключевые слова:*** *анализ текстов, эмоциональная окраска, машинное обучение*

## Введение

В настоящее время в Интернет ежедневно появляется огромный объем новой информации, которую пользователи добавляют с помощью различных сервисов. Значительную часть в этом потоке информации занимают сообщения в разных социальных сетях (ВКонтакте, Facebook, Twitter), на специализированных форумах, оценки каких-либо услуг и товаров (Яндекс.Маркет, различные сервисы, позволяющие оценить фильмы, и другие, подобные им). В исследовании [1], например, были изучены радикально-настроенные форумы. Целью была поставлена задача поиска экстремистских сообщений, например, разжигающих расовую ненависть. Анализировались сообщения как на английском, так и на арабском языках. В основе этого анализа лежала классификация сообщений по шкалам "позитивная — отрицательная" и "субъективная — объективная". В данной работе отмечается, что террористические структуры все активнее используют Интернет для своей преступной деятельности. Эффективный анализ подобных сообщений мог бы помочь органам охраны правопорядка понять принципы и цели террористической деятельности в виртуальном пространстве. Однако трудности реализации такого анализа на практике заключаются в том, что обработать вручную такой огромный поток подлежащей обработке информации технологически сложно и ресурсозатратно. Возможность автоматического анализа подобных сообщений существенно облегчила бы задачу.

В рамках социальной сети Twitter интерес представляет мониторинг сообщений по определенной тематике в динамике их поступления с анализом того, как реагируют пользователи Интернет на определенные события. Большое число исследований посвящено и другой задаче — корреляции содержания сообщений в социальных сетях с каким-либо процессом. В работе [2] рассматривались результаты опросов на различные темы, например, мнение о состоянии экономики в США, и сообщения в Twitter за тот же период на ту же тему. В работе [2] показано, что между ними существует корреляция, и в перспективе достаточно дорогие масштабные и оперативные социальные опросы можно заменить анализом сообщений в социальных сетях. В некоторых публикациях рассмотрены и менее очевидные зависимости. В работе [3], например, изучается влияние эмоциональной окраски сообщений в Twitter на состояние рынка ценных бумаг. В этой работе утверждается, что существуют фон-

ды, осуществляющие хеджирование (определенный вид сделок на рынке бумаг) на основе анализа сообщений социальных сетей. В исследовании [4] показано, что на основе анализа окраски сообщений в Twitter об определенном фильме можно предсказать его кассовые сборы в первые дни после выхода на экраны.

Что касается оценки услуг и товаров, это может представлять большой интерес для компаний, которым важно следить за отношением потребителей к различным брендам. Существует большое число коммерческих сервисов, предоставляющих услуги по анализу сообщений в социальных сетях на предмет изучения мнения о данной компании или отношения к выпускаемому продукту. Подобный анализ основан в том числе и на определении настроения пользователя по оставленным им сообщениям.

Целью исследования и разработки, результаты которых представлены в статье, является программный комплекс, на вход которого поступает текст, написанный на русском языке. На выходе возвращается класс, которому принадлежит текст — "позитивная окраска" или "отрицательная окраска" — с некоторой точностью $P$. Значение $P$ оценивается на основе применения алгоритма к большой выборке текстов, для которых уже известна их окраска. Оценить $P$ можно как долю объектов, верно классифицированных алгоритмом. Существуют также и другие способы оценки алгоритма, описанные в разделе "Метрики качества".

## Существующие алгоритмы

В большинстве случаев каждый из алгоритмов, решающих поставленную выше задачу, можно классифицировать по двум составляющим: по выбору характеристик текста, коррелирующих с его окраской, и по методу их обработки. Далее приводится перечисление некоторых таких характеристик и методов.

## Характеристики текста

Характеристика текста, как правило, представляет собой некую величину, которую можно вычислить для данного текста. В качестве примера можно рассмотреть распределения длин слов для данного текста. Существует большое число задач анализа текста, требующих предварительного определения его характеристик. К их числу относится, например, задача об определении авторства данного текста, изучение которой исторически началось несколько ранее, чем решение рассматриваемой в настоящей работе задачи. Эффективность использования данной характеристики оценивалась экспериментально. В качестве характеристик текста для определения его авторства в разное время предлагались следующие.

- Распределение длин слов, средняя длина слова [5—7].
- Словарный запас [8, 9]. В работе [10] в том числе предложены такие показатели, как $V_1 = \frac{V}{L}$; $V_2 = \frac{V}{\sqrt{L}}$; $V_3 = \frac{\log V}{\log L}$; $V_4 = \frac{\log V}{\log\log L}$, где $V$ — число уникальных слов в тексте, $L$ — общее число слов в тексте.
- Распределение частей речи и их позиции в предложении [11—13]. Рассматривалось распределение частей речи на нескольких первых позициях предложения и в конце предложения.
- Распределение высокочастотных (часто встречающихся) в языке слов [14, 15]. Для русского языка существует частотный словарь [16] с указанием частоты использования различных слов, в зависимости от части речи и ее стиля. Он составлен на основе Национального корпуса русского языка [17]. Самыми часто используемыми словами являются в основном предлоги, союзы и местоимения. Отметим, что на первом месте по частоте использования находится союз "и", который употребляется 35801,8 раз на миллион слов в коллекции текстов, использовавшихся для сбора статистики.
- Распределение сочетаний символов длины ($n$ — буквенные $n$-граммы) [18, 19].

Рассматриваются и другие характеристики текста.

Впоследствии выяснилось, что большинство этих характеристик также несет в себе информацию об окраске текста. В их числе:

- распределение длин слов, средняя длина слова [20, 27];
- распределение сочетаний $n$ слов $n$-граммы [21];
- распределение частей речи и их позиции в предложении [21].

В качестве характеристик текста, обладающих некоторой спецификой применительно к задаче определения эмоциональной окраски, использовались перечисленные далее.

- Наличие слов с эмоциональной окраской [22]. Для этого автоматически или вручную составляют словарь слов с различной окраской. В некоторых случаях им приписывают веса, показывающие степень выраженности окраски. Для русского языка подобный список можно получить, используя Национальный корпус русского языка [17], в котором вручную размечен большой объем лексики.
- Наличие фраз, синтаксических конструкций с эмоциональной окраской [23].
- Наличие определенных знаков пунктуации [20]. Например, можно считать число восклицательных, вопросительных знаков.
- Число редко встречаемых слов [24]. Под редко встречаемыми словами понимаются те, которые встречались малое число раз по коллекции текстов, однако достаточное, чтобы не считать их словом с опечаткой или просто несуществующим.

Встречаются также другие, отличные от представленных выше, характеристики.

Существует ряд характеристик текста, которые можно использовать именно при анализе сообщений из Интернета и социальных сетей. В частности, к их числу относятся следующие характеристики:

- наличие слов, написанных целиком заглавными буквами [20];

- наличие слов, окруженных астерисками (**хороший**) [20];
- наличие смайлов [20];
- наличие ссылок в данном тексте/на данный текст [25];
- наличие и содержание хэштегов (для сообщений в Twitter) [26]. Хэштеги — это метки, которые пользователи Twitter могут вставлять в свое сообщение, представляющие собой одну строку со знаком решетки перед ними. Строка может состоять как из одного слова (#праздник), так и из нескольких (#парадпобеды). Зачастую эти метки несут какую-либо эмоциональную окраску.

Все отмеченные выше исследования проводились для текстов на английском языке. Работ на тему эмоциональной окраски текстов на русском языке гораздо меньше. В разное время проводились различные соревнования для всех желающих по решению данной задачи. С 2006 по 2010 гг. можно было участвовать в конкурсе TREC, по условиям которого нужно было оценивать записи на английском языке, собранные более чем из 100 000 блогов. В 2006 г. в Японии был организован конкурс NTCIR, по условиям которого нужно было выполнять различные задачи, связанные с классификацией текстов, используя новостные статьи на английском, китайском и японском языках. Следует отметить соревнование по классификации отзывов пользователей, проводившееся в рамках Российского семинара по оценке методов информационного поиска (РОМИП) в 2011 г. [28]. В нем приняли участие 12 исследовательских групп. Подробнее условия соревнования будут представлены в разделе "Результаты тестирования предложенного метода". Пока же приведем характеристики, использовавшиеся для анализа текстов участниками этого соревнования. К их числу относятся:

- *n*-граммы [29—31];
- наличие смайлов [29, 31];
- наличие слов с эмоциональной окраской [32, 33];
- наличие фраз, синтаксических конструкций с эмоциональной окраской [33—35];
- пунктуация [32];
- длина предложений, структура предложения [32];
- *d*-граммы, предложенные в работе [30], основанные на синтаксическом дереве зависимостей предложения;
- наличие ссылок [31].

## Методы обработки характеристик текста

Одними из первых для определения эмоциональной окраски были методы, основанные на правилах (Rule-based methods). Они заключаются в поиске в тексте сообщения характерных синтаксических конструкций, указывающих на эмоциональную окраску. Простым примером правила может быть следующий: частица не перед словом с положительной окраской дает отрицательную окраску. Более сложные правила могут учитывать структуру предложения, например, наличие противительных и соединительных союзов в сложносочиненных или сложноподчиненных предложениях: "это кино хорошее, но чего-то ему не хватает, и сюжет не очень". Качество таких методов напрямую зависит от качества и количества предложенных правил и требует участия в работе лингвиста. Автоматический синтаксический разбор, особенно для русского языка, также является достаточно сложной задачей. Пример подобного подхода рассмотрен в работе [35]. Простые элементы данного подхода используют в сочетании с другими методами, например, с методами машинного обучения.

В некоторых случаях в задачах классификаций текстов используют различные статистические методы и критерии. Так, в работе [36] текст рассматривают как марковскую цепь символов. В работе [10] предложено использовать Хи-квадрат — статистику как меру определения сходности авторства двух текстов. Теоретически, подобные методы можно использовать и в задаче определения окраски текста, но надежных экспериментальных результатов на этом направлении пока нет.

Чаще всего на практике используют методы машинного обучения. В отличие, например, от методов, основанных на правилах, они позволяют учитывать большее число характеристик. Этот метод в большей степени независим от языка текста и автоматизирован.

## Методы классификации с использованием машинного обучения

Формальная постановка задачи: существует множество объектов $X$, есть множество классов $Y$, каждый объект принадлежит какому-либо из классов в $Y$. Существует конечное множество функций $f_i: X \to \Re$, называемых признаками объекта. Каждому объекту можно однозначно поставить в соответствие вещественный вектор признаков:

$$g: X \to \Re^n,\ g(x) = (\xi_1, \xi_2, ..., \xi_n) = (f_1(x), f_2(x), ..., f_n(x)).$$

Здесь $\xi_i$ — значение $i$-го признака для объекта $x$. В дальнейшем вместо объекта будем рассматривать соответствующий ему вектор признаков.

## Общие принципы работы методов

Любой метод машинного обучения работает в два этапа.

- *Этап обучения.* На вход подается выборка $T$ — набор пар $\{(x_i, y_i)\}$, где $x_i$ — объект, $y_i$ — класс, к которому он принадлежит. На основе выборки строится алгоритм $a$ классификации. Данный алгоритм получает на вход вектор признаков объекта, на выходе возвращает класс из множества $Y$. Как правило, алгоритм принадлежит заданному параметрическому семейству алгоритмов.

- *Этап применения*. На вход подается объект $x$, к нему применяется алгоритм классификации, полученный на первом этапе.

Для оценки качества работы вводится понятие функционала качества алгоритма на выборке объектов:

$$L(T, a) = \sum_i \frac{1}{|T|} I_{a(x_i) = y_i}.$$

Здесь $I_{a(x_i) = y_i}$ — функция-индикатор, равная единице, если $a(x_i) = y_i$, и нулю иначе. Одним из способов оценки результатов работы метода является разделение выборки на две части — обучающую и контрольную. На первом этапе формируется алгоритм классификации на основе обучающей выборки, затем вычисляется значение функционала качества на полученном алгоритме и контрольной выборке. Возможен вариант оценки качества методом кросс-валидации, а именно — выборка разделяется на $k$ равных частей таким образом, что $L = \coprod_{i=1}^{k} L_i$. Обучается $k$ алгоритмов, причем $i$-й алгоритм в качестве обучающей выборки использует множество $L/L_i$. Тогда для измерения качества классификации можно использовать параметр

$$L_x(T) = \frac{1}{|k|} \sum_{i=1}^{k} L(T_i, a_i).$$

Этот параметр лучше чем другие характеризует качество метода. Однако его вычисление при больших размерах выборки может занимать значительное время [37].

## Метод опорных векторов

Пусть $Y = \{1, -1\}$. Метод опорных векторов строит классификатор вида

$$a(x, w, w_0) = \text{sign}\left( \sum_{i=1}^{n} w_i \xi_i - w_0 \right).$$

Здесь $w = (w_1, ..., w_n) \in \Re^n$, $w_0 \in \Re$ — параметры алгоритма. Уравнение $\langle w, x \rangle = w_0$ описывает гиперплоскость в пространстве $\Re^n$. Идея метода заключается в построении оптимальной в некотором смысле гиперплоскости, разделяющей классы $y = 1$ и $y = -1$. Выборка линейно разделима, если существует такая гиперплоскость, что объекты разных классов находятся по разные стороны этой гиперплоскости. Оптимальной гиперплоскостью называется такая разделяющая гиперплоскость, что расстояние от нее до ближайшего объекта максимально по сравнению с другими разделяющими плоскостями. Эти требования записываются системой

$$\begin{cases} \langle w, w \rangle \to \min; \\ y_i(\langle w, x_i \rangle - w_0) \geqslant 1, \ i = 1...l. \end{cases}$$

На практике выборки обычно линейно неразделимы. Метод обобщается на этот случай путем разрешения ошибок на объектах. В этом случае требования будут записываться системой

$$\begin{cases} 0{,}5\langle w, w \rangle + C \sum_{i=1}^{k} e_i \to \min; \\ y_i(\langle w, x_i \rangle - w_0) \geqslant 1 - e_i, \ i = 1...l; \\ e_i \geqslant 0, \ i = 1...l. \end{cases}$$

Здесь $e_i$ — размер ошибки на объекте $x_i$, $C$ — параметр алгоритма, оптимальное значение которого находится экспериментально и регулирует соотношение между шириной полосы и суммарной ошибкой на объектах.

В случае если выборка линейно неразделима, как правило используется "ядерный трюк" (*kernel trick*), суть которого заключается в следующем. Пусть дано отображение $\varphi: \Re^n \to H$, где $H$ — пространство со скалярным произведением $\langle a_1, a_2 \rangle_H$. Если это пространство более высокой размерности, то вполне вероятно, что в нем выборка будет разделимой. Уравнение разделяющей гиперплоскости в пространстве $H$ будет иметь вид $\langle \varphi(X), w \rangle_H = w_0$. Ответ на вопрос "какой вид эта поверхность будет иметь в пространстве $x$", зависит от $H$. На практике само отображение нас не интересует, достаточно знать ядро — функцию $K(x_1, x_2) = \langle \varphi(x_1), \varphi(x_2) \rangle_H$.

Примеры ядер:

- $K(x_1, x_2) = \langle x_1, x_2 \rangle_{\Re^n}$ — линейное ядро, разделяющая поверхность будет иметь вид гиперплоскости;
- $K(x_1, x_2) = (\langle x_1, x_2 \rangle_{\Re^n})^d$ — полиномиальное ядро, разделяющая поверхность будет иметь вид поверхности порядка $d$;
- $K(x_1, x_2) = e^{\frac{\|x_1 - x_2\|^2}{2\sigma^2}}$ — гауссово ядро с параметром $\sigma$;
- $K(x_1, x_2) = \text{th}(k_0 + k_1 \langle x_1, x_2 \rangle_{\Re^n})$ — сигмоидное ядро с параметрами $k_0$, $k_1$.

Существуют правила, по которым можно построить свои ядра, однако во многих случаях на практике используют какие-либо из отмеченных выше ядер. Для модификации алгоритма с учетом ядер достаточно везде в программе заменить обычное скалярное произведение на функцию ядра.

Метод опорных векторов широко используют в различных задачах машинного обучения, в которых он показывает хорошие результаты. Возможность варьировать ядро $K$ и параметр $C$ обеспечивает методу гибкость. В общем случае временная сложность алгоритма достаточно высокая, а именно полиномиальная от выборки. Для линейного ядра существуют методы позволяющие найти близкую к оптимальной гиперплоскость за линейное время [37].

## Предлагаемый метод

В работе, результаты которой представлены далее, была выдвинута гипотеза, что использование буквенных *n*-грамм как характеристик текста, в сочетании с методами машинного обучения может дать хороший результат при определении эмоциональной окраски коротких текстовых сообщений и, в частности, сообщений из социальных сетей. Основанием для такого вывода является то обстоятельство, что одним из самых часто применяемых методов при классификации текстов является использование обычных *n*-грамм в сочетании с каким-либо методом машинного обучения, как правило это метод опорных векторов (support vector machine, *SVM*). Однако в силу специфики социальных сетей, рассматриваемые тексты в подавляющем большинстве содержат небольшое число слов, что отрицательно влияет на качество классификации при использовании *n*-грамм. Тексту соответствует большее число буквенных *n*-грамм. Кроме того, буквенные *n*-граммы были успешно использованы в задаче классификации текстов по их авторству [10] в сочетании с использованием статистических критериев. Были рассмотрены следующие два варианта.

- Каждому тексту ставится в соответствие численный вектор характеристик, состоящий из частоты встречаемости в нем буквенных *n*-грамм при $n \leqslant 3$, или индикаторов наличия буквенных *n*-грамм при $n > 3$.
- Для того чтобы избежать большого числа характеристик, можно использовать расстояния между распределениями. С этой целью для каждого заданного *n* вычисляется распределение буквенных *n*-грамм на данном тексте. Затем высчитываются расстояния данного распределения до распределений буквенных *n*-грамм на множестве текстов с положительной окраской и на множестве текстов с отрицательной окраской. В качестве расстояний были взяты следующие функции:

$$L_2(p, q) = \sum_i (p_i - q_i)^2;$$

$$L_1(p, q) = \sum_i |p_i - q_i|;$$

$$D_{KL}(p, q) = \sum_i \ln\left(\frac{p_i}{q_i}\right) p_i.$$

Здесь $p$, $q$ — дискретные распределения с вероятностью $i$-го значения $p_i$ и $q_i$ соответственно, $D_{KL}$ — расстояние Кульбака—Лейблера между двумя распределениями, которое в том числе использовалось при классификации текстов [38]. Таким образом, если использовать $m$ расстояний, то данному тексту при данном $n$ можно поставить в соответствие $2m$ расстояний: $m$ расстояний до распределения $n$-грамм на множестве текстов с положительной окраской и $m$ расстояний до множества текстов с отрицательной окраской.

Для получения лучшего результата при вычислении характеристик используют только *n*-граммы, встречающиеся в коллекции текстов больше заданного числа раз. Такой подход позволяет учитывать возможность опечаток в тексте. Кроме того, в данной задаче было предложено использовать ряд синтаксических характеристик текста, которые успешно использовались в задаче определения авторства текста. Таким образом, составлены следующие три набора характеристик.

- Синтаксические, в числе которых:
  - длина предложения в словах;
  - средняя длина слова в символах;
  - средняя длина предложения в словах;
  - средняя длина предложения в символах;
  - распределение количества слов заданной длины (от 1 до 20);
  - распределение частей речи;
  - распределение частей речи, стоящих на первой и второй позициях предложения.
- Буквенные *n*-граммы. Здесь можно рассматривать как набор характеристик при одном *n*, так и объединение нескольких наборов характеристик при разных *n*. Например, целесообразно рассматривать 2- и 3-граммы вместе в силу относительно небольшого их количества, в этом случае им соответствует набор характеристик, состоящий из частот встречаемсти буквенных 2- и 3-грамм.
- Расстояния между распределением буквенных *n*-грамм в данном тексте и распределением на множестве всех текстов одного класса.

## Метрики качества

Для оценки эффективности работы алгоритма на данной выборке необходимо ввести метрики качества. Используемые обозначения:

$tp_x$ — число объектов класса $x$, классифицированных алгоритмом как объект класса $x$;

$fp_x$ — число объектов не класса $x$, классифицированных алгоритмом как объект класса $x$;

$fn_x$ — число объектов класса $x$, классифицированных алгоритмом как объект не класса $x$;

$tn_x$ — число объектов не класса $x$, классифицированных алгоритмом как объект не класса $x$;

$S$ — множество всех классов, которым принадлежат объекты выборки.

Основными используемыми метриками качества являются следующие.

- *Precision* (точность) — доля объектов, классифицированных алгоритмом как $x$, которые действительно принадлежат классу $x$:

$$P = \frac{tp_x}{tp_x + fp_x}.$$

- *Macro Precision* — среднее для точности по всем классам:

$$Macro\_P = \frac{1}{|S|} \sum_{x \in S} \frac{tp_x}{tp_x + fp_x}.$$

• *Recall* (полнота) — доля объектов класса *x*, которая правильно классифицирована алгоритмом:

$$R = \frac{tp_x}{tp_x + fn_x}.$$

• *Macro Recall* — среднее по полноте по всем классам:

$$Macro\_R = \frac{1}{|S|}\sum_{x \in S}\frac{tp_x}{tp_x + fn_x}.$$

• *F1-measure* — гармоническое среднее для метрик *Precision* и *Recall*:

$$F1 = \frac{2PR}{P+R}.$$

• *Macro F1-measure* — среднее по *F1-measure* по всем классам:

$$Macro\_F1 = \frac{1}{|S|}\sum_{x \in S} F1_x.$$

• *Accuracy* — доля верно классифицированных объектов среди всех объектов по всем классам:

$$Accuracy = \frac{1}{|S|}\sum_{x \in S}\frac{tp_x + tn_x}{tp_x + fn_x + tn_x + fp_x}.$$

## Краткое описание работы программы

Для подтверждения гипотезы и исследования предложенных характеристик была написана программа, классифицирующая тексты по их эмоциональной окраске на два класса — "положительная" и "отрицательная". Программа написана на языке python и состоит из модуля, извлекающего описанные в разделе "Предлагаемый метод " характеристики текстов, и модуля, использующего метод опорных векторов для обучения и классификации текстов. Размер программы без учета обучающей выборки составляет примерно 28 кбайт, размер обучающей выборки 16 Мбайт.

*Пример кода программы:*

```
path = r'data/imhonet-films.xml' collection = Collection(path)
train_f = collection.get_features_long()
make_trainng_file(train_f, r'long.txt')
```

Краткое описание работы программы представляет собой описание последовательности действий на перечисленных далее этапах (см. рисунок).

• Анализ выборки текстов, предоставленной для обучения алгоритма. На этом этапе вычисляют характеристики каждого текста и средние характеристики для каждого класса. Для выделения характеристик текста использовались написанные на языке python библиотеки NLTK(Natural Language Toolkit) и pymorphy2. Библиотека NLTK для графематического, морфологического, семантического анализа, стемминга, лемматизации позволяет работать с текстами на различных языках. Эта библиотека распространяется под лицензией apache. Библиотека pymorhy2 позволяет проводить морфологический анализ текстов на русском языке и распространяется под открытой лицензией *MIT*. В качестве обучающей выборки использована коллекция отзывов на фильмы, предоставленная организаторами научного семинара РОМИП, подробнее выборка описана в разд. "Результаты тестирования предложенного метода".

• Формирование обучающей выборки, которая представляет собой набор векторов характеристик для каждого из текстов в выборке.

• Этап обучения метода машинного обучения применительно к полученной на предыдущем этапе

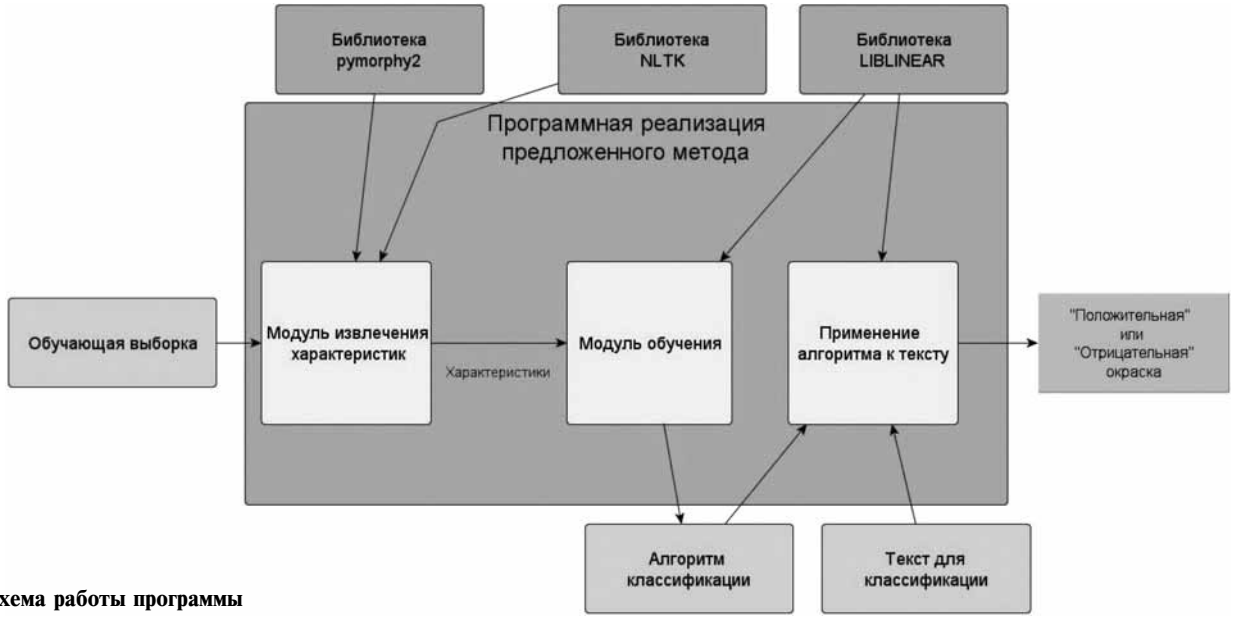


**Схема работы программы**

| Алгоритм | Метрика | | | |
|---|---|---|---|---|
| | *Macro_P* | *Macro_R* | *Macro_F1* | *Accuracy* |
| Участника zzz-23 | 0,776 | 0,797 | 0,786 | 0,881 |
| Участника zzz-9 | 0,706 | 0,794 | 0,730 | 0,812 |
| Участника zzz-14 | 0,743 | 0,597 | 0,623 | 0,860 |
| Baseline | 0,427 | 0,500 | 0,461 | 0,854 |
| SVM с использованием синтаксических характеристик | 0,427 | 0,500 | 0,461 | 0,854 |
| SVM с использованием расстояний между распределениями буквенных 2,3-грамм | 0,82 | 0,53 | 0,57 | 0,52 |
| SVM с буквенными 2- и 3-граммами | 0,81 | 0,55 | 0,59 | 0,545 |
| SVM с буквенными 4-граммами | 0,79 | 0,54 | 0,59 | 0,53 |
| SVM с буквенными 5-граммами | 0,613 | 0,683 | 0,603 | 0,6745 |
| SVM с буквенными 6-граммами | 0,654 | 0,710 | 0,668 | 0,766 |

выборке. Для этого используется реализация метода опорных векторов, представленная библиотекой LIBLINEAR.

- Этап применения — на этом этапе возможна классификация текстов, принадлежащих не только обучающей выборке.

## Технические требования

Для корректной работы разработанного программного обеспечения необходимы:

- система Windows или Linux;
- оперативная память не менее 4 ГБайт, зависит от объема обучающей выборки;
- программное обеспечение python версии 2.6 или выше, также должны быть установлены библиотеки NLTK, pymorphy2 для pytho.

### Результаты тестирования предложенного метода

Для задачи об определении эмоциональной окраски по шкале "негативная — позитивная" были протестированы наборы характеристик, описанные в разд. "Предлагаемый метод". Для второго набора использовались 2-, 3-, 4- и 5-граммы, для третьего были протестированы 2- и 3-граммы.

В качестве выборки для тестирования реализованного модуля были использованы тексты, предоставляемые для участников соревнований по оценке эмоциональной окраски текста РОМИП. Выборка состояла из следующих четырех наборов текстов.

- Отзывы о фильмах с портала Imhonet.ru. Коллекция представляет собой набор отзывов пользователей рекомендательного портала Imhonet.ru на фильмы различных жанров. Каждый отзыв имеет численную оценку от 1 до 10 баллов, соответствующую тексту. Число текстов в выборке — 15 718.
- Аналогичная выборка с отзывами о книгах с портала Imhonet.ru. Число текстов в выборке — 24 160.
- Отзывы о цифровых фотокамерах с сайта Яндекс.Маркет. Коллекция представляет собой набор отзывов пользователей о цифровых фотокамерах, оставленные пользователями на площадке Яндекс.Маркет. Каждый отзыв имеет численную оценку от 1 до 5 баллов. Число текстов в выборке — 10 370.
- Коллекция текстов из блогов с разметкой по оценочной тональности и объектам. Коллекция представляет собой набор текстов из блогов, которые участвовали в тестировании дорожек в рамках РОМИП-2011. Каждый текст относится к одной из трех тематик: книги, фильмы или цифровые фотокамеры. Кроме того, у каждого текста есть оценка асессора по двух-, трех- и пятибалльной шкале.

На время проведения соревнования пользователям были доступны только первые три выборки. Четвертая выборка использовалась для оценки качества классификации. Для оценки качества классификации на два класса использовались метрики *MacroPrecision*, *MacroRecall*, *Macro_F1*, *Accuracy*. В таблице представлены несколько лучших результатов участников конкурса при классификации отзывов на фильмы.

В таблице Baseline — алгоритм, который причисляет все объекты к самому многочисленному в выборке классу. Как видно, тестовая выборка сильно неравномерна, 85 % отзывов в ней являются положительными. В этом случае показатель *Macro_F1* является более объективным, что, в частности, видно по значениям показателей для Baseline-алгоритма.

Алгоритм с использованием синтаксических признаков показал тот же результат, что и Baseline алгоритм. Этот факт свидетельствует, что полученные численные представления документов неинформативны и оптимальной поверхностью оказалась та, по

одну сторону от которой оказались все объекты обучающей выборки.

Было проведено исследование влияния значения $n$ на точность классификации. Во всех случаях показание точности (*Accuracy*) не очень высоко. Это обусловлено большим перекосом в количестве положительных отзывов. Значение *Macro_F1* с увеличением $n$ возрастает и при при $n = 6$ превосходит третий результат в соревнованиях. Таким образом, 6-граммы показали третий по качеству результат среди 27 участников.

С учетом того обстоятельства, что алгоритм был обучен на достаточно небольшой выборке (многие участники использовали внешние источники для сбора статистики), полученный результат можно считать относительно высоким.

## Заключение

В данной работе представлены результаты исследования методов и средств для решения задачи определения эмоциональной окраски коротких сообщений. В рамках этих исследований изучены существующие способы определения эмоциональной окраски математическими методами. Предложены варианты модификации существующих алгоритмов. Предложены и исследованы характеристики текстов для данной задачи, а именно буквенные $n$-граммы и ряд синтаксических признаков. Реализовано программное обеспечение извлечения характеристик текста, описывающих эмоциональную окраску. Описан метод машинного обучения "Метод опорных векторов" и показан способ его применения для анализа полученных характеристик. Предложенные методы и программные средства на их основе прошли тестовые испытания, в том числе на реальных по объему и содержанию коллекциях тестовых данных.

В качестве дальнейшего направления исследований предполагается рассмотреть методы автоматического составления словарей слов с позитивной и отрицательной окраской. С учетом специфики сообщений в социальных сетях имеет смысл при поиске слов из словаря в тексте учитывать возможные опечатки. Для этого можно использовать методы, сходные с теми, что использует автор при поиске похожих сотрудников в системе ИСТИНА. При добавлении пользователем новой статьи в систему, ему необходимо указать всех ее авторов. В силу частых опечаток, при необнаружении указаного автора в системе проводится поиск сотрудников с похожими именами, при этом используются алгоритмы сравнения строк [39].

### Список литературы


1. **Abbasi A., Chen H., Salem A.** Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums // ACM Transactions on Information Systems. 2008. Vol. 26. Is. 3. P. 1—34.

2. **Balasubramanyan R., Routledge B., Smith N. A.** From tweets to polls: Linking text sentiment to public opinion time series. URL: http://www.cs.cmu.edu/~nasmith/papers/oconnor+balasubramanyan+routledge+smith.icwsm10.pdf

3. **Bollen J., Mao H., Zeng X.** Twitter mood predicts the stock market // Journal of Computational Science. 2011. Vol. 2. P. 1—8.

4. **Asur S. and Huberman B. A.** Predicting the future with social media // Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. IEEE Computer Society Washington, DC, USA. 2010. P. 492—499.

5. **Mendenhall T. C.** A mechanical solution of a literary problem // The Popular Science Monthly, 1901. Vol. 60. P. 97—105. URL: http://en.wikisource.org/wiki/Popular_Science_Monthly/Volume_60/December_1901/A_Mechanical_Solution_of_a_Literary_Problem.

6. **Radday Y.** Isaiah and the computer: A preliminary report // Computers and the Humanities, 1970. Vol. 5. P. 65—73.

7. **Juola P.** Authorship attribution // Foundations and Trends in Information Retrieval. 2006. Vol. 1. P. 233—334.

8. **Stamatatos E.** Author identification: Using text sampling to handle the class imbalance problem // Information Processing and Management: an International Journal. 2008. Vol. 44. P. 790—799.

9. **Kjetsaa G.** The battle of the quiet don: Another pilot study // Computers and the Humanities. 1977. Vol. 11. P. 341—346.

10. **Grieve J.** Quantitative authorship attribution: An evaluation of techniques. Simon Fraser University, 2005.

11. **Herdan G.** The advanced theory of language as choice and chance. Kommunikation und Kybernetik in Einzeldarstellungen. Springer-Verlag, 1966.

12. **Udny Y. G.** The Statistical Study of Literary Vocabulary. Shoe String Press Inc. U.S., 1968.

13. **Ledger G., Merriam T.** Shakespeare, fletcher, and the two noble kinsmen // Literary and Linguistic Computing. 1994. Vol. 9. P. 235—248.

14. **Burrows J. F.** Not unles you ask nicely: The interpretative nexus between analysis and information // Literary and Linguistic Computing. 1992. Vol. 7. P. 91—109.

15. **Smith T. and Witten I. H.** Language inference from function words. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi = 10.1.1.50.5903&rep = rep1&type = pdf

16. **Ляшевская О. Н., Шаров С. А.** Частотный словарь современного русского языка. М.: Азбуковник, 2009.

17. **Молдован А. М., Плунгян В. А., Сичинава Д. В.** Национальный корпус русского языка: новые перспективы русистики // Русский язык: исторические судьбы и современность: III Международный конгресс исследователей русского языка. Москва, МГУ им. М. В. Ломоносова, 20—23 марта 2007 г. Труды и материалы / Составители М. Л. Ремнева, А. А. Поликарпов. М.: МАКС Пресс, 2007. С. 26—27.

18. **Keselj V., Peng F., Cercone N., Thomas C.** N-gram-based author profiles for authorship attribution, 2003. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.7388&rep = rep1&type = pdf

19. **Bennett W. R.** Scientific and Engineering Problem-Solving with the Computer. Prentice Hall Series in Automatic Computation. Prentice Hall, first edition, 1976.

20. **Mishne G.** Experiments with mood classification in blog posts. URL: http://staff.science.uva.nl/~gilad/pubs/style2005-blogmoods.pdf

21. **Pang B., Lee L.** A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts // Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics. ACL'04, Stroudsburg, PA, USA. Association for Computational Linguistics, 2004. P. 271—278.

22. **Yu H., Hatzivassiloglou V.** Towards answering opinion questions: separating facts from opinions and identifying the polarity of opinion sentences // In Proceedings of the 2003 conference on Empirical methods in natural language processing. EMNLP '03, Stroudsburg, PA, USA. Association for Computational Linguistics, 2003. P. 129—136.

23. **Wilson T., Wiebe J. and Hoffmann P.** Recognizing contextual polarity in phrase-level sentiment analysis// In Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processin. HLT'05, Stroudsburg, PA, USA. Association for Computational Linguistics, 2005. P. 347—354.

24. **Wiebe J., Wilson T., Bruce R., Bell M., Martin M**. Learning subjective language // Journal of Computational Science. 2004, Vol. 30. P. 277—308.

25. **Agrawal R., Rajagopalan S., Srikant R., Xu Y.** Mining newsgroups using networks arising from social behavior // Proceedings of the 12th international conference on World Wide Web, 2003. P. 529—535.

26. **Kouloumpis E., Wilson T., Moore** J. Twitter sentiment analysis: The good the bad and the omg! // Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media. Barcelona, Spain, 17—21 July, 2011. AAAI Press, 2011. P. 538—541. URL: http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/download/2857/3251

27. **Michael G.** Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis // Proceedings of the 20th international conference on Computational Linguistics, COLING '04. 2004. P. 841—847.

28. **Chetviorkin I. I., Braslavski P. I., Loukachevitch N. V.** Sentiment analysis track at romip 2011 // Компьютерная лингвистика и интеллектуальные технологии: По материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая — 3 июня 2012 г. М.: Изд-во РГГУ, 2012. Т. 2. С. 1—14. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/83.pdf

29. **Chetviorkin I. I.** Testing the sentimentclassification approach in variousdomains — romip 2011 // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая — 3 июня 2012 г. М.: Изд-во РГГУ, 2012. Т. 2. С. 15—26. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/82.pdf

30. **Pak A., Paroubek P.** Language independent approach to sentiment analysis (limsi participation in romip'11 // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая — 3 июня 2012 г. М.: Изд-во РГГУ, 2012. Т. 2. С. 37—50. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/70.pdf

31. **Poroshin V.** Proof of concept statistical sentiment classification at romip 2011 // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая — 3 июня 2012 г. М.: Изд-во РГГУ, 2012. Т. 2. С. 60—65. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/96.pdf

32. **Kotelnikov E. V., Klekovkina M. V.** Sentiment analysis of texts based on machine learning methods // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая—3 июня 2012 г. М.: Изд-во РГГУ, 2012. Т. 2. С. 27—36. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/105.pdf

33. **Polyakov P. Yu., Kalinina M. V. Pleshko V. V.** Research on applicability of thematic classification methods to the problem of book review classification // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая — 3 июня 2012 г. М.: Изд-во РГГУ, 2012. Т. 2. С. 51—59. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/103.pdf

34. **Vasilyev V. G., Khudyakova M. B., Davydov S.** Sentiment classification by fragment rules // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая — 3 июня 2012 г. М.: Изд-во РГГУ, 2012. Т. 2. С. 66—76. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/132.pdf

35. **Kan D.** Rule-based approach to sentiment analysis at romip 2011 // Компьютерная лингвистика и интеллектуальные технологии: по материалам ежегодной Международной конференции "Диалог". Бекасово, 30 мая — 3 июня 2012 г. URL: http://www.dialog-21.ru/digests/dialog2012/materials/pdf/Kan.pdf

36. **Кукушкина О. В., Поликарпов А. А., Хмелев Д. В.** Определение авторства текста с использованием буквенной и грамматической информации // Проблемы передачи информации. 2001. Т. 37. С. 96—109.

37. **Воронцов К. В.** Курс лекций "Машинное обучение". URL: http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf

38. **Bigi B.** Using kullback-leibler distance for text categorization // Proceedings of the 25th European conference on IR research, 2003. Berlin, Heidelberg, Springer-Verlag, ECIR'03 2003. P. 305—319.

39. **Васенин В. А., Афонин С. А., Козицын А. С., Голомазов Д. Д., Бахтин А. В., Ганкин Г. М.** Интеллектуальная система тематического исследования научно-технической информации (ИСТИНА) // Обозрение прикладной и промышленной математики. 2012. Т. 19. Вып. 2. С. 239—240.

## *ИНФОРМАЦИЯ*

**С 12 по 15 ноября 2013 г. в Санкт-Петербурге в ВК Ленэкспо состоятся**

## 5-я юбилейная конференция "Информационная безопасность. Невский диалог" и 4-я специализированная выставка "Информация: Техника и технологии защиты"

*Мероприятия состоятся совместно с 22-й Международной выставкой и форумом "Охрана и безопасность — SFITEX"*

Выставка "Информация: Техника и Технологии Защиты" — единственная в России площадка, на которой представлен весь спектр современных технических средств обеспечения информационной безопасности: программные, программно-аппаратные, аппаратные средства защиты информации.

**Разделы выставки:**

- Сетевая безопасность
- Технические средства защиты информации
- Криптография и ограничение доступа
- Специальные технические средства
- Услуги и обучение
- Интеллектуальные карточные системы
- Безопасность электронного документооборота

**Подробнее о выставке и конференции www.iscs-expo.ru**

УДК 004.056.5

**В. П. Соловьёв,** канд. техн. наук, доц., зав. каф., **Д. А. Корнев,** аспирант, инженер,
e-mail: da.kornev@gmail.com,
Московский государственный университет путей сообщения (МИИТ)

# Сетевое взаимодействие в системах виртуализации Virtual Box и VMware

*Кратко проанализированы способы сетевого взаимодействия между виртуальной машиной и хостом, которые реализуются большинством современных систем виртуализации. К ним относятся NAT, сетевой мост, подключение только к хосту и внутренняя виртуальная сеть, не имеющая выход за пределы хоста. Также отмечены основные проблемные вопросы реализации этих режимов.*

***Ключевые слова:*** *сетевые технологии, безопасность данных, анонимность в информационных сетях, системы виртуализации*

В настоящее время широко используется технология виртуализации информационных сетей и отдельных вычислительных систем. В информационных технологиях под термином "виртуализация" обычно понимается абстрагирование вычислительных ресурсов рассматриваемой системы от их физических аналогов и предоставление пользователю системы, которая "инкапсулирует" (скрывает в себе) все свойства ее физической реализации [1]. Перспективные направления применения виртуализации стали очевидны в 1990-х гг. К ним относятся, например, исследования в области информационной безопасности, обеспечения стабильности разнообразных сервисов, повышение мобильности инфраструктуры сети и другие подобные им.

С ростом производительности аппаратуры как персональных компьютеров, так и серверов, появилась возможность использовать несколько виртуальных машин (ВМ) на одной физической платформе. В 1997 г. компания Connectix выпустила первую версию Virtual PC для платформы Macintosh, а в 1998 г. компания VMware запатентовала свои технологии виртуализации. На настоящее время продукты этих компаний являются основными конкурентами на рынке технологий виртуализации.

В настоящей статье представлен краткий анализ способов сетевого взаимодействия между ВМ и используемым для моделирования базовым хостом (далее для краткости — хостом), которые реализуют большинство современных программных систем виртуализации. В качестве таких систем далее преимущественно рассматриваются Virtual Box и VMware.

Также перечислены некоторые общие уязвимости, связанные с таким способом взаимодействия, основной из которых является высокая прозрачность, а следовательно, и простота компрометации передаваемого ВМ трафика со стороны хостовой системы.

Одним из основных направлений технологии виртуализации является виртуализация платформ, позволяющая создавать абстрактные информационные модели систем — ВМ с соответствующими программно-аппаратными ресурсами. Такие информационные модели могут использовать уже существующее на базовом хосте программно-аппаратное обеспечение или быть независимыми от него.

Рассмотрим принцип функционирования ВМ (как гостевой системы) в пределах хоста. Для этого необходимо, чтобы программное и аппаратное обеспечение хоста предоставляло соответствующий набор интерфейсов для доступа к его ресурсам. Существует несколько различных подходов к виртуализации, различающихся степенью абстрагирования эмуляции от аппаратного обеспечения подлежащего виртуализации физического аналога.

При виртуализации адресного пространства используемая для этого система создает несколько экземпляров аппаратного окружения, в частности, пространства адресов. Такой подход позволяет одновременно использовать ресурсы хоста и изолировать процессы, но не приводит к созданию обособленных операционных систем (ОС). Строго говоря, при таком способе виртуализации не создаются ВМ, а происходит обособление отдельных процессов на уровне ОС. Примером может служить использование режима UML

(*User-mode Linux*), в котором "гостевое" ядро запускается в пользовательском пространстве базового ядра (в его контексте).

При полной программной виртуализации все аппаратное оборудование ВМ представляет собой программную эмуляцию и никак не зависит от реального оборудования хоста. Этот подход позволяет создать ВМ любой архитектуры, например, использовать ОС Mackintosh на процессоре Intel. Однако следует заметить, что такой подход к виртуализации требователен к вычислительной производительности хоста, обладает низким быстродействием и по этой причине применяется относительно редко. Примером систем, реализующих такие подходы к виртуализации, являются Bochs, PearPC, QEMU (без ускорения), Hercules Emulator.

Наибольшее распространение на практике получила частичная (нативная) виртуализация. Причина в том, что быстродействие ВМ, основанных на принципе нативной виртуализации, с применением как программных, так и аппаратных возможностей хоста, существенно выше, чем при полной виртуализации, а требования к потребляемым ресурсам значительно ниже. Частичная виртуализация предполагает эмуляцию лишь некоторого аппаратного обеспечения, поэтому гостевые ОС должны быть разработаны для архитектуры соответствующего хоста. Частичная виртуализация использует специальную "прослойку" (гипервизор) между гостевой ОС и хостом, позволяющую гостевой системе напрямую взаимодействовать с аппаратным обеспечением. Данный подход к виртуализации обеспечивает хорошую вычислительную производительность ВМ при достаточно высоком, хотя и не полном, уровне абстрагирования от физических устройств. Примером таких систем виртуализации являются: VMware Workstation, VMware Server, VMware ESX Server, Virtual Iron, Virtual PC, VirtualBox, Parallels Desktop.

Одной из возможностей применения систем виртуализации является исследование механизмов взаимодействия компьютеров в сетях передачи данных. Для такого исследования необходимо обеспечить корректное взаимодействие между ВМ и сетевым интерфейсом хоста, а также требуемый уровень безопасности [2]. Поскольку несколько ВМ могут одновременно существовать на одном хосте, пропорционально возрастает и риск их уязвимости (компрометации). Так, например, любой трафик, передаваемый ВМ во внешнюю сеть за пределами хоста, проходит через сетевой интерфейс этого хоста. Это обстоятельство позволяет нарушителю при получении доступа к хосту прослушивать всю входящую и исходящую информацию, практически не выдавая себя. Нет необходимости перегружать ARP-таблицы сетевых устройств в целях перенаправления трафика и совершать другие аналогичные действия. Решением задачи по защите передаваемого ВМ трафика от компрометации со стороны хоста может стать шифрование всего трафика, исходящего из ВМ (например, используя SSL, TTL, HTTPS и другие протоколы). Разумеется, это не гарантирует полной безопасности виртуальных сетей, однако существенно усложняет сниффинг и требует применение MITM-атак для подделки сертификатов безопасности, что может быть выявлено при сличении подписи.

Подключение к сети ВМ осуществляется созданием ее виртуального сетевого интерфейса, который может быть "соединен" с физическим интерфейсом хоста. С точки зрения виртуализации сетевого интерфейса системы VirtualBox и VMware Workstation обладают близкими по функциональным возможностям механизмами. Однако следует отметить, что система VirtualBox имеет больше режимов сетевого взаимодействия. По этой причине принцип сетевого взаимодействия ВМ с хостом будет рассмотрен на ее основе. Очевидно, что существуют некоторые различия в реализации такого подключения. Так, например, системы VMware и VirtualBox имеют сходный режим сетевого моста, но в VMware виртуализируется коммутатор, а в VirtualBox — повторитель. Однако этот факт не имеет принципиального значения.

Система VirtualBox поддерживает до восьми виртуальных сетевых карт PCI Ethernet для каждой из ВМ. Интерфейс программы позволяет для каждой виртуальной карты указать тип эмулируемого оборудования и тип подключения (режим эмуляции и взаимодействия виртуальных и физических сетевых устройств хоста). Системой VirtualBox предусмотрены шесть типов подключения ВМ к сети:

- none — в гостевой системе отсутствует соответствующий интерфейс (сетевая карта) (рис. 1);
- null — в гостевой системе присутствует сетевой интерфейс, но отсутствует подключение к сети (как если бы кабель был отключен от интерфейса) (рис. 2);
- NAT (*Network Address Translation*) — сетевой интерфейс использует сетевую трансляцию адресов (рис. 3);
- bridge — сетевой интерфейс взаимодействует через сетевой мост (рис. 4);
- intnet — сетевой интерфейс взаимодействует с другими ВМ, используя внутреннюю сеть (рис. 5);
- hostonly — сетевой интерфейс ВМ совмещен с сетевым интерфейсом хоста (рис. 6).

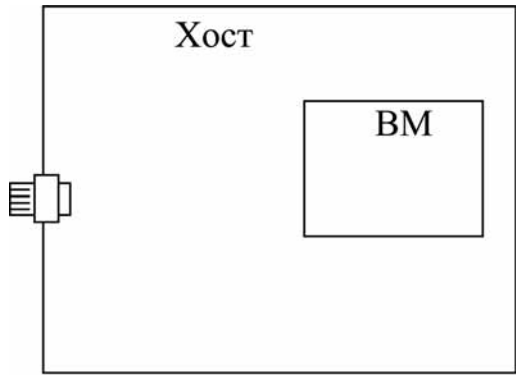


**Рис. 1. Режим none**

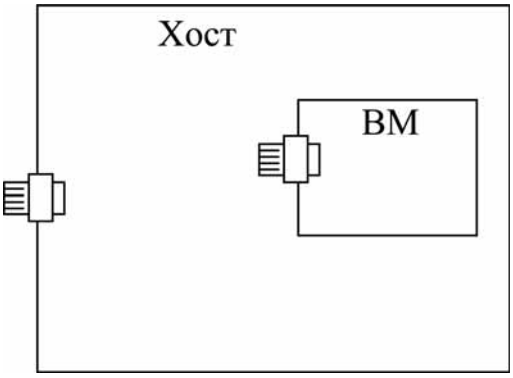


**Рис. 2. Режим null**

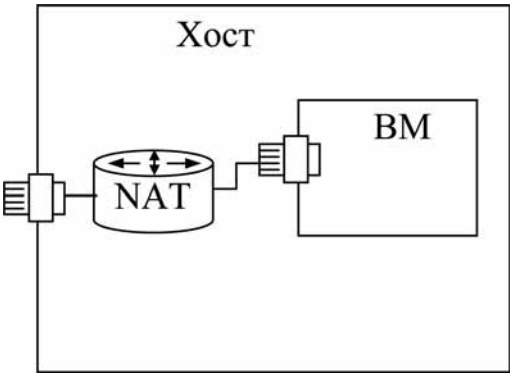


**Рис. 3. Режим NAT**

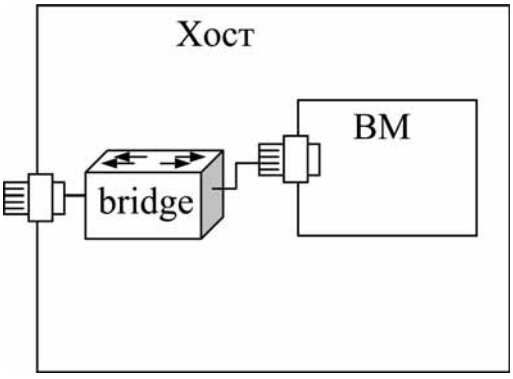


**Рис. 4. Режим bridge**

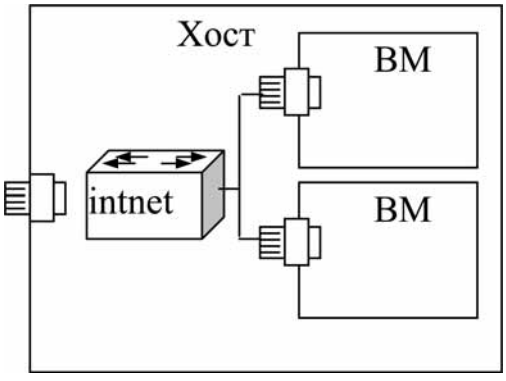


**Рис. 5. Режим intnet**

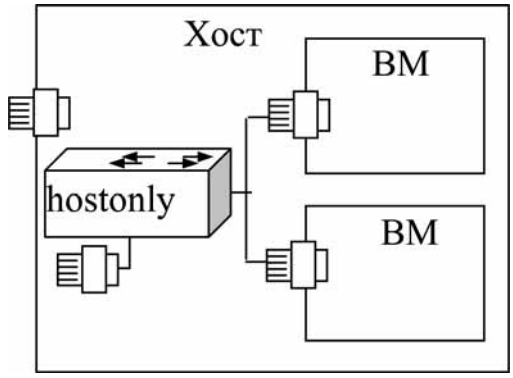


**Рис. 6. Режим hostonly**

Типы взаимодействия физического и виртуального сетевых интерфейсов none и null не реализуют подключения ВМ к хосту или к сети, поэтому не требуют дополнительных средств защиты.

Технологию NAT используют для подключения сетевого интерфейса ВМ к сети (см. рис. 3). Трансляцию сетевых адресов выполняет сетевой модуль VirtualBox (шлюз), который обрабатывает сетевой трафик ВМ. Источником сетевых адресов для ВМ в данном случае является DHCP-сервер, встроенный в VirtualBox. Сетевые пакеты, которые посылаются гостевой ОС, получает модуль VirtualBox-NAT, который обрабатывает данные стека TCP/IP и пересылает их ОС хоста. Механизмы ОС определяют данные, которые необходимо передать хосту, а также другим ВМ, расположенным в той же сети, что и хост. При этом используется сетевой интерфейс хоста. Система VirtualBox также принимает пакеты, адресованные ВМ, и пересылает их на ее виртуальный интерфейс.

При использовании технологии NAT поддерживается механизм загрузки ОС с удаленных источников (PXE). Сервер NAT DHCP предоставляет загрузочный файл vmname.pxe, если в каталоге с файлом VirtualBox.xml существует каталог TFTP.

Поскольку ВМ подключается к своей собственной сети VirtualBox и недоступна для хоста, ее сетевые службы также недоступны как для хоста, так и для компьютеров внешней сети, что является особенностью технологии NAT. Одновременно снижается прозрачность сети, а для выполнения ряда сетевых запросов требуется выполнение "перенаправления" портов (*port forwarding*). Однако путем "перенаправления" портов можно сделать выбранные сервисы доступными для компьютеров внешней сети. Такая процедура позволяет перенаправлять весь трафик с определенного порта хоста на сетевой интерфейс ВМ. Тем не менее применение технологии NAT дает некоторые преимущества. Например, уязвимость или отказ виртуальной сетевой службы не ведет к компрометации ОС хоста, так как сервис работает в другой системе. Настройку процесса "перенаправления" портов нельзя выполнить при работающей ВМ.

При реализации технологии трансляции адресов в системе VirtualBox существуют некоторые ограничения. К их числу относятся перечисленные далее.

- При применении сетевых утилит, таких как ping или tracerouting, использующих протокол ICMP, возможно их некорректное функционирование. Виртуальная машина может осуществлять ненадежное получение широковещательных пакетов. Такая ситуация складывается потому, что широковещательные пакеты доставляются только в определенный интервал времени, после того как ВМ отсылает пакет UDP. Такая организация процесса взаимодействия ВМ и сети позволяет повысить производительность процесса обработки информации. Как следствие, протокол разрешения имен NetBios не всегда работает корректно. Разрешить эту ситуацию можно, используя непосредственно IP-адреса для доступа к сетевым ресурсам. Следует отметить, что протокол WINS корректно осуществляет работу с широковещательными пакетами.
- При использовании технологии NAT в системе VirtualBox поддерживаются только TCP- и UDP-протоколы. Это означает, что нельзя использовать протокол VPN (или PPTP от Microsoft), а необходимо применять другие реализации протоколов аналогичных VPN, которые используют TCP и UDP.
- В приложениях ОС Unix (Linux, Solaris, MacOS X), которые запущены не с правами "root", нельзя использовать порты с номерами меньше 1024. В противном случае настроить "перенапраление" таких портов в ВМ не удастся (ВМ не запустится).

Применение технологии NAT существенно повышает уровень сетевой безопасности и усложняет проведение многих атак, начиная с этапа именуемого разведкой. Вместе с тем ее использование накладывает на сеть ряд серьезных ограничений. В частности, затрудняется удаленное администрирование, требуется "перенаправление" портов, снижается прозрачность используемой топологии, а следовательно, усложняется администрирование подобных сетей. Кроме того, усложняется процесс идентификации пользователя, выходящего во внешнюю сеть с использованием NAT, а также потенциально могут появиться трудности с реализацией некоторых протоколов.

В режиме bridge (сетевой мост) VirtualBox использует драйвер устройства, установленного на ОС хоста (см. рис. 4). Этот драйвер, обычно именуемый "net filter", обрабатывает данные, проходящие через физический сетевой интерфейс. Он позволяет системе VirtualBox отфильтровывать нужные пакеты из физической сети, вносить в них необходимые изменения и создавать новые программные сетевые интерфейсы. При использовании такого программного интерфейса ВМ воспринимается подключенной к физической сети, а ОС хоста может посылать и принимать данные от ВМ. Такой подход дает возможность применять физический интерфейс хоста в качестве маршрутизатора или шлюза между ВМ и физической сетью. Выполнить рассматриваемую процедуру, как было указано выше, можно с помощью драйвера от VirtualBox на ОС хоста.

В зависимости от ОС хоста существуют некоторые отмеченные далее ограничения при использовании режима "bridge".

- В ОС Macintosh ограничены функциональные возможности интерфейса AirPort. В настоящее время VirtualBox поддерживает только протокол IPv4 для AirPort, а для других протоколов, таких как IPv6 и IPX, необходимо выбирать проводные интерфейсы.
- В ОС Linux ограничены функциональные возможности беспроводных интерфейсов. Система VirtualBox поддерживает для беспроводных соединений только протокол IPv4, а для других протоколов, таких как IPv6 и IPX, нужно выбирать проводные интерфейсы.
- В ОС Solaris отсутствует поддержка беспроводных интерфейсов. Фильтрация трафика ВМ при использовании технологии IPFilter полностью не поддерживается, что связано с техническими ограничениями сетевой подсистемы Solaris. Предполагается, что этот недостаток будет устранен в следующих выпусках OpenSolaris.

Типы подключения intnet (внутренняя сеть) и bridge (сетевой мост), похожи. Однако режим подключения intnet позволяет осуществлять взаимодействие только между ВМ одного хоста (см. рис. 5). Режим bridge обладает более широкими возможностями для работы ВМ. Используя режим bridge ВМ может связываться с физической сетью через сетевой мост, что не позволяет осуществлять режим intnet. При этом сетевой мост реализует сеть между ВМ одного хоста, используя физический интерфейс. Тем не менее для использования режима intnet в системе VirtualBox существуют следующие аргументы.

1. Безопасность. В режиме bridge трафик между отдельными ВМ проходит через сетевой интерфейс хоста, поэтому существует возможность подключения сниффера (например, Ethereal или Wireshark) к интерфейсу хоста. Это позволяет "прослушивать" трафик, проходящий через интерфейс. Если необходимо защитить трафик между отдельными ВМ от "прослушивания", то необходимо исключить сетевой интерфейс хоста из маршрута трафика. Такую возможность предоставляет режим intnet.

2. Скорость. Режим intnet обладает большей скоростью передачи данных, чем режим bridge, так как VirtualBox исключает из маршрута трафика сетевой интерфейс хоста.

Внутренние сети создаются автоматически, т. е. не существует их централизованной настройки. Каждая внутренняя сеть идентифицируется своим именем. При существовании на хосте более чем одной виртуальной сетевой карты с одним идентификатором внутренней сети драйвер VirtualBox автоматически поддерживает соединение их интерфейсов в одну сеть, выступая в роли сетевого коммутатора. Система VirtualBox полностью поддерживает Ethernet-коммутацию, широковещательную и многоадресную рассылку сетевых пакетов, а также режим promiscuous mode.

Режим intnet удобно использовать для взаимодействия нескольких, предварительно настроенных ВМ, которые предназначены для совместной работы. Например, такой режим эффективен, если одна ВМ представляет собой web-сервер, который использует вторую ВМ с сервером базы данных. Режим bridge позволяет соединить web-сервер с внешней сетью для передачи данных, однако в этом случае серверная ВМ с базой данных не будет доступна для внешней сети.

Для сетевых карт ВМ в режиме intnet, как правило, используется статическая настройка IP-адресов. Для назначения IP-адресов во внутреннюю сеть также возможно использование DHCP-сервера, встроенного в VirtualBox. В качестве меры безопасности в режиме intnet ОС Linux предоставляет доступ к этой сети только ВМ, запущенным от имени пользователя, создавшего эту внутреннюю сеть.

Режим hostonly (виртуальный адаптер хоста) можно рассматривать как гибрид двух режимов: bridge (сетевого моста) и intnet (внутренней сети) (см. рис. 6). Как и в режиме моста, ВМ могут соединяться друг с другом и с хостом, что соответствует их соединению через физический коммутатор. Вместе с тем как и в режиме внутренней сети, нет необходимости в предоставлении физического сетевого интерфейса для подключения ВМ к хосту или к другим ВМ, запущенным на том же хосте. В этом случае ВМ не могут взаимодействовать с внешней сетью, так как они никаким образом не связаны с физическим сетевым интерфейсом хоста. При использовании режима intnet VirtualBox создает новый программный интерфейс, который добавляется к списку существующих сетевых интерфейсов хоста. Другими словами, для подключения ВМ к сети в режиме сетевого моста используется существующий физический интерфейс, а в режиме внутренней сети создается новый "петлевой" интерфейс хоста. По этой причине в режиме внутренней сети трафик между ВМ недоступен с хоста, а трафик "петлевого" интерфейса возможно "перехватить".

Из изложенного выше следует, что виртуальные сетевые устройства, обеспечивающие взаимодействие физических и виртуальных сетевых интерфейсов, работают по принципу коммутаторов, а чаще повторителей, не поддерживающих большинство решений в области информационной безопасности. Кроме того, между ВМ и хостом нельзя включить внешнее устройство, например, маршрутизатор или межсетевой экран, которые взяли бы на себя функции разграничения и сортировки трафика. Отмеченные особенности позволяют, используя скомпрометированную ВМ, совершать сетевые атаки на хост или на другие ВМ, подключенные к этой виртуальной сети. В крупных компаниях ВМ часто мигрируют между хостами, снижая уровень безопасности внутренней сети. Это обстоятельство создает серьезные дополнительные трудности, так как все механизмы информационной безопасности передаются внутренним брандмауэрам и антивирусам, установленным на ОС хостов и ВМ. В современных условиях таких средств защиты сетевой инфраструктуры явно недостаточно. В целях снижения рисков компанией VMware был разработан программный пакет VShield, в некоторой степени снижающий угрозу атак на виртуальные структуры.

Из приведенного краткого анализа способов сетевого взаимодействия следует, что для выхода ВМ во внешнюю сеть используется физический интерфейс этого хоста. Это обстоятельство приводит к повышенной уязвимости передаваемого трафика для прослушивания со стороны хоста. По этой причине очень важно обеспечить защищенный транзит трафика от ВМ во внешнюю сеть через базовый хост.

Кроме представленных выше соображений следует учитывать то обстоятельство, что любая информация, которая передается ВМ во внешнюю сеть (исключение составляет режим intnet), в любом случае пройдет через физический интерфейс хоста, и, соответственно, может быть им перехвачена и скомпрометирована. Причем, в отличие от реализации классических атак, направленных на перехват информации, нарушителю не нужно пытаться перенаправить трафик для прохождения через его сетевой интерфейс. Это происходит автоматически. Выявить такого нарушителя значительно сложнее, так как он не предпринимает никаких активных действий по перехвату — он только "прослушивает" "свой" сетевой интерфейс. Единственным решением задачи перехвата трафика на уровне хоста может быть шифрование всего трафика. Режим intnet позволяет взаимодействовать нескольким ВМ, запущенным на одном хосте. Передаваемая информация при этом не проходит через сетевой интерфейс хоста, таким образом повышается уровень безопасности. Вместе с тем возможности выхода за пределы хоста такой тип подключения не предоставляет.

Для подтверждения этого предположения авторами был выполнен ряд тестов на информационную уязвимость к активным и пассивным атакам, направленным на ВМ со стороны хоста, на котором эти машины создаются.

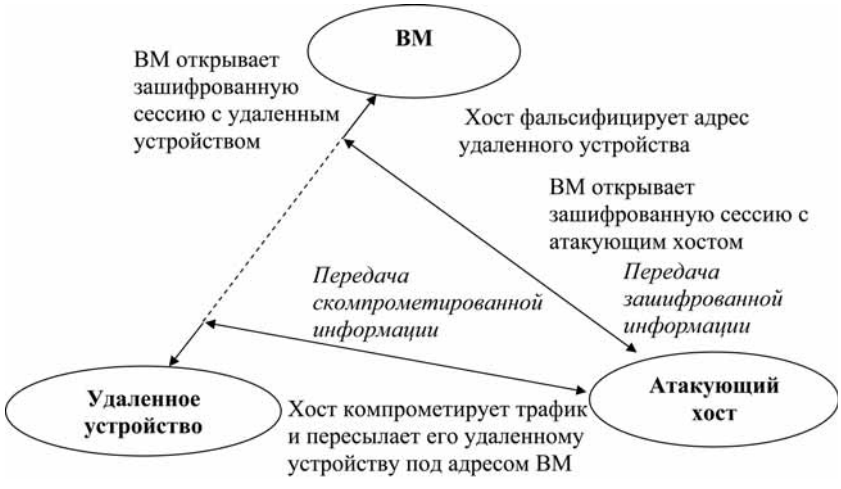


**Рис. 7. Механизм MITM-атаки на ВМ со стороны хоста**


**Рис. 8. Обобщенная блок-схема MITM-атаки в ходе тестирования на информационную уязвимость ВМ со стороны хоста**

В качестве тестовой платформы использовался персональный компьютер с ОС Linux Debian и системами виртуализации Virtual Box и VMware. На базе тестовой платформы были созданы ВМ с применением средств виртуализации обеих систем. Перехват трафика от ВМ осуществлялся программным обеспечением Wireshark и dsniff, установленным на хосте (рис. 7). В ходе тестовых испытаний осуществлялось подключение ВМ, созданных на разных системах виртуализации, к внешней и внутренней сетям, и последующая попытка компрометации информации со стороны хоста. Обобщенная блок-схема алгоритма проводимой атаки представлена на рис. 8. Результаты тестирования показали высокую степень прозрачности трафика ВМ со стороны хоста и простоту проведения как пассивных (прослушивание), так и активных (Man-in-the-Middle) атак. Этот факт свидетельствует о том, что обеспечение безопасности транзитной передачи информации от ВМ к удаленной станции является актуальной задачей.

Виртуализация информационных систем представляет широкие возможности для решения многих задач. Так, например, ВМ часто используются как "песочницы" для непроверенного и, возможно, опасного (зараженного или нестабильного) ПО. В случае системных сбоев в работе сервиса его можно быстро восстановить, загрузив ВМ из резервной копии. Общие репозитории ВМ позволяют пользователю работать со "своей" машиной, независимо от хоста.

Однако существуют и недостатки виртуализации. Виртуальные машины всегда работают медленнее, чем хост. "Песочницы" не всегда надежны — существуют вирусы, которые с достаточной долей вероятности выявляют системы виртуализации. В зависимости от реализации вируса он может не запуститься на ВМ, а в некоторых случаях "заразить" хост из ВМ (например, через общие папки). Кроме того, уплотнение информации на одном физическом оборудовании всегда ведет к повышению риска. Например, в случае "заражения" хоста, и "прослушивания" его сетевых интерфейсов, нарушитель сможет увидеть всю информацию, передаваемую ВМ в сеть.

Таким образом, обеспечение информационной безопасности при использовании сетевого взаимодействия в системах виртуализации приводит к необходимости решения следующих задач:

- защита сетевых каналов серверов виртуализации для обеспечения закрытой передачи фрагментов памяти гостевых машин, содержащих конфиденциальную информацию;
- создание аналогов классических межсетевых экранов и других средств информационной защиты, разграничивающих передаваемый трафик и память ВМ в пределах одного хоста;
- защита гипервизора для исключения ситуаций его использования при проведении Man-in-the-Middle-атак в процессе взаимодействия ВМ с другими системами.

На настоящее время наиболее перспективным направлением в области защиты виртуальных систем является создание новых протоколов, ориентированных на максимальный уровень анонимности и защиты передаваемой информации. В качестве альтернативы может рассматриваться разработка ПО, позволяющего разграничивать и фильтровать трафик между ВМ и хостом, а также выполняющего функции маршрутизатора, а не коммутатора или повторителя.

## Список литературы

1. **Самойленко А.** Виртуализация: новый подход к построению IT-инфраструктуры. URL: http://www.ixbt.com/cm/virtualization.shtml

2. **Oracle VM VirtualBox*.** Руководство пользователя. Oracle Corporation Copyright © 2004—2011 Oracle Corporation. URL: https://www.virtualbox.org/manual/UserManual.html

# CONTENTS



**ООО "Издательство "Новые технологии".** 107076, Москва, Стромынский пер., 4

Дизайнер *Т. Н. Погорелова*. Технический редактор *Е. М. Патрушева*. Корректор *Е. В. Комиссарова*

Сдано в набор 04.07.2013 г. Подписано в печать 21.08.2013 г. Формат 60×88 1/8. Заказ PI913
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
105120, г. Москва, ул. Нижняя Сыромятническая, д. 5/7, стр. 2, офис 2.