ТРУДЫ

ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 34 Выпуск 6

ISSN Online 2220-6426
Volume 34 Issue 6

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2022

ИСП РАН

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН
Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.
Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.
Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.
Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.
The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.
Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Сайт: http://www.ispras.ru/proceedings/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Web: http://www.ispras.ru/en/proceedings

Содержание

Table of Contents

DOI: 10.15514/ISPRAS-2022-34(6)-1

Статический анализатор для языков с обработкой исключений

[1,2] *Афанасьев В.О., ORCID: 0000-0002-8036-0633 <vafanasiev@ispras.ru>*
[1,3] *Дворцова В.В., ORCID: 0000-0002-7424-8442 <vvdvortsova@ispras.ru>*
[1] *Бородин А.Е., ORCID: 0000-0003-3183-9821 <alexey.borodin@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Национальный исследовательский университет Высшая школа экономики, 101000, Россия, г. Москва, ул. Мясницкая, д. 20*
[3] *Московский государственный университет им. М.В. Ломоносова, Россия, г. Москва, Ленинские горы д.1*

Аннотация. В статье описывается статический анализ для языков с обработкой исключений. В данной работе предложено низкоуровневое промежуточное представление для поддержки исключений; описаны анализы потока данных для поиска недостижимого кода, связанного с исключениями; приведена общая схема для статического анализа с учётом возможных путей, возникающих при использовании исключений. Алгоритмы реализованы в анализаторе Svace для языков C++, Java, Kotlin.

Ключевые слова: статический анализ; поиск ошибок; обработка исключений; Java; Kotlin; C++; Svace; JVM

Для цитирования: Афанасьев В.О., Дворцова В.В., Бородин А.Е. Статический анализатор для языков с обработкой исключений. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 7-28. DOI: 10.15514/ISPRAS-2022-34(6)-1

Static analysis for languages with exception handling

[1,2] *Afanasyev V.O., ORCID: 0000-0002-8036-0633 <vafanasiev@ispras.ru>*
[1,3] *Dvortsova V.V., ORCID: 0000-0002-7424-8442 <vvdvortsova@ispras.ru>*
[1] *Borodin A.E., ORCID: 0000-0003-3183-9821 <alexey.borodin@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the RAS, 25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia*
[2] *National Research University Higher School of Economics, 20, Myasnitskaya Str., Moscow, 101000, Russian Federation*
[3] *Lomonosov Moscow State University, Leninskie gory 1, Moscow, Russian Federation*

Abstract. This paper describes static analysis for the languages with exception handling. A low level intermediate representation, which supports exceptions, is proposed in this study. Data flow analyses for unreachable code detection were described. A general scheme of static analysis that takes exception related paths into account was given. The algorithms were implemented as a part of the static analysis tool Svace for C++, Java and Kotlin languages.

Keywords: static analysis; search for defects; vulnerabilities; Java; Kotlin; C++; Svace; JVM

For citation: Afanasyev V.O., Dvortsova V.V., Borodin A.E. Static analysis for languages with exception handling. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 7-28 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-1

1. Введение

Обработка исключений – это широко используемый метод управления ошибками в программах, он поддерживается в том числе такими языками как C++, Java и Kotlin. Специальные языковые конструкции позволяют бросить исключение, создав объект специального типа, а также поймать исключение по его типу, выполнив действия для обработки исключительной ситуации.

Исключения позволяют в некоторых случаях сделать код проще, позволяя избежать постоянной обработки кодов возврата. Кроме этого, выполнение части действий может гарантироваться семантикой языка. Например, в языке C++ будут вызваны деструкторы объектов во всех случаях выхода из функции. Последнее позволяет не забыть про освобождение ресурсов при возникновении исключительных ситуаций. В языках Java и Kotlin есть возможность автоматического вызова функции `close` с помощью конструкции «`try` с ресурсами». На листинге 1 показан код двух семантически эквивалентных функций. Функция `readConfigEx` значительно компактнее и проще читается.

```
1 void readConfig(String fileName, Reader reader) {
2   InputStream is = null;
3   try {
4     is = new FileInputStream(fileName);
5     if (reader.isEnd(is)) {
6       is.close();
7       return;
8     }
9   } catch (Throwable e) {
10    if (is != null)
11      is.close();
12    throw e;
13  }
14 }
15
16 void readConfigEx(String fileName, Reader reader) {
17   try (InputStream is = new FileInputStream(fileName)) {
18     if (reader.isEnd(is)) {
19       return;
20     }
21   }
22 }
```

Листинг 1. Пример автоматического вызова close
Listing 1. Example of try-with-resource

Многие уязвимости из перечня CWE (Common Weakness Enumeration) описывают ситуации, связанные с неправильной обработкой исключений, ошибочных ситуаций и т.п. [1]. Примеры таких ошибок: неправильное закрытие ресурсов при возникновении исключения, что может привести к их утечке; отсутствие конструкции, обрабатывающей исключение, которое возникает где-либо в программе, из-за чего данное исключение может аварийно завершить программу.

В статье мы рассматриваем задачу статического анализа исходного кода, содержащего инструкции обработки исключений, с целью поиска ошибок в нём. Мы не ограничиваемся лишь видами ошибок, напрямую связанными с исключениями, а ставим задачу создания анализатора для поиска произвольных ошибок в программах, использующих исключения.

Использование исключений в коде программ означает наличие специфических путей выполнения, возникающих при их бросании и обработке. Для успешного анализа статический анализатор должен уметь учитывать эти пути. Мы предлагаем подход, основанный на создании низкоуровневого промежуточного представления, явно содержащего переходы на возможные пути выполнения, связанные с обработкой исключений. Таким образом, задача разбивается на генерацию промежуточного представления, содержащего в явном виде все необходимые переходы, и на создание анализатора, который оперирует этим промежуточным представлением.

Описываемый подход был реализован в инструменте статического анализа Svace [2-4] для языков C++, Java, Kotlin, требующих поддержку исключений.

Статья построена следующим образом. В разд. 2 обсуждаются ситуации в исходном коде, которые требуют анализа исключений. В разд. 3 описывается анализируемый язык – низкоуровневое представление, предлагаемое для анализа исключений. Разд. 4 описывает анализ. Разд. 5 посвящен генерации промежуточного представления для языков C++, Java и Kotlin. Результаты тестирования на проектах с открытым исходным кодом представлены в разд. 6. В разд. 7 проведено сравнение с другими работами.

2. Исключения в исходном коде

В данном разделе мы приведём примеры ошибок, связанных с исключениями, и опишем необходимые свойства, требуемые от статического анализатора для поиска подобных ошибок в коде.

2.1 Пути выполнения, связанные с исключениями

Одна из самых часто встречающихся ошибок при работе с исключениями – неправильное освобождение ресурсов [5]. В листинге 2 приведён пример неправильного закрытия потока ввода. Код на строке 5 закрывает объект типа `FileInputStream`, но при этом в строке 4 может возникнуть исключение, что приведёт к тому, что поток не будет закрыт. В данном случае, ошибка может быть исправлена переносом строки 5 в `finally`-блок и объявлением переменной `is` перед началом `try`-блока.

```
1 try {
2   InputStream is = new FileInputStream(fileName);
3   byte b[] = new byte[is.available()];
4   is.read(b);
5   is.close();
6 } catch (Throwable t) {
7   System.err.println(t.getMessage());
8 }
```

Листинг 2. Пример неправильного освобождения ресурсов
Listing 2. Example of an incorrect resource release

Другой пример подобного рода уязвимости – некорректная обработка исключительной ситуации, приводящая к неправильному состоянию объекта. В листинге 3 приведён пример такой ошибки. Если исключение возникнет при вызове метода на строке 7, то объект `lock` всегда будет находиться в «захваченном» состоянии, что может привести к взаимной блокировке в другой части программы.

```
1 public class Foo {
2   private final Lock lock = ...;
3
4   public void foo() {
5     try {
6         lock.lock();
7         doSomething();
```

```
8           lock.unlock();
9         } catch (Throwable t) {
10          System.err.println(t.getMessage());
11        }
12      }
13
14    private void doSomething() { ... }
15  }
```

Листинг 3: Пример возможной взаимной блокировки
Listing 3: Example of a possible deadlock

В обоих случаях инструкция вызова функции может бросить исключение, которое приведёт к появлению другого пути выполнения программы. Игнорирование таких путей приводит к ошибкам в программе. Фактически выполнение инструкции вызова функции может создавать две группы путей: нормальное выполнение и выполнение с брошенным исключением.

2.2 Недостижимый код

Помимо создания новых путей, исключения могут сделать часть путей невыполнимыми. В листинге 4 приведён пример подобного кода. Строка 14, в которой значение входного аргумента `y` равно нулю, недостижима, так как в начале этой функции находится соответствующая проверка, где бросается исключение при нулевом значении.

```
1  void error(const char* msg) {
2      throw std::logic_error(msg);
3  }
4
5  int rem(int x, int y) {
6      if (y == 0) error("Division by zero");
7
8      int sign = (x < 0) ? -1 : 1;
9      if (y > 0) {
10         return sign * (std::abs(x) % y);
11     } else if (y < 0) {
12         return -sign * (std::abs(x) % -y);
13     } else {
14         return sign * INT_MAX;
15     }
16 }
```

Листинг 4: Пример недостижимого пути, вызванного брошенным исключением
Listing 4: Example of an unreachable execution path caused by thrown exception

Анализ, не умеющий отсеивать такие пути, будет менее точным и может выдавать малополезные предупреждения.

Более того, часть источников недостижимого кода может быть непосредственно связана с обработкой исключений. Например, код на языке C++, приведённый в листинге 5, содержит `catch`-блок, который никогда не может быть достигнут – так как исключение типа `std::logic_error` наследуется от `std::exception`, а обработчик исключения типа `std::exception` стоит выше, второй обработчик никогда не будет достигнут во время выполнения вне зависимости от реализации функции `foo`. Стоит отметить, что подобная проблема актуальна для языков C++ и Kotlin, но не для языка Java – компилятор этого языка посчитает подобный код некорректным.

```
1  try {
2      foo();
3  } catch (std::exception) {
4      // Catch std::exception
```

```
5 } catch (std::logic_error) {
6     // Catch std::logic_error
7 }
```

Листинг 5: Пример недостижимого catch-блока
Listing 5: Example of an unreachable catch-block

Таким образом, в ходе анализа важно уметь определять, что исключения могут быть подтипами друг друга.

2.3 Бесполезные предупреждения

В некоторых случаях разработчик может полагаться на наличие конструкции, обрабатывающей исключения.

В листинге 6 приведён пример кода, содержащего потенциальное разыменование нулевой ссылки. Метод на строке 2 может вернуть нулевую ссылку. Дальнейшее выполнение программы приведёт к разыменованию этой ссылки на строке 3.

```
1 try {
2   var ref = mayReturnNull();
3   ref.calc();
4 } catch (NullPointerException e) {
5 ...
6 }
```

Листинг 6: Пример catch-блока, обрабатывающего NPE
Listing 6: Example of a catch-block that handles NPE

Статический анализатор может найти эту ошибку и сообщить о ней. Но в данном случае предупреждение будет бесполезным, так как программист обрабатывает исключение типа `NullPointerException`.

3. Анализируемый язык

Для описания нашего подхода мы используем в данной статье следующий язык. Для краткости изложения мы ограничиваем наш язык лишь конструкциями, представляющими интерес с точки зрения обработки исключений.

Язык содержит следующие типы данных:

- целые числа;
- указатели[1] – поддерживаются указатели на целые числа и другие указатели; для простоты мы не стали рассматривать указатели на функции;
- исключения – имеют уникальное имя, представляют собой тип объектов, создаваемых и обрабатываемых в исключительных ситуациях, могут находиться в отношении наследования между собой.

Программы в этом языке будут представлять собой последовательность функций, последняя из которых является точкой входа. Каждая функция описывается графом потока управления (ГПУ), содержащим инструкции следующих видов:

- `assume(cond)` – выполнение завершается, если `cond` ложное;
- `!cond` – логическое отрицание;
- `a == b` – сравнение на равенство;

[1] Здесь и далее, когда мы будем упоминать несколько языков одновременно, мы будем использовать термин «указатель», подразумевая под ним указатели в языке C++ и ссылки в языках Java и Kotlin.

- `a != b` – сравнение на неравенство;
- `a = *b` – разыменование;
- `*a = b` – запись в память;
- `return a` – единственный возврат из функции;
- `a = alloca()` – выделение памяти на стеке;
- `r, e = func(a, b, …)` – вызов функции, которая может бросить исключение; здесь `e` – код исключения, который может иметь значение ∅, если функция не бросает исключение; для функции, которая точно не бросит исключение, также будем использовать обозначение `r = func(a, b, …);`
- `throw e` – возврат из функции с исключением;
- `try_enter(e1, e2, …)` – вход в блок, обрабатывающий исключения `e1, e2, …`;
- `try_exit(e1, e2, …)` – выход из блока, обрабатывающего исключения `e1, e2, …`;
- `e1 ≤: e2` – возвращает истину, если `e1` является подтипом `e2` или типы `e1` и `e2` эквивалентны.

Условные конструкции в нашем языке представляют собой недетерминированное ветвление и линейную инструкцию `assume(cond)`, имеющую следующую семантику: инструкция завершает выполнение программы, если условие `cond` ложно. Инструкция `assume` является линейной, то есть на графе потока управления имеет одно входное ребро и одно выходное, что значительно упрощает анализ.

В данной работе мы не будем рассматривать случаи, когда выполнение произвольной инструкции, кроме инструкций вызова и оператора `throw`, может привести к исключению (так, в Java разыменование нулевой ссылки приведёт к исключению типа `NullPointerException`). С одной стороны, это бы затруднило анализ, так как увеличило бы размер промежуточного представления и потребовало бы адаптации каждого детектора. С другой стороны, ошибки, возникающие при выполнении подобных инструкций, можно обнаружить при помощи специализированных анализов, которые не требуют поддержки анализа исключений. Так, ошибка выхода за границы массива может искаться отдельным детектором (например, при помощи подхода, описанного в статье [6] поэтому поддержка путей выполнения, создаваемых исключениями типа `ArrayIndexOutOfBoundsException` в языках Java и Kotlin, была бы избыточной.

Для простоты будем считать, что в языке нет глобальных переменных. Они легко моделируются с помощью параметров функции. Кроме этого мы не рассматриваем программы с циклами, так как с точки зрения обработки исключений циклы не вносят существенных изменений по сравнению с условными выражениями.

На рис. 1 приведён пример ГПУ, который может быть сгенерирован для кода из листинга 7.

```
1 r = -1;
2 try {
3   r = foo(a, b, ...);
4 } catch (e1) {
5 // Catch-block e1
6 } catch (e2) {
7   // Catch-block e2
8 }
9 return r;
```

Листинг 7: Пример кода, обрабатывающего исключения
Listing 7: Example of code with exception handling

На приведённом ГПУ на рис. 1 есть четыре группы путей:

- путь для выполнения без исключений;
- путь, обрабатывающий исключение `e1`;
- путь, обрабатывающий исключение `e2` (заметим, что если исключение `e2` вляется подтипом `e1`, то этот путь будет невыполнимым);
- путь, соответствующий другим исключениям, отличным от `e1` и `e2`.

В зависимости от реализации функции `foo`, любой из этих путей может быть невыполнимым.

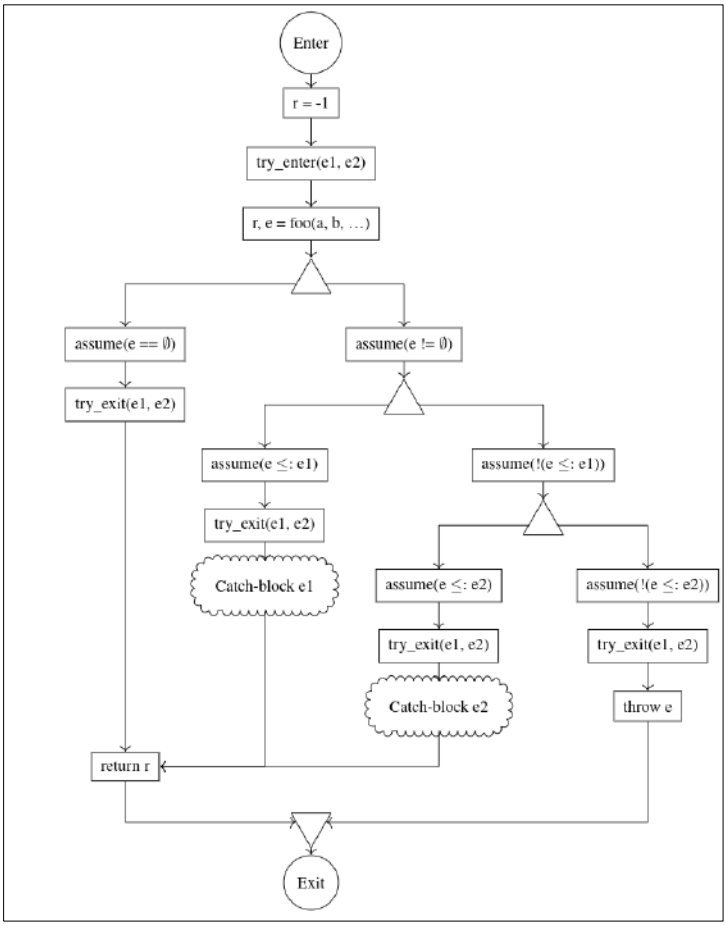

Рис. 1: Пример ГПУ для кода из листинга 7
Fig. 1: Example of a CFG for the code from listing 7

4. Анализ

4.1 Общая схема анализа

Для поддержки межпроцедурного анализа мы используем анализ на основе резюме. При таком подходе после завершения анализа функции создаётся её резюме, которое используется в точках вызова функции. Таким образом, межпроцедурный анализ состоит из двух операций: создание резюме и применение резюме в контексте вызова. Подобный анализ

является *контекстно-чувствительным*, так как одно и то же резюме для некоторой функции применяется по-разному в контексте каждого её вызова.

Для анализа отдельной функции используется символьное выполнение с объединением состояний в точках слияния путей. Подробно данный анализ описан в [4]. Здесь отметим, что этот анализ имеет чувствительность к путям, выполняет моделирование полей структур, указываемых ячеек памяти и элементов массивов. На его базе реализовано множество детекторов, в том числе ошибки разыменования нулевых указателей, деления на ноль, переполнения буфера, утечки ресурсов.

Перед запуском символьного выполнения для каждой функции запускается дополнительный движок анализа, основанный на анализе потока данных (DFA-engine) [7], целью которого является поиск недостижимого кода. Движок помечает недостижимые рёбра, которые он сможет обнаружить. При анализе объединений на ГПУ учитываются только достижимые входные рёбра. Таким образом, анализ недостижимого кода может улучшить точность основного анализа.

В данной работе необходимо дополнить анализ путями выполнения, связанными с обработкой исключений, что обеспечивается использованием описанного промежуточного представления. Также требуется расширить DFA-движок для определения новых источников недостижимого кода.

4.2 Анализ потока данных

Общая схема анализов в DFA-движке выглядит следующим образом:

1) для каждой инструкции создаётся передаточная функция, которая для входного состояния для некоторой полурешётки возвращает выходное состояние;
2) запускается итеративный анализ по ГПУ с использованием передаточных функций.
3) при необходимости информация сохраняется в резюме функции;
4) выполняется обход ГПУ, который проверяет состояния на входных рёбрах и саму инструкцию; если можно сделать вывод о недостижимости, то выходное ребро помечается как недостижимое;
5) информация о недостижимости распространяется на последующие инструкции базового блока; кроме того, если все входные рёбра в точке объединения путей недостижимы, то выходное ребро так же помечается как недостижимое.

Различные анализы выполняются поочерёдно и дополняют результаты друг друга. Если ребро помечено как недостижимое, то все последующие анализы не будут его учитывать.

Для анализа исключений мы расширили DFA-движок анализом функций, которые бросают исключения только определенных типов (п. 4.2.2). В дополнение к основному анализу был реализован вспомогательный анализ, сохраняющий обрабатываемые исключения (п. 4.2.1). Реализованные анализы имеют потоковую чувствительность без чувствительности к путям выполнения.

4.2.1 Анализ, сохраняющий обрабатываемые исключения

Целью данного анализа является определение для каждого ребра на ГПУ функции типов исключений, которые в этой точке обрабатываются. Данный анализ в DFA-движке выполняется первым.

Анализ реагирует на инструкции `try_enter(e1, e2, …)` и `try_exit(e1, e2, …)`: при входе в `try-catch`-конструкцию обрабатываемые типы исключений добавляются в соответствующее множество, а при выходе из неё типы исключений, обработка которых заканчивается в этой точке, удаляются из множества. В точке сбора происходит пересечение множеств.

Данный анализ является вспомогательным. Некоторые детекторы используют его результаты для подавления неактуальных предупреждений. Например, они используются, чтобы подавлять неактуальные предупреждения о разыменовании нулевой ссылки, если исключение типа `NullPointerException` обрабатывается (см. листинг 6).

4.2.2 Анализ бросаемых исключений

Элементами полурешётки является множество типов бросаемых исключений. Нижним элементом является пустое множество. Также выделен специальный элемент, являющийся верхним элементом, означающий все возможные варианты. Оператором сбора является объединение множеств бросаемых исключений. С целью производительности мы ограничили размер множества десятью элементами. Если размер превышен, то множество заменяется на верхний элемент.

Полурешётка имеет следующую семантику: если на некотором ребре значением полурешётки является множество Exc, то это значит, что при выполнении программы могут быть брошены только перечисленные исключения. Невозможна ситуация, когда бросается исключение $e \notin Exc$.

Передаточные функции анализа:

$$TF[throwe] = Exc \to \{e\}$$
$$TF[r, e = foo(\dots)] = Exc \to Exc \cup summary(foo)$$
$$TF[r, e = foo(\dots); summary(foo) = \emptyset] = Exc \to T$$
$$TF[r = foo(\dots)] = Exc \to Exc$$
$$TF[assumee == \emptyset; r, e = call] = Exc \to \emptyset$$
$$TF[assumee \leq: type; r, e = call] = Exc \to \emptyset$$
$$TF[assume!(e \leq: type); r, e = call] = Exc \to Exc \setminus \{type\}$$

Для инструкции бросания исключения `throw e` результирующее множество содержит только это исключение. Для инструкции вызова функции `r, e = foo(…)` ко множеству возможных исключений добавляется множество из резюме этой функции. Если по каким-то причинам для вызова функции нет резюме (`summary(foo) = ∅`), то мы используем верхний элемент решётки, предполагая, что может быть брошено любое исключение. Для функций, не бросающих исключение, используется передаточная функция, которая ничего не меняет.

Если функция не бросила исключение (`assume e == 0; r, e = call`), то используется нижний элемент решётки. Инструкция `assume e ≤: type; r, e = call` означает, что было перехвачено исключение определённого типа. При выполнении данного кода никакие другие исключения не могут быть брошены, поэтому мы также используем нижний элемент. И для случая, когда какое-то исключение не было поймано (`assume !(e ≤: type); r, e = call`), из множества возможных исключений убирается этот тип.

Результаты данного анализа могут быть использованы для пометки исключений для двух случаев:

1) `assume e ≤: type` – выходное ребро помечается как недостижимое, если множество исключений не содержит данный тип: $type \notin Exc$. Этот случай соответствует ребру входа в `catch`-блок;
2) `assume !(e ≤: type)` – выходное ребро помечается как недостижимое, если множество исключений содержит только данный тип: $Exc = \{type\}$. Данный случай соответствует ребру обхода `catch`-блока.

Описанный анализ позволяет понять, что в листинге 8 блок `catch(…)` недостижим. Последний факт позволяет анализу ошибок деления на ноль не выдать ложное

предупреждение, так как на всех остальных путях переменной x присваивается ненулевое значение.

```
1 void foo(int a) {
2    if (a <= 0) throw 8;
3 }
4 int bar(int a) {
5   int x = 0;
6   try {
7     foo(a);
8     x = 1;
9   } catch(int e) {
10    x = 2;
11 } catch(...) { }
12   return 100 / x;
13 }
```

Листинг 8: Недостижимый `catch`-блок
Listing 8: Unreachable `catch`

Частный, но важный случай, находимый этим анализом, – это функции, которые точно не бросают исключения. Для вызова таких функций все рёбра, соответствующие обработке исключений, помечаются как недостижимые.

4.2.3 Анализ точно брошенных исключений

Некоторые функции всегда бросают исключения при выполнении. Для анализа таких функций был разработан несложный анализ, задача которого – поместить в резюме информацию, что нормальное выполнение функции невозможно и функция всегда бросает исключение.

Рёбра ГПУ, соответствующие нормальному выполнению таких функций, помечаются как недостижимые.

4.3 Детектор необработанных исключений

Одна из часто встречающихся уязвимостей при обработке исключительных ситуаций – необработанные исключения [8]. Данная уязвимость может привести к аварийному завершению программы, что может быть использовано злоумышленниками при атаке типа «отказ в обслуживании»..

В листинге 9 приведён пример необработанного исключения. В строке 5 производится вызов метода `parseInt`, который может бросить исключение типа `NumberFormatException`, если передаваемая строка не является числом. Так как исключение данного типа не обрабатывается, а передаваемый аргумент не проверяется, то программа может завершиться аварийно при некорректных значениях.

```
1 public static void main(String[] args) {
2   int threadsNum = 1;
3   for (int i = 0; i < args.length; ++i) {
4     if (args[i].startsWith("-j")) {
5       threadsNum = Integer.parseInt(args[i].substring("-j".length()));
6     }
7   }
8   System.out.println("Number of threads: " + threadsNum);
9 }
```

Листинг 9: Пример необработанного исключения
Listing 9: Example of an unhandled exception

Для поиска необработанных исключений был разработан специальный детектор.

Детектор использует межпроцедурный анализ для поиска исключений, которые бросаются в программе, но не обрабатываются и могут аварийно завершить эту программу.

При анализе используются три множества типов исключений[2]:

- *Expected* – типы исключений, которые ожидаются в данной точке программы (то есть, обрабатываются конструкцией `try-catch`). Формируется при помощи результатов вспомогательного анализа, описанного в п. 4.2.1};
- *Uncaught* – типы исключений, которые не были обработаны в `catch`-блоке; в точке сбора происходит объединение данных множеств;
- *Processed* – типы исключений, которые были обработаны в `catch`-блоке; в точке сбора происходит пересечение данных множеств.

При достижении вызова функции исключения, которые могут быть брошены в результате этого вызова, разделяются на пойманные и не пойманные, формируя множества *Processed* и *Uncaught* соответственно. При этом трассы предупреждений расширяются меткой `call` с информацией о вызываемой функции.

При достижении инструкции промежуточного представления, соответствующей оператору `throw`, тип исключения ищется во множествах *Processed* и *Uncaught*. Возможны три следующих ситуации:

- если исключение находится во множестве *Processed*, то оно было поймано `catch`-блоком, но в итоге было брошено заново – тогда трасса для этого типа исключения расширяется меткой `rethrow`, тип удаляется из множества *Processed* и добавляется во множество *Uncaught*;
- если исключение находится во множестве *Uncaught*, то оно не было поймано `catch`-блоком, но достигло `finally`-блока и после его завершения было брошено заново – в этом случае трасса для этого типа исключения остаётся прежней, тип остаётся во множестве *Uncaught*;
- если исключения нет ни в одном из множеств, то такая ситуация соответствует созданию и бросанию нового типа исключения через оператор `throw` – в таком случае создаётся новая трасса для этого типа исключения с меткой *Uncaught* и информация добавляется во множество *Uncaugh.*

При достижении выхода из функции, её сигнатура сверяется с функцией `main` – если текущая функция является точкой входа в программу, то для всех исключений из множества *Uncaught* должно быть выдано соответствующее предупреждение. В ином случае, если функция не является функцией `main`, то все исключения из множества *Uncaught* добавляются в резюме анализируемой функции и используются при обработке вызовов.

Для языка C++ помимо функции `main` проверяются все деструкторы, так как деструктор может быть вызван во время раскрутки стека при уже брошенном исключении, и если в этот момент будет брошено ещё одно исключение, то будет вызвана функция `std::terminate`, завершающая программу [9].

Также нужно отметить, что в языке C++ исключение может завершать программу аварийно и при использовании так называемых «exception specifications»: если функция определяет, что из неё могут быть брошены исключения типов `e1, …, eN`, но при этом бросает исключение типа `u` такое, что `!(u ≤: ei)` для всех `i`, то будет произведён вызов функции `std::terminate`. Поэтому, предупреждение о необработанном исключении будет выдано и в том случае, если множество *Uncaught* содержит типы исключений, которые не описаны в объявлении функции.

[2] Все три множества являются частью множества *B* состояния программы.

Для описания свойств поведения библиотечных функций, важных для детекторов, анализатор *Svace* использует так называемые спецификации, которые моделируют эти свойства при помощи вызовов специальных функций. Мы добавили спецфункции `sf_could_throw` и `sf_could_throw_pedantic`, означающие, что из функции может быть брошено исключение определённого типа. Версия `pedantic` используется для случаев, когда возникновение исключений маловероятно.

Отметим, что не все типы исключений одинаково критичны – некоторые возникают в очень редких случаях, а другие могут указывать о наличии серьёзных проблем в работе программ. Например, исключение типа `OutOfMemoryError` в JVM вряд ли должно обрабатываться приложением, так как оно не сможет корректно восстановиться после данного исключения. Соответственно, можно разделить типы предупреждений на те, которые относятся к критичным типам исключений, и те, которые относятся к исключениям, которые могут быть не обработаны или не являются критичными. Для пометки типов исключений, как те, которые обязаны быть обработаны, в спецификациях *Svace* для Java и Kotlin была добавлена аннотация `MustBeCaught`, при помощи которой можно пометить необходимый тип исключения. Этот признак наследуется: если класс `A` имеет аннотацию `MustBeCaught` и является родителем класса `B`, то исключение типа `B` тоже должно быть поймано.

Также стоит отметить, что для некоторых типов исключений предупреждение выдавать и вовсе бессмысленно. Например, исключение типа `AssertionError` в JVM бросается только при проверке инвариантов в программе при помощи ключевого слова `assert`, поэтому пользовательская программа не должна обрабатывать исключения такого типа. К исключениям, для которых предупреждение выдавать бессмысленно, мы точно так же отнесли `ExceptionInInitializerError`, возникающее при брошенном исключении в статическом инициализаторе класса, и `LinkageError`, которое возникает только при загрузке классов и при специфичных случаях использования Reflection API [10].

В коде на C++ исключение типа `std::bad_alloc` может возникнуть в большом количестве мест, поэтому было решено выделить отдельный подтип предупреждения для этого типа исключений.

В некоторых случаях программист может ожидать, что некоторые типы исключений будут завершать программу аварийно. Такое возможно, например, при первичной обработке входных аргументов – исключение, которое будет брошено при встрече с некорректным аргументом, не вызовет никаких критических ошибок в работе приложения, если не будет поймано. В таких случаях, программист может пометить в функции `main` те исключения, которые он ожидает (при помощи ключевого слова `throws` в случае Java, при помощи аннотации `Throws` в случае Kotlin или при помощи «exception specification» в случае C++). В этом случае детектор не будет выдавать предупреждения для указанных типов исключений.

5. Генерация промежуточного представления

Инструмент *Svace* осуществляет поиск ошибок в два этапа:

- с помощью модифицированного компилятора для соответствующего языка создаётся низкоуровневое промежуточное представление;
- полученное промежуточное представление подаётся на вход анализатору, который преобразует его в унифицированное промежуточное представление (см. разд. 3), для которого выполняется анализ.

Наше промежуточное представление генерируется разными компиляторами: Javac OpenJDK [11], Koltinc [12] и Clang [13].

5.1 Информация об исключениях в промежуточном представлении для C++

Используемый в *Svace* для C++ компилятор Clang использует разное ABI при генерации промежуточного представления для ОС Linux и Windows, в результате промежуточное представление для них существенно отличается.

Промежуточное представление для Linux имеет Itanium ABI [14]. В анализаторе *Svace* мы полностью поддерживаем все инструкции, генерируемые подобным представлением. Для анализа нам важны следующие инструкции:

- `invoke` – чтобы знать, для каких вызовов исключения обрабатываются;
- `landingpad` – для определения типов обрабатываемых исключений[3];
- `resume`, `__cxa_rethrow`, `__cxa_throw` – чтобы знать о брошенном исключении;
- `__cxa_call_unexpected` – чтобы знать, в каких случаях программа может завершиться вызовом `std::terminate`;
- `llvm.eh.typeid.for` – для получения типа исключения, на соответствующий обработчик которого сделан переход.

Для ОС Windows в LLVM используется представление, отличающееся от Itanium ABI [15]. На момент написания статьи, из всех специфичных инструкций данного промежуточного представления в *Svace* поддерживается только инструкция `__CxxThrowException`.

Таким образом, мы решили расширить существующее промежуточное представление *Svace* инструкциями LLVM для обработки исключений. Создание нового промежуточного представления является нетривиальной задачей и не является необходимостью в данном случае.

5.2 Информация об исключениях в байт-коде JVM

В JVM, в отличие от LLVM, нет специализированных инструкций, относящихся к обработке исключений. Для хранения информации об обрабатываемых исключениях в JVM используется структура данных, именуемая «exception table», которая содержит: тип обрабатываемого исключения; отрезок инструкций, на котором данное исключение обрабатывается; смещение инструкции, на которую должен быть осуществлён переход при пойманном исключении.

Мы используем библиотеку ASM [16] для чтения class-файлов JVM, поэтому у нас нет проблем с корректным извлечением данной информации. Тем не менее, большую проблему составляет построение промежуточного представления анализатора на основе этой информации. Так как библиотека ASM читает байт-код сверху вниз, построение ГПУ осуществляется в аналогичном порядке (при этом смещение каждой метки в момент чтения неизвестно). Это приводит к тому, что невозможно для каждой инструкции в момент её обработки определить, приводит ли её выполнение к выходу из текущего обработчика исключений.

Мы решили данную проблему следующим образом. Когда анализатор встречает начало отрезка, обрабатывающего исключения, то он генерирует искусственное ветвление из инструкций `Try (e1, e2, ...})` и `Catch (e1, e2, …)`: первая инструкция просто

[3] Мы не проводим явной трансформации этой инструкции в описанные ранее `try_enter(e1, e2, …)` и `try_exit(e1, e2, …)`. Это можно сделать разными способами, например, расставляя возле каждой инструкции `invoke` пару из `try_enter(e1, e2, …)` и `try_exit(e1, e2, …)`, используя типы исключений, обрабатываемые соответствующим `landingpad`. Но мы решили оставить данную инструкцию в нашем представлении, необходимым образом модифицировав движок анализа.

ведёт на следующую инструкцию байт-кода, а вторая – на первую инструкцию `catch`-блока, обрабатывающего типы исключений `e1, e2, ...`. Первая соответствует ранее описанной инструкции `try_enter(e1, e2, …)`. Вторая инструкция всегда помечается недостижимой и не является путём, через который может пойти программа. Её наличие необходимо лишь для того, чтобы показать анализу точки, в которых заканчивается обработка типов исключений `e1, e2, ...` – этими точками являются непосредственные постдоминаторы базовых блоков, в которых находятся инструкции `Try (e1, e2, ...})` и `Catch (e1, e2, …)`, а также все выходы из метода. Добавляя во все эти точки инструкцию `try_exit(e1, e2, …)`, получим представление, описанное ранее[4].

5.3 Представление вызовов в ГПУ

LLVM-биткод использует инструкции `call` и `invoke` для вызовов функций. Инструкция `call` используется в тех случаях, когда не требуются переходы на базовые блоки, обрабатывающие исключения. Для остальных случаев используется инструкция `invoke`, которая может передавать управление в различные точки программы, в зависимости от того, возникло ли исключение при вызове [17].

Так как в байт-коде JVM нет подобного рода инструкций вызова, которые бы генерировались компиляторами, мы решили использовать похожий подход при построении ГПУ внутри анализатора. Если в `try`-блоке присутствуют инструкции вызова, то мы считаем, что эти вызовы могут сгенерировать любое отслеживаемое `catch`-блоками исключение (а также любое исключение, описанное в объявлении метода при помощи ключевого слова `throws` – вместо обычной инструкции вызова генерируется ветвление, одна ветвь которого соответствует нормальному завершению вызова, а другая предполагает, что возникло исключение, и ведёт в один из `catch`-блоков (если соответствующего `catch`-блока найдено не было, то ветвь ведёт на выход из метода по исключению).

Мы генерируем переходы только для вызовов, исключения от которых обрабатываются в вызывающей функции. Например, в листинге 10 функция `bar`, так же, как и функция `foo`, может бросить исключение. Мы не генерируем явных переходов для этого случая и оставляем задачу точного определения выполненных инструкций анализу.

```
1 try {
2   r = foo(a, b, ...);
3   return r;
4 } catch (e1) {
5   // Catch-block e1
6 } catch (e2) {
7   // Catch-block e2
8 }
9 r = bar(a, b, ...);
10 return r;
```

Листинг 10: Пример кода, обрабатывающего исключения
Listing 10: Example of code with exception handling

[4] Стоит оговориться, что мы не проводим данную трансформацию явно в нашем промежуточном представлении. Вместо этого мы модифицировали анализ из п. 4.2.1, чтобы передаточная функция для инструкции `Catch (e1, e2, …)` не добавляла типы `e1, e2, …` во множество обрабатываемых исключений. Таким образом, пересечение множеств в точке сбора позволяет добиться того же эффекта, который бы дала обработка инструкции `try_exit(e1, e2, …)`.

5.4 Представление `finally`-блоков в JVM

Компиляторы языков Java и Kotlin при генерации `try-catch-finally` конструкций генерируют дублирующийся код, соответствующий коду из `finally`-блока, и добавляют его после всех инструкций, после которых происходит выход из `try-catch-finally` конструкции – это требуется самой семантикой `finally`-блока, так как код из него должен выполняться всегда. Но дублирование `finally`-блоков неудобно тем, что исходный код программы перестаёт взаимно-однозначно отображаться на байт-код – каждой строке исходного кода может соответствовать сразу несколько инструкций из промежуточного представления.

Чтобы избежать ложных срабатываний, вызванных подобной кодогенерацией, мы используем модифицированные компиляторы языков Java и Kotlin, в которых используем старую схему генерации `finally`-блоков при помощи подпрограмм [18] (и их инструкций `jsr` и `ret`), позволяющих генерировать код из `finally`-блока лишь единожды: инструкция `jsr` осуществляет переход в `finally`-блок, сохраняя адрес возврата на стек, а инструкция `ret` возвращается по сохранённому адресу при завершении `finally`-блока. Подробнее о деталях реализации можно прочитать в [19].

Стоит также отметить один интересный нюанс. При использовании конструкции `try-finally` без блоков `catch`, вследствие эвристик, описанных ранее, считается, что никакое исключение не бросается внутри `try`-блока. Поэтому, для более консервативного анализа мы считаем, что при наличии в коде `finally`-блока, инструкции вызова в соответствующем ему `try`-блоке могут бросить исключение самого общего типа – типа `Throwable` (но только при условии, что нет более конкретного типа исключения, бросаемого из `try`-блока).

6. Результаты

Для оценки влияния анализа исключений на результат мы взяли старую версию анализатора *Svace*, в которой была упрощённая обработка исключений. Во-первых, в ней отсутствовали специализированные DF-анализы для исключений. Во-вторых, для JVM использовалась упрощённая схема построения промежуточного представления: считалось, что метод может бросить только те исключения, которые указаны в его определении при помощи ключевого слова `throws`, поэтому соответствующие пути в ГПУ строились только для этих исключений.

Для оценки результатов были выбраны следующие проекты с открытым исходным кодом:

- JVM:
  - операционная система Android 11 [20]; проект содержит более 33 миллионов строк кода на языке Java;
  - компилятор языка Kotlin версии 1.4.10 [12]; на данный момент, это самый крупный проект с открытым исходным кодом на языке Kotlin; проект содержит приблизительно 2 миллиона строк Kotlin кода и 1.1 миллиона строк Java кода;

C++:

  - Kodi версии 17.0a2-Krypton [21]; проект содержит около 650 тысяч строк кода на C++;'
  - операционная система Tizen версии 2.3 [22] размером более 6.6 миллионов строк кода, из которых 1.7 миллионов написаны на C++.

В табл. 1 и 2 приведены группы детекторов, результаты анализа для которых существенно изменились. Строка «Суммарно» учитывает предупреждения, выданные всеми детекторами (в том числе, не перечисленными в таблице отдельно), кроме детектора необработанных исключений – его результаты представлены в табл. 3.

В табл. 1 и 2 представлены только результаты для проектов на языках Java и Kotlin, так как нами не было замечено существенных изменений на проектах на языке C++: на проекте Tizen стало на 19 ложных предупреждений меньше и на 1 истинное предупреждение меньше (при общем количестве в 49777 предупреждений), на проекте Kodi стало на 6 ложных предупреждений больше (при общем количестве в 25104 предупреждения).

Как видно из результатов, использование специального анализа с учётом исключений повышает качество результатов. Такой анализ позволяет существенно сократить количество ложных предупреждений, а также найти немного больше истинных.

Табл. 1. Результаты сравнения версий Svace для Android 11
Table 1. Results of Space versions comparison for Android 11

Группа детекторов	Предупреждений		Истинных		Ложных		Процент изменений
	До	После	Появл.	Исч.	Появл.	Исч.	
Разыменование нулевой ссылки	6267	6097	35	86	34	136	0.81%
Утечка ресурсов	636	782	74	9	42	41	10.06%
Избыточное сравнение	1377	1342	4	11	9	37	1.53%
Недостижимый код	1244	1749	41	18	36	132	9.57%
Суммарно	18103	18289	205	73	197	565	2.76%

Табл. 2. Результаты сравнения версий Svace для Kotlinc-1.4.10
Table 2. Results of Space versions comparison for Kotlinc-1.4.10

Группа детекторов	Предупреждений		Истинных		Ложных		Процент изменений
	До	После	Появл.	Исч.	Появл.	Исч.	
Разыменование нулевой ссылки	2188	1534	11	10	14	667	29.89%
Утечка ресурсов	16	28	7	2	7	0	-12.5%
Избыточное сравнение	31	32	0	0	2	1	-3.23%
Недостижимый код	87	136	30	0	26	7	12.64%
Суммарно	6351	5723	55	7	63	709	10.93%

Табл. 3. Результаты детектора необработанных исключений
Table 3. Results of detector for unhandled exceptions

Проект	Всего	Классифицировано	Истинных	Ложных	Процент истинных
Android 11	155	152	87	65	57.24%
Kotlin-1.4.10	51	46	34	12	73.91%
Kodi	175	155	151	4	97.42%
Tizen-2.3	422	184	183	1	99.46%

7. Похожие работы

В [23] описывается подход расширения статического анализатора Java для Kotlin-кода. Использовался инструмент `SECUCHECK` [24]. Авторы явным образом указывают, что не знают ни одного статического анализатора, поддерживающего глубокий анализ потока данных для языка Kotlin, что косвенно свидетельствует об актуальности нашей работы, так как мы описываем именно такой анализ. Кроме этого, низкоуровневое представление, предлагаемое в нашем подходе, позволяет анализировать Java и Kotlin схожим образом. Хотя

наш подход и предполагает существенные усилия на поддержку языка Kotlin, однако они локализованы в генерации промежуточного представления.

Clang Static Analyzer [25] включает в свой состав анализатор на основе символьного выполнения. К сожалению, мы не смогли получить значимые результаты на реальных проектах, которые можно было бы сравнить с нашим анализатором. Просмотренные срабатывания не относились к коду, связанному с исключениями. Поэтому мы создали синтетические тесты для детектора деления на ноль, чтобы выяснить возможности анализатора. Нам не удалось найти отличия, обусловленные обработкой исключений, кроме ситуации, когда обрабатывается базовый тип от бросаемого. Для таких тестов наша реализация ошибалась, так как мы не поддержали отношения подтипов для классов C++ (см. листинг 11.

```
1 class A {}
2
3 class B : public A {}
4
5 void foo() {
6   throw B();
7 }
8
9 int test() {
10  int x = 0;
11  try {
12    foo();
13  } catch (A &a) {
14    x = 1;
15  }
16  return 100 / x;
17 }
```

В C++ есть множество правил преобразования типов, например исключение типа `long` будет перехватываться блоком `catch(long& e)`, то есть ссылочный тип является подтипом основного типа. Поэтому реализация операции `(e ≤: type)` для C++ является нетривиальной задачей.

В работе [26] описывается подход к анализу исключений для C++. Там также используется предварительная трансформация программы в представление без исключений. В отличии от их подхода, мы используем более низкоуровневое промежуточное представление, которое подходит для анализа не только C++, но и других языков.

Анализатор SharpChecker [27] для языка C# реализован на основе компилятора Roslyn и использует возможности компилятора для построения ГПУ. Реализация анализа исключений [28] в анализаторе имеет не только потоковую чувствительность, но и чувствительность к путям. На наш взгляд чувствительность к путям не интересна для реализации анализов недостижимого кода, так как обработка исключений редко содержит условные выражения. Но добавление чувствительности к путям может быть полезно для детектора необработанных исключений. Например, в примере из листинга 12 на строке 16 не может возникнуть исключение при вызове конструктора, так как передаваемый в него аргумент заведомо не равен `null`. При этом наш детектор выдаёт здесь ложное срабатывание, так как не обладает чувствительностью к путям.

```
1 class Printer {
2     private final String s;
3
4     public Printer(String s) {
5         if (s == null) throw new IllegalArgumentException();
6         this.s = s;
```

```
7      }
8
9      public void print() {
10         System.out.println(s);
11     }
12 }
13
14 class Main {
15     public static void main(String[] args) {
16         new Printer("Hello, world!").print();
17     }
18 }
```

Листинг 12: Ложное срабатывание детектора необработанных исключений
Listing 12: False positive warning by unhandled exceptions detector

Также мы провели сравнение со статическими анализаторами с открытым исходным кодом SpotBugs [29] и Infer [30]. Оба анализатора были запущены на проекте Android 11. При помощи SpotBugs был проанализирован Kotlinc-1.4.10.

SpotBugs обнаруживает следующие типы дефектов, которые ищутся и при помощи *Svace*:

- разыменование нулевого указателя на пути с брошенным исключением;
- утечка ресурсов из-за необработанного исключения;
- объект типа `Lock` захватывается, но не освобождается в случае брошенного исключения.

Результаты сравнения для данных типов предупреждений приведены в табл. 4[5]

В анализаторе Infer присутствует только один тип предупреждения, интересный в данной работе, который имеет и наш инструмент – утечка ресурсов из-за брошенного исключения. Результаты сравнения представлены в табл. 5.

Табл. 4. Результаты сравнения Svace со SpotBugs
Table 4. Results of comparing Svace with SpotBugs

Истинных дефектов найдено	SpotBugs и Svace	49
	Только SpotBugs	2
	Только Svace	106
Ложных выдано	SpotBugs и Svace	10
	Только SpotBugs	57
	Только Svace	95

Табл. 5. Результаты сравнения Svace с Infer
Table 5. Results of comparing Svace with Infer

Истинных дефектов найдено	Infer и Svace	3
	Только Infer	1
	Только Svace	112
Ложных выдано	Infer и Svace	1
	Только Infer	5
	Только Svace	73

Только один из трёх пропущенных истинных дефектов является проблемой анализа исключений. Проблема заключается в том, что *Svace* считает, что нативный метод в языках

[5] Результаты для Svace, представленные в этой и следующей таблицах, не являются до конца полными, так как мы не смогли классифицировать абсолютно все предупреждения, выданные нашим инструментом, из-за большого их количества. Таким образом, реальное количество истинных и ложных предупреждений может быть выше, чем указано в таблицах.

Java и Kotlin не может бросить исключение, хотя в общем случае это неверно. Это приводит к тому, что в примере из листинга 13 наш инструмент не находит проблему с неосвобождённым объектом типа `Lock` из-за брошенного исключения. Тем не менее, SpotBugs, находящий этот дефект, выдаёт большое количество ложных предупреждений, так как считает, что любой метод может кинуть любое исключение. Также недостатком можно назвать то, что наш инструмент выдаёт большое количество ложных предупреждений типа утечки ресурсов, но это проблема не анализа исключений, а самого детектора, которую в будущем планируется исправить.

Несмотря на описанные проблемы, наш подход позволяет находить гораздо больше истинных дефектов, чем инструменты SpotBugs и Infer.

```
1 class Foo {
2   private final ReentrantReadWriteLock lock = new
        ReentrantReadWriteLock();
3   private final ReentrantReadWriteLock writeLock = lock.writeLock();
4
5   private final native void nativeFoo();
6
7   public void foo() {
8     writeLock.lock();
9     nativeFoo();
10    writeLock.unlock();
11  }
12 }
```

Листинг 13: Пример пропущенного дефекта с вызовом нативного метода
Listing 13: Example of the false negative warning with the native method call

8. Заключение

В статье описан подход к статическому анализу программ на языках с обработкой исключений. В рамках работы было предложено промежуточное представление, позволяющее анализировать программы на разных языках программирования, содержащих конструкции для работы с исключениями. Приведены описания легковесных анализов, работа которых позволяет отсеять множество недостижимых путей, возникающих при работе с исключениями. Описан алгоритм работы детектора для поиска необработанных исключений.

Все перечисленные выше результаты были реализованы и протестированы в инструменте статического анализа *Svace* для языков C++, Java и Kotlin. Для каждого языка в статье были указаны особенности генерации промежуточного представления.

Результаты работы позволили повысить общее качество анализа. Также было проведено сравнение наших результатов с другими работами. Предложенное нами решение, в отличие от рассмотренных нами работ, позволяет использовать единое представление для анализа нескольких языков.

В будущем планируется улучшение реализованных анализов, в частности, поддержка вычисления отношения подтипов для языка C++.

Список литературы / References

[1] Common Weakness Enumeration. CWE-703: Improper Check or Handling of Exceptional Conditions. URL: https://cwe.mitre.org/data/definitions/703.html, accessed 28.01.2022.

[2] Иванников В.П., Белеванцев А.А. и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 231-250. DOI: 10.15514/ISPRAS-2014-26(1)-7 / Ivannikov V.P., Belevantsev A.A. et al. Static analyzer Svace for

finding defects in a source program code. Programming and Computer Software, vol. 40, issue 5, 2014, pp. 265-275.

[3] Бородин А.Е., Горемыкин А.В. и др. Поиск уязвимостей небезопасного использования помеченных данных в статическом анализаторе Svace. Труды ИСП РАН, том 33, вып. 1, 2021 г., стр. 7-32. DOI: 10.15514/ISPRAS–2021–33(1)–1 / Borodin A., Goremykin A. et al. Searching for Taint Vulnerabilities with Svace Static Analysis Tool. Programming and Computer Software, vol. 47, issue 6, 2021, pp. 466-481.

[4] Бородин А.Е., Дудина И.А. Внутрипроцедурный анализ для поиска ошибок на основе символьного выполнения. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 87-100. DOI: 10.15514/ISPRAS–2020–32(6)–7 / Borodin A., Dudina I. Intraprocedural Analysis Based on Symbolic Execution for Bug Detection. Programming and Computer Software, vol. 47, issue 8, 2021, pp. 858-865.

[5] Common Weakness Enumeration. CWE-460: Improper Cleanup on Thrown Exception. URL: https://cwe.mitre.org/data/definitions/460.html, accessed 28.01.2022.

[6] Дудина И.А., Кошелев В.К., Бородин А.Е. Поиск ошибок доступа к буферу в программах на языке C/C++. Труды ИСП РАН, том 28, вып. 4, 2016 г., стр. 149-168 / Dudina I., Koshelev V., Borodin A. Statically detecting buffer overflows in C/C++. Trudy ISP RAN /Proc. ISP RAS, vol. 28, issue 4, 2016, pp. 149-168 (in Russian). DOI: 10.15514/ISPRAS-2016-28(4)-9.

[7] Мулюков Р.Р., Бородин А.Е. Использование анализа недостижимого кода в статическом анализаторе для поиска ошибок в исходном коде программ. Труды ИСП РАН, том 28, вып. 5, 2016 г., стр. 145-158 / Mulyukov R.R., Borodin A.E. Using unreachable code analysis in static analysis tool for finding defects in source code. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 5, 2016, pp. 145-158 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-9.

[8] Common Weakness Enumeration. CWE-248: Uncaught exception. URL: https://cwe.mitre.org/data/definitions/248.html, accessed 28.01.2022.

[9] Working Draft, Standard for Programming Language C++. 18.5.1 The std::terminate() function. URL: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf, accessed 12.08.2022.

[10] Oracle Java Documentation: The Java Tutorials. Trail: The Reflection API. URL: https: //docs.oracle.com/javase/tutorial/reflect/index.html, accessed 17.05.2022).

[11] OpenJDK. The Java programming language Compiler Group. URL: https://openjdk.org/groups/compiler/, accessed 12.08.2022.

[12] JetBrains/kotlin. The Kotlin Programming Language. URL: https://github.com/JetBrains/kotlin, accessed 22.04.2022.

[13] Clang: a C language family frontend for LLVM. URL: https://clang.llvm.org/, accessed 12.08.2022.

[14] LLVM Compiler Infrastructure. Exception Handling in LLVM. Itanium ABI Zero-cost Exception Handling. URL: https://llvm.org/docs/ExceptionHandling.html#itaniumabi-zero-cost-exception-handling, accessed 11.08.2022.

[15] LLVM Compiler Infrastructure. Exception Handling in LLVM. Exception Handling using the Windows Runtime. URL: https://llvm.org/docs/ExceptionHandling.html#exceptionhandling-using-the-windows-runtime, accessed 11.08.2022.

[16] ASM. URL: https://asm.ow2.io/, accessed 18.08.2022.

[17] LLVM Compiler Infrastructure. Exception Handling in LLVM. Try/Catch. URL: https://llvm.org/docs/ExceptionHandling.html#try-catch, accessed 11.08.2022.

[18] Lindholm T., Yellin F. et al. The Java® Virtual Machine Specification. Java SE 8 Edition. Exceptions and finally. URL: https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-4.html#jvms-4.10.2.5, accessed 20.12.2021.

[19] Меркулов А.П., Поляков С.А., Белеванцев А.А. Анализ программ на языке Java в инструменте Svace. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 57-74 / Merkulov A.P., Polyakov S.A., Belevantsev A.A. Supporting Java programming in the Svace static analyzer. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 3, 2017. pp. 57-74 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-5.

[20] Android Open Source Project. URL: https://source.android.com/, accessed 22.04.2022.

[21] xbmc/xbmc - Kodi Home Theater Software! URL: https://github.com/xbmc/xbmc, accessed 18.08.2022.

[22] Tizen Docs. URL: https://docs.tizen.org/, accessed 18.08.2022.

[23] Krishnamurthy R., Piskachev G., Bodden E. To what extent can we analyze Kotlin programs using existing Java taint analysis tools? arXiv preprint arXiv:2207.09379, 2022, 12 p.

[24] Piskachev G., Krishnamurthy R., Bodden E. SecuCheck: Engineering configurable taint analysis for software developers. In Proc. of the IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM), 2021, pp. 24—29.

[25] Clang Static Analyzer. URL: https://clang-analyzer.llvm.org/, accessed 17.08.2022.
[26] Prabhu P., Maeda et al. Interprocedural exception analysis for C++. Lecture Notes in Computer Science, vol. 6813, 2011, pp. 583-608.
[27] Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40. DOI: 10.15514/ISPRAS-2016-28(1)-2 / Koshelev V.K., Ignatiev V.N. et al. SharpChecker: Static analysis tool for C# programs. Programming and Computer Software, vol. 43, issue 4, 2017, pp. 268—276.
[28] Belyaev M., Ignatyev V. Exception Analysis for Errors Detection in the SharpChecker Static Analyzer for C#. In Proc. of the 2021 Ivannikov ISPRAS Open Conference, 2021, pp. 8-16.
[29] SpotBugs. Find bugs in Java Programs. URL: https://spotbugs.github.io/, accessed 22.04.2022.
[30] Infer. URL: https://fbinfer.com/, accessed 22.04.2022.

Информация об авторах / Information about authors

Виталий Олегович АФАНАСЬЕВ – студент магистратуры факультета компьютерных наук НИУ ВШЭ, сотрудник ИСП РАН. Сфера научных интересов: компиляторные технологии, статический анализ, JVM-языки.

Vitaly Olegovich AFANASYEV – graduate student at the Faculty of Computer Science, NRU HSE, employee of ISP RAS. Research interests: compiler technologies, static analysis, JVM languages.

Варвара Викторовна ДВОРЦОВА – студентка магистратуры факультета ВМК МГУ, сотрудник ИСП РАН. Сфера научных интересов: компиляторные технологии, статический анализ, анализ Golang.

Varvara Viktorovna DVORTSOVA – graduate student at the Faculty of Computational Mathematics and Cybernetics of Moscow State University, employee of ISP RAS. Research interests: compiler technologies, static analysis, Golang analysis.

Алексей Евгеньевич БОРОДИН — кандидат физико-математических наук, старший научный сотрудник ИСП РАН. Сфера научных интересов: статический анализ исходного кода программ для поиска ошибок.

Alexey Evgenevich BORODIN – PhD, researcher of ISP RAS. Research interests: static analysis for finding errors in source code.

DOI: 10.15514/ISPRAS-2022-34(6)-2

Повышение точности статического анализа за счет учета значений полей класса, имеющих единственное константное значение

[1,2] *В.С. Карцев, ORCID: 0000-0001-7482-0835 <karcev.vs@ispras.ru>*
[1,3] *В.Н. Игнатьев, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[3] *Московский государственный университет им. М.В. Ломоносова, Россия, г. Москва, Ленинские горы д.1*

Аннотация. В данной статье описывается подход, позволяющий с помощью предварительного анализа полей с единственным возможным значением повысить точность работы детекторов статического анализатора уровня символьного выполнения. Помимо этого, детектор предупреждает программиста о забытых модификаторах readonly и о неиспользуемых полях. Детектор был реализован в рамках промышленного статического анализатора SharpChecker для языка C#. Анализ проводится на уровне абстрактного синтаксического дерева, что позволяет снизить затраты времени и ресурсов. Сохраненные значения для ряда полей позволяют использовать конкретные константы вместо символьных значений на этапе символьного выполнения, тем самым повышая точность. В результате удалось заметно улучшить качество некоторых анализаторов, таких как UNREACHABLE_CODE (улучшение на 7.57%) или DEREF_OF_NULL (улучшение на 1.33%), и получить новые срабатывания в случаях с забытым readonly или неиспользуемыми полями. Хорошие результаты позволили включить детектор в основную ветку SharpChecker и предоставить его пользователям. В работе подробно рассмотрен алгоритм работы детектора и приведены примеры срабатываний на наборе ПО с открытым исходным кодом.

Ключевые слова: статический анализ; C#; неиспользуемое поле; забытый readonly; повышение точности; анализ использования полей; единожды инициализированное поле

Для цитирования: Карцев В.С., Игнатьев В.Н. Повышение точности статического анализа за счет учета значений полей класса, имеющих единственное константное значение. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 29-40. DOI: 10.15514/ISPRAS-2022-34(6)-2

Improving the accuracy of static analysis by accounting for the values of class fields that can have only one constant value

[1,2] *V.S. Karcev, ORCID: 0000-0001-7482-0835 <karcev.vs@ispras.ru>*
[1,3] *V.N. Ignatiev, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the RAS,*
25 Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Moscow Institute of Physics and Technology*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[3] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. The paper describes the approach for the improvement of the accuracy of general purpose static symbolic execution analysis of C# sources based on the accounting for the values of class fields that can have only one possible value. In addition, we propose the detector of forgotten readonly modifiers and unused fields, that use data collected by the main analysis. The approach and detectors were implemented as part of the industrial static analyzer SharpChecker. The main analysis is performed at the AST level to reduce time and resource costs. Collected values of the fields are used during symbolic execution phase allowing it to use concrete value instead of symbolic for the subset of class fields. As a result, we managed to noticeably improve the accuracy of some analyzers, such as UNREACHABLE_CODE (improved by 7.57%) or DEREF_OF_NULL (improved by 1.33%) and get new results in cases with forgotten **readonly** or unused fields. Achieved results allow to use analysis and detectors in the main branch of the SharpChecker and make it available to users. The paper considers in detail the algorithm of the detector and provides examples of results on the set of open source software.

Keywords: static analysis; C#; unused field; forgotten readonly; improving accuracy; field usage analysis; once-initialized field

For citation: Karcev V. S., Ignatiev V.N. Improving the accuracy of static analysis by accounting for the values of class fields that can have only one constant value. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 29-40 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-2

1. Введение

В языке C#, помимо привычного модификатора `const`, используется также модификатор `readonly`. В отличие от константных полей, поля с таким модификатором можно инициализировать любым, даже неконстантным значением. Однако инициализация возможна только в конструкторе, что гарантируется компилятором, а далее такое поле будет доступно только для чтения. Так, если такое поле будет инициализироваться конкретным константным значением при любых сценариях использования, то на протяжении всего выполнения это значение останется неизменным. Информация о единственном константном значении такого поля может пригодиться в процессе статического анализа.

Часто встречаются ситуации, когда какое-либо поле, не помеченное `const` или `readonly`, единожды инициализируется каким-либо константным значением, или же все конструкторы присваивают ему единственное значение, которое больше нигде не меняется. Такие поля могут быть причиной некоторых ошибок, так как можно легко забыть, что поле может иметь лишь одно возможное значение. Помимо этого, наличие единственно возможного значения может сигнализировать об ошибках при инициализации такого поля.

При анализе значения таких полей не учитываются, так как анализатор не проверяет, могут ли их значения быть перезаписаны. Это снижает точность результатов. Отсюда вытекает возможность выявлять такие поля и использовать полученные данные при дальнейшем анализе исходного кода.

Кроме того, поиск полей с единственно возможными значениями включает в себя поиск полей, у которых, скорее всего, был забыт модификатор `readonly`. О таких полях

необходимо предупреждать программиста, чтобы он мог исправить возможную ошибку и не допустить некорректного использования поля в дальнейшем.

1.1 Актуальность

В настоящее время язык программирования C# является пятым по популярности согласно рейтингу TIOBE [1]. Он используется для широкого спектра задач, начиная с прикладных программ и заканчивая веб-приложениями. Так, объем проектов на данном языке неизменно растет, что влечет повышение сложности анализа. Таким образом, возникает необходимость производить максимально качественный анализ за минимальное время.

Предлагаемый в статье подход позволяет повысить точность анализа, практически не увеличивая его ресурсоемкость, а также является абсолютно консервативным. В процессе анализа исследуются только те поля, которые никак не могут изменяться во время работы программы. Авторам неизвестно о других инструментах, реализующих указанный алгоритм повышения точности за счет анализа полей, имеющих лишь одно константное значение.

1.2 Мотивационный пример

В качестве ошибочной ситуации, обнаруживаемой предлагаемым подходом, можно привести следующий упрощенный пример.

Допустим, в некоторой библиотеке создан класс логгера, приведенный на листинге 1. Разумно было бы объявить поле `Log.LogFileName` с модификатором `readonly`, так как изменение данного поля во время работы программы не подразумевается в логике библиотеки.

```
1 public class Log
2 {
3     private string LogFileName = "log.txt";
4
5     public string GetName()
6     {
7         return LogFileName;
8     }
9 }
```

Листинг 1. Пример класса с забытым readonly
Listing 1. A class example with forgotten readonly

На листинге 2 приведен пример использования данного класса сторонним разработчиком в своем проекте. Так как пользователь не осведомлен о том, что поле не может меняться, он добавляет бесполезную проверку, что приводит к проблеме недостижимого кода.

```
1 public class User
2 {
3     var Logger = new Log();
4
5     public Register()
6     {
7         if (Logger.GetName() != "log.txt")
8         {
9           ...
10        }
11
12        else
13        {
14          ...
```

```
15          }
16      }
17  }
```

Листинг 2. Ошибочное использование класса
Listing 2. Wrong class usage

В приведенном примере опущенный модификатор `readonly` привел к тому, что сторонний программист написал код, который никогда не будет исполнен. Нахождение таких полей, и выяснение их значений могут повысить точность поиска ошибок различных типов, обнаруживаемых методом символьного исполнения, например, `UNREACHABLE_CODE`, `DEREF_OF_NULL`.

1.3 Цели и задачи

Цели данной работы перечислены ниже:

1) повышение точности анализа за счет использования неизменных значений полей классов;
2) определение значений полей, которые инициализируются единожды и не меняются;
3) поиск полей с забытым `readonly`.

Результатом работы должен стать анализатор, который будет

- искать поля классов, в которых, вероятно, был забыт модификатор доступа `readonly` и предупреждать о них программиста;
- выяснять, существует ли единственно возможное значение этого поля при любых вариантах использования класса и запоминать это единственное значение;
- предоставлять другим анализаторам информацию о полях, имеющих единственное возможное значение.

Помимо прочего, так как анализатор будет строить результат на основе статистики использования этого поля, можно будет дополнительно генерировать предупреждения о забытых `readonly` и неиспользуемых полях.

1.4 Существующие решения

В большинстве промышленных инструментов, таких как Svace [2-4] или SonarQube [5], значения полей, инициализированных единожды, не используются в процессе анализа. Срабатывания, найденные с помощью информации, собранной новым анализатором, не удалось воспроизвести в других статических анализаторах. Таким образом, можно предположить, что в других продуктах значения единожды инициализированных полей не учитываются в анализе.

В основном статические анализаторы предполагают, что поля классов, не объявленные как `const`, не могут иметь константное значение или же анализируют поведение лишь локальных переменных, не исследуя точное поведение объектов. Так, статический анализатор Pagai [6] анализирует только промежуточное представление кода LLVM, в связи с чем не может обнаружить поле с единственным константным значением.

В свою очередь, поиск забытых `readonly` и неиспользуемых полей, являющийся побочным результатом проекта, реализован во многих решениях. Детекторы неиспользуемых полей и забытых модификаторов `readonly` включены во многие IDE, включая, например, Visual Studio [7, 8]. Однако существующие решения в основном являются составляющими частями небольших продуктов, предназначенных в основном для рефакторинга кода [9] и выдачи предупреждений «на лету». Так, существующие решения могут обнаруживать лишь простейшие ошибки из-за специфики реализации.

2. Реализация

Анализатор был разработан в рамках промышленного инструмента SharpChecker [10], предназначенного для поиска ошибок в объемных проектах методами статического анализа.

2.1 Схема работы SharpChecker

В первую очередь SharpChecker, используя инфраструктуру Roslyn [11], перехватывает компиляцию и собирает решение. Далее запускается этап анализа, состоящий, по большей части, из анализа абстрактного синтаксического дерева и символьного исполнения.

На этапе анализа синтаксического дерева можно сделать выводы о корректности исходного кода, найти синтаксические ошибки.

На этапе символьного исполнения производится масштабируемый межпроцедурный анализ, чувствительный к потоку управления [12]. Символьное исполнение в SharpChecker допускает ложные срабатывания и пропущенные ошибки. Его цель – найти максимальное количество ошибок за минимальное время при заданном проценте истинных срабатываний. На данном этапе анализируется граф потока управления, возможные значения переменных и выполнимость условий. С помощью этого метода можно найти ошибки, связанные, например, с недостижимым кодом или использованием удаленного ресурса.

Так как результаты работы предлагаемого анализатора должны использоваться на уровне символьного исполнения, то они вычисляются раньше - на уровне анализа синтаксического дерева. В SharpChecker при анализе синтаксического дерева используются механизмы Roslyn. Конкретно, для поиска информации о полях необходимо иметь «идентификатор» каждого поля. В Roslyn для этого используется `IFieldSymbol`, являющийся представлением поля в таблице символов. Однако в различных проектах одно и то же поле будет иметь различные `IFieldSymbol`. Для этих целей можно использовать реализованный в SharpChecker `IFieldSymbolId`, представляющий собой «полное имя» поля, включающее в себя пространство имен и класс, которому принадлежит поле. В отличие от встроенного в Roslyn `IFieldSymbol`, он остается одним и тем же в различных проектах, что позволяет производить анализ решений, состоящих из множества проектов и учитывать использования поля в каждом из них.

Используя эти возможности, можно реализовать анализатор использования полей классов, запоминающий, при наличии, единственное константное значение, возможное для этого поля.

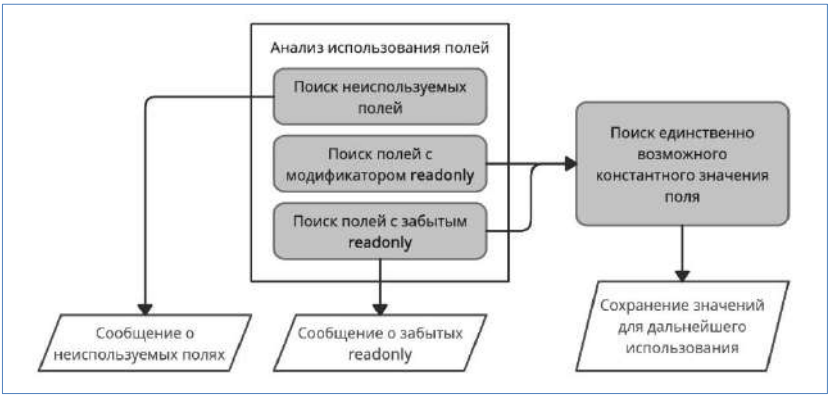

Рис. 1: Схема работы анализа
Fig.. 1: Scheme of the analysis

2.2 Схема работы алгоритма

Несмотря на то, что все этапы анализа полей происходят параллельно, формально их можно разбить на последовательные шаги. Тогда принцип работы анализатора можно представить схемой на рис. 1.

Фактически механизм реализован с помощью механизмов Roslyn, позволяющих создать обработчики для каждого типа узлов синтаксического дерева. Схематически алгоритм изображен на рис. 2.

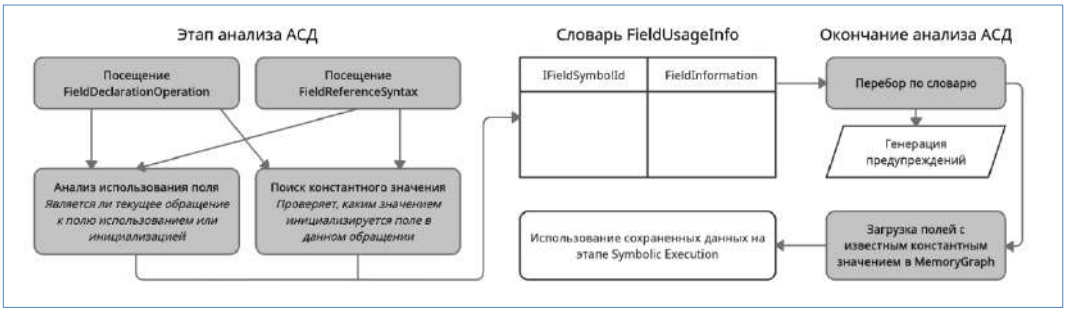

Рис. 2: Схема работы алгоритма
Fig. 2: Scheme of the algorithm

Как видно из схемы, каждый из обработчиков вызывает две функции-анализатора, первая из которых определяет, является обращение к полю использованием или инициализацией. Этот анализатор собирает информацию, необходимую для вывода о необходимости добавить к объявлению поля модификатор `readonly` и о том, что поле не используется. Второй анализатор занимается сбором данных о том, какое значение присваивается полю в каждой из инициализаций, переданных на обработку.

Оба анализатора хранят информацию в словаре, ключом в котором является `IFieldSymbolId` анализируемого поля, а все информация о нем содержится в объекте класса `FieldInformation`. Данный класс предназначен для удобного хранения информации об использовании поля, местах инициализаций и возможных значениях.

В конце этапа анализа абстрактного синтаксического дерева вызывается метод, создающий диагностики для всех полей, которые должны быть `readonly` и для полей, которые не используются. Параллельно вся информация о полях, имеющих лишь одно возможное значение, переносится в `MemoryGraph`, из которого на этапе символьного исполнения можно будет извлечь их значение.

2.2.1 Выбор кандидатов

Для начала необходимо отсеять поля, которые гарантированно не могут иметь единственного константного значения. В первую очередь это поля, доступные для изменения извне. Такое поле, не имея ни одного присваивания в видимой области кода, может быть изменено сторонним разработчиком в его коде. Помимо этого, не могут иметь единственное значение те поля, которые получают различные значения при различных условиях.

Подытожим: наличие единственно возможного константного значения возможно для полей, удовлетворяющих одному из следующих условий:

- поле имеет модификатор доступа readonly, а значит, не может иметь инициализаций вне конструкторов, что гарантируется компилятором;
- поле недоступно для изменения вне анализируемого кода, а все инициализации в области видимости располагаются в конструкторах.

Отдельно необходимо рассмотреть второй вид кандидатов. Недоступными для внешнего редактирования будут поля, имеющие модификатор доступа `private` или `internal`.

Помимо этого, недоступны извне будут и поля классов, имеющих соответствующий уровень доступа или же содержащиеся внутри приватных классов. Можно заметить, что такие поля, недоступные для внешнего редактирования и имеющие инициализации только внутри конструкторов, полностью удовлетворяют определению `readonly`-полей.

Так, о схожести такого поля с `readonly` необходимо сообщить программисту. Для этого был создан новый тип ошибки – `FORGOTTEN_READONLY`.

2.2.2 Определение полей с забытым readonly

Для поиска полей с забытым `readonly` необходимо проанализировать каждое обращение к исследуемому полю и подсчитать количество инициализаций внутри и вне конструкторов. Так, если вне конструкторов не найдено ни одной инициализации и выполняется одно из условий:

- поле имеет модификатор доступа private или internal;
- класс, к которому принадлежит поле, является приватным;
- класс, к которому принадлежит поле, вложен в приватный класс,

то данное поле не может изменяться извне и у него мог быть забыт модификатор `readonly`.

Вывод о необходимости объявить поле как `readonly` возможно сделать только после окончания анализа всего исходного кода. Таким образом, необходимо хранить информацию по всем полям, в том числе и по тем, для которых достоверно известно, что они меняются вне конструкторов. Таким образом, после завершения анализа всего исходного кода для каждого поля можно будет сделать вывод о необходимости добавления модификатора `readonly`.

2.2.3 Поиск неиспользуемых полей

Поиск неиспользуемых полей аналогичен поиску полей с забытым `readonly`, однако необходимо подсчитать количество использований этого поля, результат которых используется в дальнейшем. Так, например, использование поля для изменения собственного значения (пример приведен на листинге 3) можно не считать использованием.

```
1 class Counter
2 {
3     private int CValue = 0;
4
5     private void Increase(int value)
6     {
7         CValue = CValue + value;
8     }
9 }
```

Листинг 3. Использование поля для изменения собственного значения
Listing 3. Using a field to change its own value

Этим примером можно проиллюстрировать использование поля для изменения его же значения.

2.2.4 Поиск единственного константного значения

Одновременно с анализом на забытое readonly для каждого поля необходимо проводить анализ на возможное константное значение. Так, каждый раз, встречая инициализацию поля, необходимо сверять новое значение с сохраненным для данного поля. Возможны следующие случаи.

- **Для поля не сохранено никакого значения.** В таком случае необходимо запомнить найденное значение в качестве известного для данного поля.
- **Для поля уже сохранено отличное от текущего значение.** В данном случае можно

понять, что поле принимает различные значения в разных сценариях использования. Для такого поля невозможно судить о его значении.

- **Для поля уже сохранено такое же значение.** В данном случае не нужно делать каких-либо дополнительных действий.

Помимо этого, если встречена инициализация неконстантным значением, можно сделать вывод, что поле не может иметь единственно возможного константного значения.

В случае, если поле не инициализируется при объявлении, в качестве известного константного значения будет запомнено значение по умолчанию для данного типа данных. Таким образом можно будет обработать ситуации, когда в некоторых конструкторах отсутствует инициализация поля, а в других оно инициализируется значением по умолчанию для типа.

3. Результаты

Тестирование производилось на наборе ПО из 21 проекта с открытым исходным кодом, имеющих суммарный размер 6 млн. строк кода. Анализ выполнялся на сервере с 8-ядерным процессором Intel(R) Core(TM) i7-6700 и 32 Gb RAM.

После интеграции анализатора время анализа уменьшилось на 0.2% и составило 1 час, 16 минут и 51 секунду. Незначительное уменьшение времени анализа можно объяснить случайной погрешностью. Таким образом, созданный анализатор не влияет на итоговую производительность SharpChecker.

3.1 Краткий обзор

После интеграции решения в SharpChecker было получено улучшение в работе других анализаторов, в частности:

- заметное улучшение (+7.57%) работы `UNREACHABLE_CODE`: из 68 новых срабатываний 66 оказались истинными, 1 ложным; при этом ушло 10 ложных срабатываний.
- некоторое улучшение (+1.33%) в работе `DEREF_OF_NULL`: были получены 3 новых истинных срабатывания.
- незначительное улучшение в работе анализатора `DEREF_AFTER_AS`.

Помимо улучшения работы прочих анализаторов, было найдено множество ошибок `FORGOTTEN_READONLY` и `UNUSED_FIELD`.

Улучшение работы связано с тем, что значения полей, имеющих единственное константное значение, начали учитываться, даже если неизвестно, каким образом создавался объект.

3.2 Демонстрация улучшения работы сторонних анализаторов. Примеры новых срабатываний

3.2.1 UNREACHABLE_CODE

Рассмотрим срабатывание, обнаруженное в проекте OpenSim версии 0.9.0.0 в файле AnimationSet.cs в строке 135 (листинг 4).

```
1 public class AnimationSet
2 {
3     private bool m_parseError = false;
4     ...
5
6     public Byte[] ToBytes()
7    {
8         if (m_parseError) // UNREACHABLE_CODE
9        {
```

```
10              ...
11          }
12          ...
13      }
14  }
```

Листинг 4. Недостижимый код в проекте OpenSim
Listing 4. Unreachable code in the OpenSim project

Здесь приватное поле `AnimationSet.m_parseError` при объявлении инициализируется значением `false`, однако больше нигде не меняется внутри класса. Кроме того, из-за того, что поле объявлено как приватное, его нельзя изменить вне класса `AnimationSet`. Таким образом, данное поле может иметь лишь одно значение – `false`.

Далее данное поле используется в методе `ToBytes()`, который меняет свое поведение в зависимости от значения этого поля. Однако имея информацию о том, какое значение единственно возможно для данного поля, можно сделать вывод, что блок кода, обрамленный условием, будет недостижим.

3.2.2 DEREF_OF_NULL

Приведенное на листинге 5 срабатывание было найдено в проекте Lucene.Net [13] версии 4.8.0 в строке 535 файла TestRuleSetupAndRestoreClassEnv.cs.

```
1  internal sealed class TestRuleSetupAndRestoreClassEnv
2  {
3      ...
4
5      internal HashSet<string> AvoidCodecs;
6
7      ...
8
9      private bool ShouldAvoidCodec(string codec)
10     {
11         return AvoidCodecs.Count > 0 && // DEREF_OF_NULL
12             AvoidCodecs.Contains(codec);
13     }
14 }
```

Листинг 5. Разыменование null в проекте Lucene.Net
Listing 5. Null dereferencing in the Lucene.Net project

В данном примере приватное поле `AvoidCodecs` ни разу не инициализировано внутри области видимости, а значит, имеет значение по умолчанию для ссылочного типа – `null`.

Таким образом, при обращении к данному полю возникает проблема разыменования `null`.

3.3 Примеры срабатываний

3.3.1 FORGOTTEN_READONLY

В качестве примера можно рассмотреть фрагмент кода 6 из проекта BobBuilder версии 1.0.0.42 (листинг 6). Ошибка была обнаружена в файле Actions.cs в строке 15.

```
1  namespace BobBuilder.Actions
2  {
3      class CloseTag : AbstractEditAction
4      {
5          IEditAction _OldAction = null; // FORGOTTEN_READONLY
6
```

```
7          public CloseTag( IEditAction oldAction )
8          {
9              _OldAction = oldAction;
10          }
11          ...
12          }
13      ...
14  }
```

Листинг 6. Забытое readonly в проекте BobBuilder
Listing 6. Forgotten readonly in the BobBuilder project

В данном примере анализатор определил поле `_OldAction` как поле с забытым модификатором `readonly`. Действительно, данное поле присваивается лишь дважды – при объявлении и внутри конструктора. Так как данное поле имеет уровень доступа по умолчанию (`private`), то оно недоступно для редактирования вне анализируемого кода.

Поэтому данное поле, вероятно, должно быть помечено как `readonly`.

3.3.2 UNUSED_FIELD

Срабатывание в проекте Spartacus версии 0.45.1, обнаруженное в файле ChainSubRule.cs в строке 7 (листинг 7).

```
1  using System;
2
3  namespace PDFjet.NET {
4      class ChainSubRule {
5       ...
6
7       int inputGlyphCount; // UNUSED_FIELD
8       ...
9      }
10 }
```

Листинг 7. Неиспользуемое поле в проекте Spartacus
Listing 7. Unused field in the Spartacus project

В результате работы поле `inputGlyphCount` было помечено как неиспользуемое, так как класс, которому оно принадлежит, имеет уровень доступа `internal`, а внутри сборки использований поля не было найдено. Соответственно, данное поле не используется в анализируемом коде и не может быть использовано в коде, использующем данную библиотеку.

4. Заключение

Изначально значения полей с единственно возможным константным значением не учитывались в анализе. С помощью реализованного анализатора удалось добиться улучшения работы SharpChecker, были найдены новые ошибки. Таким образом, результаты анализа единожды инициализированных полей могут использоваться для повышения точности других анализаторов. В результате, реализацией было доказано, что учет таких полей возможен без увеличения затрат и полезен при анализе больших проектов.

Помимо прочего, в процессе удалось найти поля с, вероятно, забытым модификатором доступа readonly и неиспользуемые поля.

В связи с улучшением результатов работы SharpChecker патч был включен в основную ветку инструмента, и его результаты теперь могут использоваться у конечных пользователей.

Список литературы / References

[1] Tiobe index for ranking the popularity of programming languages, 2022. URL: https://www.tiobe.com/tiobe-index.

[2] Иванников В.П., Белеванцев А.А. и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 231-250. DOI: 10.15514/ISPRAS-2014-26(1)-7 / Ivannikov V.P., Belevantsev A.A. et al. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, issue 5, 2014, pp. 265-275.

[3] Аветисян А., Бородин А. Механизмы расширения системы статического анализа svace детекторами новых видов уязвимостей и критических ошибок. Труды ИСП РАН, том 21, 2011 г., стр. 39-54 / Avetisyan A., Borodin A. Mechanisms for extending the system of static analysis Svace by new types of detectors of vulnerabilities and critical errors. Trudy ISP RAN/Proc. ISP RAS, vol. 21, 2011, pp. 39-54 (in Russian).

[4] Аветисян А., Белеванцев А. и др. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ. Труды ИСП РАН, том 21, 2011 г., стр. 23-38 / Avetisyan A., Belevantsev A. et al. Using static analysis for finding security vulnerabilities and critical errors in source code. Trudy ISP RAN/Proc. ISP RAS, vol. 21, 2011, pp. 23-38 (in Russian).

[5] Lenarduzzi V., Lomio F. et al. Are SonarQube rules inducing bugs? In Proc. of the IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 501−511.

[6] Henry J., Monniaux D., Moy M. Pagai: a path sensitive static analyser. Electronic Notes in Theoretical Computer Science, vol. 289, 2012, pp. 15-25.

[7] Detecting forgotten readonly in visual studio. Available at: https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/style-rules/ide0044, accessed 20.08.2022.

[8] Detecting unused class members in visual studio. Available at: https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/style-rules/ide0051, accessed 20.08.2022.

[9] Atkinson D.C., King T. Lightweight detection of program refactorings. In Proc. of the 12th Asia-Pacific Software Engineering Conference (APSEC′05), 2005, 8 p.

[10] Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40. DOI: 10.15514/ISPRAS-2016-28(1)-2 / Koshelev V.K., Ignatiev V.N. et al. SharpChecker: Static analysis tool for C# programs. Programming and Computer Software, vol. 43, issue 4, 2017, pp. 268−276.

[11] dotnet/roslyn: The Roslyn .NET compiler provides C# and Visual Basic languages with rich code analysis APIs. Available at: https://github.com/dotnet/roslyn, accessed 23.10.2021.

[12] Baldoni R., Coppa E. et al. A survey of symbolic execution techniques. ACM Computing Surveys (CSUR), vol. 51, issue 3, 2018, pp. 1-39.

[13] Lucene.NET 4.8.0. Available at: https://lucenenet.apache.org, accessed 23.10.2021.

Информация об авторах / Information about authors

Вадим Сергеевич КАРЦЕВ – студент бакалавриата Физтех-школы Радиотехники и Компьютерных Технологий МФТИ, сотрудник ИСП РАН. Научные интересы: компиляторные технологии, статический анализ программ, статическое символьное выполнение, поиск дефектов в исходном коде.

Vadim Sergeevitch KARCEV is a bachelor’s student at the Department of Radio Engineering and Computer Technologies of MIPT, an employee of the ISP RAS. Research interests: compiler technologies, static program analysis, static symbolic execution, defect search in source.

Валерий Николаевич ИГНАТЬЕВ, кандидат физико-математических наук, старший научный сотрудник ИСП РАН, доцент кафедры системного программирования факультета ВМК МГУ. Научные интересы включают методы поиска ошибок в исходном коде ПО на основе статического анализа.

Valery Nikolayevich IGNATYEV, PhD in computer sciences, senior researcher at Ivannikov Institute for System Programming RAS and associate professor at system programming division of CMC faculty of Lomonosov Moscow State University. He is interested in techniques of errors and vulnerabilities detection in program source code using static analysis.

DOI: 10.15514/ISPRAS-2022-34(6)-3

Поиск использований освобожденного ресурса в исходном коде на языке C# методами статического анализа

[1,2] *У.В. Тяжкороб, ORCID: 0000-0002-2375-6842 <tsiazhkorob@ispras.ru>*
[1,3] *В.Н. Игнатьев, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>*
[1,3] *А.А. Белеванцев, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[3] *Московский государственный университет им. М.В. Ломоносова, Россия, г. Москва, Ленинские горы д.1*

Аннотация. В данной работе описывается масштабируемый детектор для поиска использований освобожденного ресурса в исходном коде на основе статического символьного выполнения. Данный детектор выполняет межпроцедурный анализ, чувствительный к потоку управления и контексту вызовов. Детектор реализован в рамках промышленного инструмента SharpChecker, его точность (около 70% истинных срабатываний) позволяет включить его в число основных детекторов и предоставить функционал конечным пользователям. В работе рассматривается алгоритм детектора, адаптированный для SharpChecker. Также представлены результаты тестирования детектора на наборе ПО с открытым исходным кодом и примеры срабатываний на реальных проектах.

Ключевые слова: статический анализ; символьное выполнение; поиск ошибок; освобожденный ресурс

Для цитирования: Тяжкороб У.В., Игнатьев В.Н., Белеванцев А.А. Поиск использований освобожденного ресурса в исходном коде на языке C# методами статического анализа. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 41-50. DOI: 10.15514/ISPRAS-2022-34(6)-3

Благодарности. Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект №20-01-00581 А.

Detection of uses of disposed resources in C\# source code using static analysis

[1,2] *U.V. Tsiazhkorob, ORCID*: *0000-0002-2375-6842* <tsiazhkorob@ispras.ru>
[1,3] *V.N. Ignatiev, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>*
[1,3] *A.A. Belevantsev, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the RAS,*
25 Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Moscow Institute of Physics and Technology*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[3] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. The paper is devoted to the scalable approach for the detection of uses of disposed resources in C# source code, that is based on static symbolic execution. The resulting detector is implemented as a part of an industrial SharpChecker, that performs a scalable inter-procedural path-, and context-sensitive analysis. The evaluation of the developed detector shows 70% true positive ratio allowing it to include to the standard set of detectors and provide functionality to users. The paper describes a detection algorithm that takes into account the limitations imposed by the existing infrastructure of SharpChecker, its evaluation on the set of open-source programs containing 6 mln LOC and some examples of found errors in real projects.

Keywords: static analysis; symbolic execution; use-after-free; bug detection; disposed resource

For citation: Tsiazhkorob U.V., Ignatiev V.N., Belevantsev A.A. Detection of uses of disposed resources in C# source code using static analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 41-50 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-3

Acknowledgments. This work was supported by the Russian Foundation for Basic Research, project №20-01-00581 A

1. Введение

1.1 Ресурсы в языке C#

В языке программирования C# существует два типа ресурсов: управляемые и неуправляемые. Память, выделяемая под управляемые ресурсы, контролируется сборщиком мусора.

Когда количество ссылок на такой ресурс становится нулевым, сборщик мусора автоматически освобождает занимаемую этим ресурсом память. Неуправляемые ресурсы – все, за чем не следит сборщик мусора. Такие ресурсы необходимо явно освобождать после использования. Наиболее распространенные из них: файлы, сетевые подключения, подключения к базам данных и прочее.

`IDisposable` – встроенный .NET интерфейс – механизм высвобождения неуправляемых ресурсов, предлагаемый Microsoft. Он предназначен для детерминированного освобождения неуправляемых ресурсов любых типов, владеющих как управляемыми, так и неуправляемыми ресурсами, а также типов, унаследованных от класса, реализующего `IDisposable`.

Данный интерфейс предоставляет метод `Dispose()`, с помощью которого производится освобождение ресурса. Также в языке C# существует конструкция `using`, которая сама будет обрабатывать вызов `Dispose()`.

Если не освободить неуправляемые ресурсы, ошибка может не проявиться при тестировании и долгое время может не давать о себе знать во время работы программы, но в некоторый момент она проявится, например, при попытке записать данные в освобожденный ресурс, и тогда информация может быть безвозвратно утеряна.

1.2 Актуальность

В наше время проекты становятся больше, поэтому в них сложнее заметить ошибки, некоторые из которых могут привести к проблемам во время исполнения программы. Кроме того, чем раньше на этапе разработки будет обнаружена ошибка, тем дешевле обойдется ее исправление. Чтобы систематически находить ошибки до запуска программы, используются статические анализаторы. Учитывая то, что ряд ошибок сложно обнаружить тестированием, преимущество статического анализа [2, 3] в том, что ошибки находятся сразу, еще до запуска программы, не требуя подбора входных данных для воспроизведения. Это достигается за счет того, что методы статического анализа позволяют обрабатывать сразу все пути выполнения программы.

Ошибка использования освобожденного ресурса, как и многие другие, сложно обнаруживается на тестах, может проявляться по-разному в разных окружениях, зачастую приводя к аварийному завершению программы или потере данных. Для поиска таких ошибок в исходном коде может использоваться статический анализ. В данной работе предлагается детектор использования неуправляемых ресурсов после их освобождения на основе статического символьного выполнения [1] для исходного кода на языке C#. Метод символьного выполнения обеспечивает чувствительность к путям и контексту вызовов, что важно для снижения доли ложных срабатываний. Необходимость реализации данного детектора объясняется высокой популярностью и востребованностью языка C#. Согласно рейтингу TIOBE [4], этот язык является пятым по популярности в мире.

1.3 Мотивационный пример

Рассмотрим упрощенный пример, основанный на реальной проблеме в коде на языке C# (листинг 1).

```
1  class Test
2  {
3      public abstract class ImgClassAbs : IDisposable
4      {
5          Bitmap Img = new Bitmap(100, 100);
6
7          public abstract void Abort();
8
9          public void Dispose()
10         {
11             Img.Dispose();
12         }
13         public void Use()
14         {
15             Img.Clone(); // использование ресурса
16         }
17     }
18     public class ImgClassDisp : ImgClassAbs
19     {
20         public override void Abort()
21         {
22             Dispose(); // освобождение ресурса
23         }
24     }
25     public class ImgClassNotDisp : ImgClassAbs
26     {
```

```
27          public override void Abort() { }
28      }
29      public abstract class TestClass
30      {
31          public void Func(ImgClassAbs elem)
32          {
33              if (elem != null)
34              {
35                  elem.Abort(); // освобождение ресурса
36                  elem.Use(); // USE_AFTER_DISPOSE:
          Resource <elem.Img> was already disposed
37              }
38          }
39      }
40  }
```

Листинг 1. Мотивационный пример
Listing 1. Motivational example

Проблема проявляется в методе `Func()`. Метод `Abort()`, вызванный для ресурса `elem` является абстрактным методом, соответственно он имеет несколько реализаций. Можно не заметить, что в одной из этих реализаций вызывается метод `Dispose()`, как показано в примере. Таким образом, ресурс `elem.Img` может быть освобожден в методе `Abort()`. Далее следует вызов метода, использующей `elem.Img`. В таком случае в процессе исполнения возникает исключение. Именно такие ситуации и находит реализованный детектор.

1.4 Существующие решения

Для языков программирования с ручной работой с памятью существуют детекторы для поиска уязвимостей, связанных с некорректным использованием динамической памяти в процессе работы программы [5-8]. Более того, существуют и статические `USE-AFTER-FREE` детекторы [9-11], выполняющие ту же задачу, что и описываемый, однако для других языков программирования.

Существуют разные подходы к поиску использований освобожденных ресурсов, имеющие различные преимущества и недостатки. Описываемый в статье `USE-AFTER-DISPOSE` детектор реализует межпроцедурный, чувствительный к контексту, путям и потокам анализ на основе символьного выполнения. Близким аналогом к нему является реализация `USE-AFTER-FREE` детектора в SVACE [5]. Также существует подход на основе анализа помеченных данных, например, IRBIS [12], преимуществом которого является возможность обнаруживать больше ошибок, ценой значительной доли ложных предупреждений. Оба инструмента поддерживают языки программирования C и C++. В свою очередь, у языка C#, например, нет указателей, и, как следствие, появляются проблемы в анализе псевдонимов.

В C-подобных языках освобождение памяти происходит в основном при помощи функции стандартной библиотеки `free()`, применимой только для указателей и имеющей поведение, строго определенное реализацией аллокатора. В случае языка C# рассматривается метод `Dispose()`, предоставляемый интерфейсом `IDisposable` и имеющий различные реализации для различных классов.

Для языка C# существуют детекторы только для обнаружения возможных утечек ресурсов [13]. В связи с этим был создан детектор, исследующий исходный код на наличие использований освобожденных ресурсов.

2. Описание схемы работы детектора

2.1 Общая схема работы SharpChecker

SharpChecker, в рамках которого реализован рассматриваемый детектор, использует часть инфраструктуры Roslyn [14], отвечающую за разбор файлов проектов, решений, для работы с файлами системы сборки, а также компиляции исходного кода программы.

В начале анализа происходит построение АСД (абстрактного синтаксического дерева) с помощью Roslyn. При этом также строится статический граф вызовов и производится поиск синтаксических ошибок.

Далее, при обходе графа вызовов от листьев к корню, часть собираемой информации сохраняется в резюме анализируемого метода для дальнейшего ее использования при анализе методов, вызывающих данный.

Затем происходит построение графа потока управления для каждого метода с помощью графа вызовов, АСД и таблицы символов. Вершинами графа потока управления являются базовые блоки (ББ) – последовательности следующих одна за другой инструкций промежуточного представления кода. После, используя собранную информацию, обработчик завершения анализа выдает предупреждения.

И, наконец, работает чувствительный к путям и контексту анализ. При обработке вызовов методов он использует информацию, полученную из их резюме, при этом объединяет состояния в точке слияния. Состояние в точке входа в анализируемый метод параметризуется с помощью символьных переменных. То есть для каждого параметра метода вводится уникальный идентификатор `ValueId`. Таким образом, есть возможность представить результаты выполнения метода в виде выражений над символьными переменными.

В случае, когда использование какого-либо объекта достигается несколькими его определениями, вводится функция объединения значений или ϕ-функция. В этом случае его `ValueId` также будет представлен функцией объединения `ValueIds` всех достигающих этого использования значений объекта.

Для хранения информации в рамках анализа базового блока используется контекст. При переходе от анализа одного ББ к анализу следующего ББ, содержимое контекста первого вливается в контекст второго. Правила, по которым происходит перенос информации, задаются атрибутом `Join`.

2.2 Внутрипроцедурный анализ

Существует множество ситуаций, воспроизводящих описываемую проблему. Рассмотрим сначала самую простую из них. Пусть в коде не предполагалось дальнейшего использования определенного ресурса и его освободили в целях сохранения памяти, но потом все же он где-то понадобился, а про его освобождение забыли.

Сохраняя `ValueId` освобожденных ресурсов, при последующих использованиях можно будет сделать вывод о наличии ошибки. Если мы рассматриваем только внутрипроцедурный анализ, то хранить `ValueId` ресурсов нужно только до конца метода. Для этого будем использовать контекст базового блока ГПУ.

Далее рассмотрим ситуацию, когда объект удаляется и позднее используется при определенных условиях, например, в ветвлении `if-else`. В таком случае ошибка произойдет тогда и только тогда, когда условия освобождения и использования пересекаются, то есть после освобождения непременно произойдет использование ресурса. Соответственно, в контексте, помимо `ValueId`, будем сохранять условия освобождения ресурса.

Если после освобождения в переменную присваивается ссылка на новый или существующий объект, то при поиске в списке освобожденных ресурсов `ValueId` рассматриваемого ресурса, проверяем, является ли он освобожденным. Если объект «обновляется» при всех условиях, при которых он удаляется, то его необходимо убрать из списка освобожденных ресурсов.

2.3 Межпроцедурный анализ

Если освобождение некоторого ресурса произошло в вызванном методе, то его нельзя обнаружить последовательным внутрипроцедурным анализом. Тогда, будем получать информацию об освобожденных ресурсах из результатов анализа вызываемого метода. Однако контекст удаляется в конце метода, значит в этот момент нужно переносить информацию в резюме. А при вызове метода переносим содержимое его резюме в контекст вызывающего метода.

Возможны ситуации, в которых использование объекта так же происходит в вызове какого-либо метода. Тогда для анализа требуется иметь возможность получить информацию об использованиях в вызываемом методе. Таким образом, помимо удалений, в контекст будем сохранять и использования, чтобы потом, перенеся их в резюме, иметь к ним доступ из вызывающего метода.

Аналогично будем поступать в случае «обновления» ресурса.

Более того, при переносе из резюме вызываемого метода в контекст вызывающего необходимо учитывать условие, при котором происходит вызов рассматриваемого метода.

2.4 Представление ресурсов в детекторе

Стоит упомянуть, что при анализе кода имеет смысл сохранять в контекст и резюме только объекты классов, унаследованных от интерфейса `IDisposable`, так как только они являются интересующими нас неуправляемыми ресурсами.

Рассмотрим освобождение ресурса. Метод `Dispose()` может быть как библиотечным, так и заданным пользователем. В случае библиотечного метода, когда реализация недоступна при проведении анализа, или метода, в резюме которого нет сохраненных освобожденных ресурсов, запоминаем как освобожденные `IDisposable` поля объекта класса, к которому применен метод `Dispose()`, так как применяя метод `Dispose()` к какому-либо объекту, программист предполагает удаление этого объекта, то есть освобождение его полей. В случае заданного пользователем метода с непустым резюме освобожденных ресурсов – в контекст вызывающего метода переносится резюме заданного пользователем метода.

В случае вызова любого метода, отличного от `Dispose()`, используя его резюме, получаем информацию об освобожденных в нем ресурсах, и записываем ее в контекст вызывающего метода.

Также следует учесть, что значение объекта может быть ϕ-функцией, следовательно, его `ValueId` также будет ϕ-функцией. В этом случае проделываем все то же для всех достигающих это использование значений объекта, учитывая условия достижения использования.

2.5 Использование ресурсов и вызов диагностики

Следует рассмотреть различные сценарии использования объектов. В данной работе рассматривается два из них: вызов метода, и разыменование, так как только они могут породить неопределенное поведение. Встречая любой из них при последовательном внутрипроцедурном анализе или при переносе резюме какого-либо вызванного метода, из контекста мы извлекаем список освобожденных ресурсов.

Если ресурс был освобожден, и условия его использования и освобождения совместны, то есть могут выполняться одновременно, тогда создается диагностика. Если диагностика добавлена, выходим из обработки использования. Если же ресурс не был освобожден, или диагностика не добавилась, сохраняем это использование в контекст анализируемого метода вместе с его условием (а также достигающие значения в случае ϕ-функции) для дальнейшего анализа.

Чтобы исправить найденную детектором ошибку, следует понимать, в чем конкретно она заключается, где находятся освобождение и использование ресурса. Таким образом, для упрощения анализа истинности предупреждения используются возможности SharpChecker, позволяющие, зная расположение удаления и использования в коде, отобразить их. Так, в контексте и резюме, помимо условий освобождения и использования ресурса, будем хранить еще и локации.

3. Результаты

Тестирование производилось на наборе из 21 проекта с открытым исходным кодом, имеющих размер 6 млн. строк кода. Анализ выполнялся на машине с 8-ядерным процессором Intel(R) Core(TM) i7-6700, имеющей 32 Gb RAM.

Было выдано 119 предупреждений из которых все были проанализированы вручную. 83 из них были классифицированы как истинные, 36 – как ложные. В результате доля истинных срабатываний составляет 70%.

Рассмотрим некоторые из этих случаев в качестве иллюстрации.

3.1 Пример 1

Ошибка (листинг 2) найдена на проекте Roslyn версии 2.8.2, JsonWriterTests.cs: 92–97.

```
1 var stringWriter = new StringWriter();
2 using (var jsonWriter = new JsonWriter(stringWriter))
3 {
4       ...
5 }
6 // ресурс stringWriter освобожден после блока using
7 return stringWriter.ToString();// USE_AFTER_DISPOSE
    Resource <stringWriter> was already disposed
```

Листинг 2. Место срабатывания
Listing 2. Location of the error

```
1  internal sealed class JsonWriter : IDisposable
2  {
3      private readonly TextWriter _output;
4      ...
5      public JsonWriter(TextWriter output)
6      {
7          _output = output;
8      ...
9      }
10      ...
11      public void Dispose()
12      {
13           _output.Dispose();
14      }
15      ...
16 }
```

Листинг 3. Класс JsonWriter
Listing 3. JsonWriter class

В данном случае `stringWriter` и `jsonWriter` являются `IDisposable` объектами, более того, в конструкторе класса `JsonWriter` (3) не создаются новые объекты, а присваиваются переданные. Таким образом, одно из полей объекта `jsonWriter` (конкретно `_output`) является ссылкой на объект `stringWriter`, и когда в конце конструкции `using` освобождается ресурс `jsonWriter`, освобождается и `stringWriter`. Далее вызывается метод `ToString()` для освобожденного `stringWriter`. Эту проблему достаточно легко не заметить в процессе написания кода, и на этапе выполнения могло бы возникнуть неопределенное поведение.

3.2 Пример 2

Ошибка (листинг 4) найдена на проекте netmq версии 3.3.0.12, XSub.cs: 196–227.

```
1 while (true)
2 {
3     bool isMessageAvailable = m_fairQueueing.Recv(ref msg);
        // USE_AFTER_DISPOSE Resource
      <m_fairQueueing.m_atomicCounter> was already disposed
4     ...
5     while (msg.HasMore)
6     {
7         isMessageAvailable = m_fairQueueing.Recv(ref msg);
           // освобождение ресурса
8         ...
9     }
10 }
```

Листинг 4. Место срабатывания
Listing 4. Location of the error

```
1 public bool Recv(ref Msg msg)
2 {
3     return RecvPipe(null, ref msg);
       // использование ресурса, освобождение ресурса
4 }
5 public bool RecvPipe(Pipe[] pipe, ref Msg msg)
6 {
7      msg.Close();
8      ...
9 }
10
11 public void Close()
12 {
13     if (... || m_atomicCounter.Decrement() == 0)
14     {
15      ...
16     }
17     m_atomicCounter.Dispose();
18     ...
19 }
```

Листинг 5. Реализация методов
Listing 5. Method Implementation

В этом примере использование стоит раньше удаления в коде, однако это все происходит в цикле. Таким образом, когда в конце одной итерации объект удаляется, на следующей итерации он используется и возникает неопределенное поведение. Видно, что за использование и освобождение отвечает один и тот же метод `Recv(ref Msg msg)`, однако если посмотреть на его реализацию и реализации последовательно вызываемых методов

(листинг 5), видно, что сначала идет использование объекта, а потом удаление, и если два раза подряд вызвать `Recv`, то получим использование освобожденного ресурса.

При включении детектора использований освобожденных ресурсов время работы SharpChecker возрастает с 1 часа и 19 минут до 1 часа и 38 минут. Увеличение времени на 23% связано с количеством информации, которую необходимо хранить в резюме методов, копировать при анализе вызовов и обрабатывать для создания диагностик. В связи с замедлением работы планируется провести анализ накопленных данных с целью минимизации их объема.

4. Заключение

В коде больших проектов встречаются ошибки, из-за которых может произойти аварийное завершение работы программы или потеря данных из-за использования освобожденного ресурса. Некоторые из этих ошибок могут привести к непоправимым последствиям. Рассмотрев большинство случаев, где возникает использование освобожденного ресурса, получилось создать алгоритм обнаружения такого рода ошибок, являющийся основой описанного детектора с точностью около 70% истинных срабатываний.

Анализ двадцати одного проекта с открытым исходным кодом показал, что ошибки, связанные с использованием освобожденного ресурса, нередко допускаются, что свидетельствует о прикладной ценности реализованного детектора.

Список литературы / References

[1] Baldoni R., Coppa E. et al. A survey of symbolic execution techniques. ACM Computing Surveys (CSUR), vol. 51, issue 3, 2018, pp. 1-39.

[2] Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40. DOI: 10.15514/ISPRAS-2016-28(1)-2 / Koshelev V.K., Ignatiev V.N. et al. SharpChecker: Static analysis tool for C# programs. Programming and Computer Software, vol. 43, issue 4, 2017, pp. 268−276.

[3] Аветисян А., Белеванцев А. и др. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ. Труды ИСП РАН, том 21, 2011 г., стр. 23-38 / Avetisyan A., Belevantsev A. et al. Using static analysis for finding security vulnerabilities and critical errors in source code. Trudy ISP RAN/Proc. ISP RAS, vol. 21, 2011, pp. 23-38 (in Russian).

[4] Tiobe index for ranking the popularity of programming languages, 2022. URL: https://www.tiobe.com/tiobe-index.

[5] Иванников В.П., Белеванцев А.А. и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 231-250. DOI: 10.15514/ISPRAS-2014-26(1)-7 / Ivannikov V.P., Belevantsev A.A. et al. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, issue 5, 2014, pp. 265-275.

[6] Аветисян А., Бородин А. Механизмы расширения системы статического анализа svace детекторами новых видов уязвимостей и критических ошибок. Труды ИСП РАН, том 21, 2011 г., стр. 39-54 / Avetisyan A., Borodin A. Mechanisms for extending the system of static analysis Svace by new types of detectors of vulnerabilities and critical errors. Trudy ISP RAN/Proc. ISP RAS, vol. 21, 2011, pp. 39-54 (in Russian).

[7] Henry J., Monniaux D., Moy M. Pagai: a path sensitive static analyser. Electronic Notes in Theoretical Computer Science, vol. 289, 2012, pp. 15-25.

[8] Несов В. Автоматическое обнаружение дефектов при помощи межпроцедурного статического анализа исходного кода. Материалы XI Международной конференции РусКрипто, 2009.

[9] Bai J.-J., Lawall J. et al. Effective static analysis of concurrency {use-after-free} bugs in linux device drivers. In Proc. of the USENIX Annual Technical Conference (USENIX ATC 19), 2019, pp. 255-268.

[10] van der Kouwe E., Nigade V., Giuffrida C. DangSan: scalable use-after-free detection. In Proc. of the Twelfth European Conference on Computer Systems, 2017, pp. 405-419.

[11] Ye J., Zhang C., Han X. UAFChecker: scalable static detection of use-after-free vulnerabilities. In Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 1529-1531.

[12] Shimchik N., Ignatyev V., Belevantsev A. Improving accuracy and completeness of source code static taint analysis. In Proc. of the 2021 Ivannikov ISPRAS Open Conference (ISPRAS), 2021, pp. 61-68.

[13] Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40. DOI: 10.15514/ISPRAS-2016-28(1)-2 / Koshelev V.K., Ignatiev V.N. et al. SharpChecker: Static analysis tool for C# programs. Programming and Computer Software, vol. 43, issue 4, 2017, pp. 268−276.

[14] dotnet/roslyn: The Roslyn .NET compiler provides C# and Visual Basic languages with rich code analysis APIs. Available at: https://github.com/dotnet/roslyn, accessed 23.10.2021.

Информация об авторах / Information about authors

Ульяна Владимировна ТЯЖКОРОБ – студентка бакалавриата Физтех-школы Радиотехники и Компьютерных Технологий Московского физико-технического института, сотрудник ИСП РАН. Научные интересы: статический анализ программ, статическое символьное выполнение, алгоритмы поиска дефектов в исходном коде.

Uljana Vladimirovna TSIAZHKOROB is a bachelor's student at the Department of Radio Engineering and Computer Technologies of the Moscow Institute of Physics and Technology, an employee of the ISP RAS. Research interests: static program analysis, static symbolic execution, algorithms for finding defects in source.

Валерий Николаевич ИГНАТЬЕВ, кандидат физико-математических наук, старший научный сотрудник ИСП РАН, доцент кафедры системного программирования факультета ВМК МГУ. Научные интересы включают методы поиска ошибок в исходном коде ПО на основе статического анализа.

Valery Nikolayevich IGNATYEV, PhD in computer sciences, senior researcher at Ivannikov Institute for System Programming RAS and associate professor at system programming division of CMC faculty of Lomonosov Moscow State University. He is interested in techniques of errors and vulnerabilities detection in program source code using static analysis.

Андрей Андреевич БЕЛЕВАНЦЕВ – доктор физико-математических наук, ведущий научный сотрудник ИСП РАН, профессор МГУ. Сфера научных интересов: статический анализ программ, оптимизация программ, параллельное программирование.

Andrey Andreevich BELEVANTSEV – Dr.Sc., Leading Researcher at ISP RAS, Professor at MSU. Research interests: static analysis, program optimization, parallel programming.

DOI: 10.15514/ISPRAS-2022-34(6)-4

Irbis: статический анализатор помеченных данных для поиска уязвимостей в программах на C/C++

[1] *Н.В. Шимчик, ORCID: 0000-0001-9887-8863 <shimnik@ispras.ru>*
[1,2] *В.Н. Игнатьев, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>*
[1,2] *А.А. Белеванцев, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*

Аннотация. Статический анализ помеченных данных может использоваться для обнаружения различных потенциальных уязвимостей в коде программ путём исследования потоков данных между истоками и стоками помеченных данных. Чаще всего помеченными называют данные, которые были получены из внешнего источника и не были должным образом проверены. Инструмент Irbis реализует статический межпроцедурный анализ помеченных данных на основе решения задачи IFDS (Interprocedural Finite Distributive Subset), а также различные расширения, улучшающие его масштабируемость, точность и полноту. В нём реализованы 4 детектора с разными определениями помеченных данных, используемыми для нахождения выхода за границы буфера, использования освобождённой памяти, использования константных паролей и утечек данных. Определения истоков, стоков и передаточных функций хранятся в формате JSON и могут изменяться пользователем. Мы сравнили результаты анализа на проекте Juliet Test Suite for C/C++ с несколькими другими анализаторами, такими как Infer, Clang Static Analyzer и Svace. Irbis смог продемонстрировать 100% покрытие на подмножестве тестов, имеющих отношение к помеченным данным для поддерживаемых нами CWE. При этом реализованные нами эвристики смогли подавить все ложные срабатывания на данном наборе тестов. Также мы демонстрируем масштабируемость и процент ложных срабатываний инструмента при запусках на реальных проектах и показываем примеры существующих уязвимостей, которые могут быть им обнаружены.

Ключевые слова: статический анализ; анализ помеченных данных; поиск уязвимостей

Для цитирования: Шимчик Н. В., Игнатьев В. Н., Белеванцев А. А. Irbis: статический анализатор помеченных данных для поиска уязвимостей в программах на C/C++. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 51-66. DOI: 10.15514/ISPRAS-2022-34(6)-4

Благодарности. Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект №20-01-00581 А.

Irbis: static taint analyzer for vulnerabilities detection in C/C++

[1] *N.V. Shimchik, ORCID: 0000-0001-9887-8863 <shimnik@ispras.ru>*
[1,2] *V.N. Ignatiev, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>*
[1,2] *A.A. Belevantsev, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the RAS,*
25 Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2]*Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. Static taint analysis can be used to find various security weaknesses and vulnerabilities in programs by discovering dataflow paths from taint sources to taint sinks. In most cases the data is called ”tainted” if it was obtained from an untrusted source without proper sanitization. In this paper we present a static taint analyzer Irbis. It implements analysis based on IFDS (Interprocedural Finite Distributive Subset) dataflow problem, as well as various extensions aimed at improving accuracy and completeness of the analysis. It supports different definitions of tainted data, which enables it to find such weaknesses as out of buffer access, use of freed memory, hardcoded passwords, data leaks and discover dataflow paths between user-defined sources and sinks. All sources, sinks and propagators definitions are stored in JSON format and can be adjusted to meet the users’ needs. We compare analysis results on Juliet Test Suite for C/C++ with several other analyzers, such as Infer, Clang Static Analyzer and Svace. Irbis manages to demonstrate 100% coverage on taint-related subset of tests for implemented CWEs, while suppressing all the false positives using heuristics. We also show performance and false positive rate on real projects, with examples of real vulnerabilities, which can be detected by Irbis.

Keywords: static analysis; taint analysis; vulnerabilities detection

For citation: Shimchik N. V., Ignatyev V. N., Belevantsev A. A. Irbis: static taint analyzer for vulnerabilities detection in C/C++. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 51-66 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-4

Acknowledgments. This work was supported by the Russian Foundation for Basic Research, project №20-01-00581 A

1. Введение

В работе рассматривается задача поиска потенциальных уязвимостей и ошибок безопасности в коде программ на языках C и C++. Наличие уязвимостей является серьёзной проблемой для разработчиков программных продуктов, т.к. они могут быть использованы для несанкционированного доступа к данным или вмешательства в работу ПО. Ошибки могут годами не проявляться при обычном использовании программы, однако в случае обнаружения злоумышленниками создают угрозу сервисам, использующим этот код. Наиболее подвержены опасности программы, принимающие данные по сети и из других потенциально компрометируемых источников. Наличие ошибки в библиотеке делает уязвимыми все программы, ее использующие. Одним из известных примеров такого инцидента можно назвать CVE-2014-0160 [1,2], также известную как Heartbleed. Эта уязвимость в популярной криптографической библиотеке OpenSSL заключается в отсутствии проверки размера буфера, считанного из внешнего источника, и открывала злоумышленникам возможность специальным запросом получать содержимое некоторых областей памяти процесса.

Существует возможность находить такие уязвимости ещё на этапе разработки. Один из способов состоит в использовании статического анализа помеченных данных. Эта разновидность анализа сводится к обнаружению путей выполнения программы, ведущих от инструкций возникновения помеченных данных к инструкциям или функциям, в которых их использование нежелательно. В зависимости от того, что именно считается помеченными данными, такое определение подходит для описания различных классов ошибок и

потенциальных уязвимостей: от выхода за границы буфера, до использования константных паролей и утечки чувствительных данных. Подробнее об этом в разд. 2 и 3.

Для поиска уязвимостей в ПО существует множество других подходов и инструментов: динамический анализ [3-5], фаззинг [6-8], подходы, сочетающие динамический анализ и символьное выполнение [9-11]. Рассмотрение и анализ этих подходов находится за рамками данной работы, т.к. перечисленные подходы требуют непосредственного запуска программы.

Широко применяются для поиска уязвимостей и подходы на основе статического анализа, т.е. не требующие выполнения программы, включающие анализ абстрактного синтаксического дерева, анализ потоков данных [12], различные методы на основе символьного выполнения [13, 14], формальная верификация [15]. Среди распространенных статических анализаторов, реализующих такой тип анализа, можно упомянуть инструменты Svace [16], Clang Static Analyzer [17], Infer [18]. Все они ориентированы на поиск максимального числа истинных предупреждений за ограниченное время, поэтому оптимизированы таким образом, чтобы минимизировать вероятность выдачи ложного предупреждения.

Однако уязвимости часто скрываются в программах и проявляются только на редких путях выполнения в «необычных» ситуациях, которые, зачастую, эвристически отфильтровываются у статических анализаторов общего назначения. Irbis создается в результате необходимости альтернативного подхода, реализующего более консервативный анализ, при котором баланс между долей истинных и ложных предупреждений смещен с целью исключить возможность пропуска ошибки, но тем не менее допускающий это (в отличие от формальной верификации).

2. Статический анализ помеченных данных

Задача статического анализа помеченных данных состоит в построении пути выполнения между созданием и предварительно заданным способом использования помеченных данных без запуска программы. Возникновение помеченных данных происходит в результате выполнения функций или инструкций – *истоков (source)*. Сам анализ состоит в построении путей распространения помеченных данных в программе, приводящих к передаче помеченных данных в предварительно заданное множество функций или инструкций – *стоков (sink)*. В процессе распространения помеченные данные могут подвергаться манипуляциям, например, модифицироваться, передаваться в качестве аргументов в функции и т.д. Для того, чтобы определить, сохраняет ли указанная операция помеченность, вводятся *передаточные функции (propagators)* – правила распространения пометок для всех операций и *санитайзеры (sanitizers)* – функции или операции, снимающие помеченность с данных. Статический анализ помеченных данных может выполняться как на исходном коде, так и на бинарном представлении программы и анализирует все пути выполнения, включая недостижимые.

Таким образом, для задания отдельного детектора ошибки в инфраструктуре анализатора помеченных данных необходимо задать множества истоков, стоков, санитайзеров и направление анализа. Направление может быть как прямым, так и обратным и служит для оптимизации. Например, в детекторе константных паролей гораздо эффективнее проводить анализ в обратном направлении – от использования переменной в качестве пароля назад к записи в нее константной строки. В противном случае было бы необходимо помечать все строки в программе, лишь малая часть которых может впоследствии использоваться в качестве пароля. Передаточные функции, как правило, не зависят от конкретного детектора и реализованы единым образом для всех них.

Мы выделяем два подхода к реализации статического анализа помеченных данных: на основе статического символьного выполнения и на основе анализа потоков данных [19]. Первый подход предполагает распространение пометок по символьным значениям и способен

использовать информацию о различных зависимостях между переменными, а также учитывать условия на путях в программе, по которым проходят помеченные данные. Подход подразумевает однократный обход всех функций в обратном топологическом порядке по графу вызовов, чтобы собрать и сохранить в резюме информацию обо всех вызванных функциях. Таким образом подход обеспечивает масштабируемый межпроцедурный, чувствительный к путям, анализ, обеспечивающий высокую долю истинных срабатываний за счет использования SMT решателей для проверки достижимости путей и предусловий ошибки. Однако определенный порядок анализа функций приводит к невозможности обнаружения ошибок в случае, когда сток находится внутри нескольких обёрточных функций, т.к. неизвестно, могут ли на вход подаваться помеченные значения. Среди реализующих такой подход инструментов можно отметить Svace.

Второй подход основан на предложенном в работе [20] обобщении IFDS/IDE решателя, сводящего задачу к проблеме достижимости на графе. Множество задач анализа потоков данных, например, поиск достигающих определений или живых переменных, может быть переформулирован в виде межпроцедурной задачи анализа потоков данных с дистрибутивными передаточными функциями над конечными доменами фактов IFDS/IDE [21]. Эта задача относится к межпроцедурному анализу потоков данных, обладает чувствительностью к потоку и контексту вызова функции, но не обладает чувствительностью к путям: другими словами, предполагается, что распространение помеченности T в точке программы V не учитывает то, какой путь распространения предшествовал этой точке. Отсутствие чувствительности к путям предполагает большее количество ложных срабатываний по сравнению со статическим символьным выполнением, но зато такой подход позволяет покрыть почти все имеющиеся потоки данных в программе.

Во время анализа строится расширенный суперграф, каждая вершина которого задается кортежем ⟨Вершина ГПУ, Помеченный факт, Контекст вызова функции⟩. Построение вершин графа начинается с истока и каждая вершина посещается не более одного раза. Поскольку состояние вершины не зависит от того, каким именно путём анализ пришел в конкретную точку программы, и от значений прочих переменных, алгоритм не подвержен взрывному росту количества исследуемых путей в программе. Такой подход реализован в Irbis, SharpChecker [22], FlowDroid [12].

Одним из развивающихся в настоящее время инструментов является PhASAR [14] – инфраструктура статического анализа на основе LLVM [23], которая позволяет пользователю задавать и решать некоторые задачи анализа потоков данных, включая IFDS/IDE, применимый для реализации анализа помеченных данных. Однако готовых детекторов ошибок не реализовано, а приводится только образец, демонстрирующий тривиальный сценарий и не имеющий практической значимости. Инструмент имеет также и другие существенные ограничения: задача объявления и построения множеств истоков, стоков, санитайзеров, а также моделирования библиотечных функций в данном инструменте не решается, интеграция со сборкой проекта отсутствует. Однако наиболее существенным ограничением является использование LLVM-Value для идентификации помеченных данных, что не позволяет моделировать, например, отдельные элементы массива. Таким образом, PhASAR может стать перспективным инструментом после устранения важных для поиска уязвимостей ограничений.

Irbis реализует статический анализ помеченных данных на основе решения задачи IFDS (Interprocedural Finite Distributive Subset problem), предложенной в статье [24] и применённой для поиска уязвимостей в программах на языке Java в инструменте Flowdroid [12]. Выбор подхода обусловлен желанием добиться максимально возможного покрытия путей выполнения. Это достигается за счет большего количества ложных срабатываний. Однако основная проблема статического анализа помеченных данных состоит не в реализации подхода, а в разработке набора алгоритмов и эвристик, обеспечивающих приемлемое время работы и долю ложных срабатываний. Кроме того, для анализа языков C и C++

дополнительно необходима реализация анализа псевдонимов. Детали реализации статического анализа помеченных данных в нашем инструменте подробно изложены в статьях [25, 26]. Для исключения срабатываний, пропущенных другими анализаторами, реализован алгоритм анализа виртуальных вызовов и вызовов по указателю, позволяющий не терять помеченность при таких вызовах, эвристический алгоритм обнаружения истоков, позволяющий исследовать, например, библиотеки, для которых истоки и стоки находится в разных компонентах связности графа вызовов, алгоритмы моделирования окружения — функций без исходного кода.

3. Типы детекторов и обнаруживаемых уязвимостей

В традиционных инструментах анализа помеченных данных, как было отмечено ранее, детекторы задаются кортежем из четырех компонентов:

⟨мн-во истоков, мн-во стоков, мн-во санитайзеров, направление анализа⟩.

В Irbis отсутствуют функции-санитайзеры, однако применяется ряд эвристик, снимающих помеченность, описанных в разделе 4.3. Поскольку большинство функций не снимают помеченность с некоторых поданных на вход данных, а возвращают очищенные от помеченности данные в виде результата, то такие санитайзеры моделируются исключением функции-санитайзера из множества передаточных функций, останавливая таким образом распространение помеченности. Основные детекторы, поддерживаемые Irbis, представлены в табл. 1. Они сгруппированы по нескольким принципам, образуя иерархическую классификацию. Первичное деление выполнено по типу истоков, в результате чего выделено 4 группы истоков.

1) *Ненадежные входные данные* формируют истоки для многих детекторов критических уязвимостей. Они включают данные, которые могут быть модифицированы пользователем. Попадание таких данных без проверки в критическую инструкцию, например, индекс массива, или функцию, например, exec является серьезной уязвимостью. Ненадежные данные могут быть как результатом выполнения функции, так и формальными параметрами функции, например, argc, argv в функции main. Irbis предоставляет возможность выбирать интересующие аналитика группы истоков. Данные истоки могут приводить к уязвимостям, связанным с недостаточной проверкой входных данных (CWE-20).
2) *Чувствительные данные*, такие как, например, криптографические ключи, пароли, рассматриваемые в качестве истока помеченных данных, формируют класс уязвимостей типа «утечка данных». Запись чувствительных данных в файлы, вывод на экран или отправка по сети без шифрования могут скомпрометировать всю программу.
3) *Криптографические функции* являются истоком для анализа в обратном направлении в детекторах использования константных ключей шифрования, паролей.
4) *Указатель на участок освобожденной памяти* `p`, полученный, например, после выполнения `free(p);` или `delete p;` выступает в качестве истока для детекторов повторного освобождения или использования освобожденной памяти.

Для категории 1 предлагается более детальная классификация обнаруживаемых классов ошибок в зависимости от стоков. Одним из наиболее важных типов проблем безопасности являются ошибки в работе с памятью:

- чтение или запись за пределами ожидаемой области памяти (CWE-121, 122, 124, 126, 127),
- ошибки, связанные с форматной строкой (CWE-134),
- выделение слишком большого участка памяти (CWE-789),
- целочисленное переполнение при выделении памяти (CWE-680).

Табл. 1. Список основных детекторов инструмента Irbis
Tab. 1. List of main detectors of the Irbis tool

Детектор	**Исток**	**Сток**
TAINTED_ARG	ненадежные данные	список критических функций
TAINTED_PTR.LOAD	ненадежные данные	инструкция чтения памяти по помеченному указателю
TAINTED_PTR.STORE	ненадежные данные	инструкция записи в память по помеченному указателю
TAINTED_LOOP_CONDITION	ненадежные данные	условие цикла
SETTING_MANIPULATION	ненадежные данные	сохранение в качестве настройки, например, `sethostname`
PROCESS_CONTROL	ненадежные данные	запуск процесса, например, `exec`
SQL_INJECTION	ненадежные данные	использование в SQL запросе
RESOURCE_INJECTION	ненадежные данные	использование в качестве идентификатора ресурсов, например, пути файловой системы
GETLOGIN	`getlogin` (ненадежные данные)	сравнение с константой
GETHOSTBY	`gethostbyaddr` (ненадежные данные)	сравнение с константой
DATA_LEAK	чувствительные данные	печать, отправка по сети, сохранение в файл и т.д.
HARDCODED_PASSWORD	криптографические функции	константная строка
HARDCODED_PASSWORD_EXTRA	регулярное выражение имени переменной	константная строка
HARDCODED_SALT	криптографические функции	константная строка
HARDCODED_KEY	криптографические функции	константная строка
USE_AFTER_FREE	освобожденная память (`free()`, `delete`)	разыменование указателя на освобожденную память
PASSED_TO_PROC_AFTER_FREE	освобожденная память (`free()`, `delete`)	использование в качестве фактического аргумента в вызове `free()`, `delete`
DOUBLE_FREE	освобожденная память (`free()`, `delete`)	

Другой группой проблем является отказ в обслуживании, например, при использовании ненадежных данных в условии цикла для контроля количества итераций (CWE-400), а также попадание помеченных данных в другие чувствительные конструкции управления.

В третью группу выделяется использование ненадежных данных в критических функциях, что может приводить, например, к запуску команд с повышенными привилегиями (Command

injection: CWE-78), внедрению данных в БД (SQL Injection: CWE-89, LDAP Injection: CWE-90).

Отдельно рассматриваются детекторы `GETLOGIN` и `GETHOSTBY`, поскольку они обнаруживают ситуацию, когда поведение программы определяется результатом, возвращаемым функциями `getlogin` и `gethostaddr`.

Также следует отметить, что для детектора `DATA_LEAK` набор истоков и стоков задаётся самим пользователем, поскольку информация о том, какие данные являются секретными, а также выбор конкретных способов утечки, зависит от конкретного проекта.

3.1 Теги типов детекторов

Для разбиения множества предупреждений внутри одного детектора используются теги. Они присоединяются к типу предупреждения в заданном порядке, образуя подгруппы, обладающие определенными общими свойствами. Это позволяет пользователю просматривать более «интересные» предупреждения в первую очередь, за счёт выделения с помощью тегов группы с наибольшим числом истинных срабатываний, характерных для анализируемого проекта или наоборот – использования тегов, обозначающих «плохие» предупреждения. Одно предупреждение может иметь одновременно несколько тегов.

- `MACRO` означает, что сток обнаруженной ошибки происходит из макроса. Если одно предупреждение с таким тегом является ложным, то и остальные такие срабатывания с большой вероятностью окажутся ложными, так макрос предполагает одинаковую реализацию стоков.
- `OVERTAINT` помечает предупреждения, для которых в процессе распространения пометок консервативно были помечены соседние поля структуры или класса. Например, если в структуре был помечен *i*-ый элемент массива, топри отрицательных или достаточно больших значениях *i* могли быть помечены соседние с массивом поля структуры.
- `OVERFLOW`, как правило, свидетельствует о высокой вероятности истинности предупреждения, т.к. помечает предупреждения, где возможно целочисленное переполнение индекса при обращении к буферу или размера выделенной памяти.
- `HEURISTIC_SOURCE` используется для срабатываний, исток которых был угадан Irbis на основе эвристик, что бывает полезно для библиотек, когда нельзя доказать, что функция, например, читает пользовательские данные.
- `UNLIKELY` используется для известных шаблонов кода, характерных для ложных срабатываний, которые в очень редких случаях могут быть реальной уязвимостью.
- `INCONSISTENT` помечает ложные срабатывания, которые связаны с девиртуализацией и разрешением косвенных вызовов, но из-за ограничений статического анализа могут оказаться истинными. Такой тег получают пути распространения пометок, на которых для одного объекта были последовательные вызовы виртуальных функций из различных классов-наследников или вызовы по указателю функций из разных групп инициализации.
- `VARARG` сигнализирует о том, что помеченные данные были переданы через функцию с переменным числом аргументов и возможны неточности, как правило избыточная помеченность.
- `MALLOC` помечает предупреждения, в которых сток соответствует `malloc` или `new`, поскольку для многих программ это может не считаться уязвимостью.
- `BUFFER_LENGTH` помечает группу потенциально истинных предупреждений, в которых у Irbis оказалось достаточно информации о размерах задействованных буферов, чтобы судить о возможности переполнения.

4. Схема работы инструмента

В этом разделе рассматриваются три вопроса: место инструмента в инфраструктуре инструмента Svace, общая схема самого анализатора и улучшения, которые были внесены для того, чтобы инструмент имел практическую применимость.

4.1 Инфраструктура

В качестве основы для представления программы наш инструмент использует не её исходный текст, а внутреннее представление LLVM, создаваемое компилятором Clang. Получить его для целого проекта можно различными способами, одним из самых доступных из которых является использование проекта WLLVM [27], который указывается вместо обычного компилятора при сборке проекта и может создавать файлы бит-кода, соответствующие его исполняемым файлам или библиотекам.

Наш анализатор во многом полагается на инфраструктуру анализатора Svace, который также использует внутреннее представление LLVM. Хотя мы не задействуем его подсистемы анализа, мы используем реализованную в нём систему перехвата сборки для получения файлов бит-кода отдельных модулей компиляции и информации о вызовах компоновщика. Затем мы запускаем собственный скрипт, который использует *llvm-link* для объединения отдельных файлов в соответствии с полученной информацией о ходе сборки – именно результат работы этого скрипта и подаётся на вход анализатору.

После окончания анализа, его результаты переводятся в формат *svres*, что позволяет экспортировать их обратно в инфраструктуру Svace и отобразить в удобном для пользователя виде: в браузере, с переходами по точкам трассы, поиском, возможностью разметки предупреждений и другими возможностями.

В результате вся схема анализа требует минимального участия пользователя: от него требуется только последовательно вызвать набор команд, одной из которых указать команду для сборки проекта (в большинстве случаев это `make`), дождаться окончания анализа, после чего открыть браузер для просмотра сгенерированных срабатываний. Существует также возможность ручного задания набора истоков, стоков и передаточных функций, а также выбора нестандартных опций запуска, но этот этап является необязательным.

4.2 Схема анализа

Анализатор получает на вход программу во внутреннем представлении LLVM и выполняет несколько подготовительных этапов, которые могут различаться в зависимости от выбранных опций;

1) построение графа потока управления (ГПУ) для всех функций в программе;
2) поиск кандидатов для всех вызовов виртуальных методов, использующий метаданные LLVM;
3) поиск кандидатов для вызовов функций по указателю, использующий решение задачи IFDS, в которой истоками являются операции взятия адреса функции, а стоками — вызовы по указателю;
4) построение графа вызовов;
5) определение ограничений на допустимые значения переменных, следующихиз условий в коде программе.

Затем для каждого включённого детектора решается соответствующая ему задача IFDS, а найденные пути от истоков до стоков экспортируются в виде трассы распространения помеченности.

1) Осуществляется поиск всех истоков в программе, соответствующих данному детектору. Запоминаются соответствующие истокам вершины ГПУ и пути доступа, по которым записываются помеченные данные.

2) Для каждой пары ⟨Вершина ГПУ, Помеченный факт⟩ запускается отдельный IFDS-решатель, который распространяет помеченный факт в соответствии с заданными передаточными функциями и стоками (в зависимости от детектора, факты могут распространяться не только в прямом, но и в обратном направлении). Для всех достигнутых стоков запоминается вершина расширенного суперграфа, представляющая собой тройку ⟨Вершина ГПУ, Текущий помеченный факт, Текущий контекст функции⟩ .
3) После окончания анализа очередного истока берётся соответствующий ему расширенный суперграф распространения помеченности и из него выделяется самый короткий путь от истока до каждого стока – эти пути экспортируются либо в текстовый формат, либо в формат, подходящий для экспорта в Svace.

В текущей версии инструмента появилась возможность выполнять некоторые этапы анализа параллельно. При выборе соответствующей опции, каждый исток может анализироваться в отдельном процессе, полностью независимо от других – при этом количество одновременно запущенных процессов определяется автоматически или указывается пользователем. Это позволяет задействовать больше ресурсов компьютера, но для проектов с небольшими графами распространения помеченности и большим количеством истоков стоимость накладных расходов на создание процессов и передачу данных между ними может превысить выгоду от параллельного анализа и приводить к замедлению вместо ускорения. В частности, такая ситуация может быть характерна для анализа тестовых наборов.

4.3 Особенности анализа

Наивная реализация алгоритма IFDS испытывала проблемы как с масштабируемостью, так и с полнотой и точностью анализа. Ключевыми для решения этих проблем стали следующие пункты:

1) Irbis использует отдельный IFDS-решатель для каждого истока; это приводит к замедлению, за счёт повторного анализа одних участков программы, но ограничивает максимальный размер расширенного суперграфа в каждый конкретный момент времени, решая проблему нехватки памяти при анализе больших проектов;
2) для понимания того, как ведёт себя помеченность при вызове библиотечных функций, необходимы аннотации, создаваемые вручную. В ходе анализа мы отслеживаем, какие функции чаще стирают помеченность, а значит нуждаются в аннотациях в первую очередь;
3) мы также создали различные эвристики, которые убирают часть «плохих» срабатываний, либо помечают их тегами, которые позволяют пользователю смотреть их в последнюю очередь; имеются и эвристики, которые помогают находить неизвестные истоки в программе, используя названия функций и типы их аргументов.

4.3.1 Аннотации

Аннотации – это краткие описания того, что делает данная функция с точки зрения распространения помеченности. Количество и качество аннотаций напрямую влияет на качество анализа, поскольку при проведении статического анализа кода программы доступны только реализации функций, реализованных в самой программе.

Аннотации могут использоваться не только для описания поведения библиотечных функций, но и для обобщения поведения известных функций, что позволяет ускорить анализ, избавляя от необходимости «честного» распространения помеченности внутри такой функции.

В нашем инструменте аннотации задаются в формате JSON и могут свободно дополняться самим пользователем, если имеющихся по умолчанию оказывается недостаточно. Аннотации делятся на три вида: описания истоков, передаточных функций и стоков. Они позволяют задавать:

- название или сигнатуру функции или целого набора схожих функций при помощи регулярных выражений;
- порядковые номера аргументов, через которые проходит помеченность: входные и выходные аргументы, в зависимости от того, идёт речь о стоках, истоках или передаточных функциях;
- количество разыменований указателя относительно базового аргумента, байтовые смещения в структурах и именованные поля;
- прочие специальные поля, позволяющие настраивать поведение указателя; например, поле `HIDE_IMPLEMENTATION` позволяет полностью игнорировать реализацию функции и использовать только её аннотацию.

4.3.2 Константная эвристика

По умолчанию анализатор предполагает, что если помеченные данные передаются по ссылке или указателю в библиотечную функцию, о поведении которой ничего не известно, то помеченность с этих данных может быть стёрта. Это поведение можно вручную переопределить при помощи создания аннотации, в которой явно указано, что значение данного аргумента функции не изменяется.

Поскольку создавать такие аннотации для всех возможных библиотечных функций не представляется возможным, была добавлена константная эвристика, которая использует ключевое слово `const`, имеющееся в исходном коде программы (но отсутствующее в биткоде LLVM), означающее, что данное значение предназначено только для чтения. Эта эвристика состоит из двух частей:

1) патч в компилятор Clang, который сохраняет в метаданные информацию об использованиях ключевого слова const в параметрах библиотечных функций;
2) отдельный метод в анализаторе, считывающий эти метаданные и сообщающий, если помеченный факт в данном вызове не будет переписан.

4.3.3 Виртуальные и косвенные вызовы

В языках C и C++ существуют вызовы, в которых на этапе компиляции неизвестно, какая именно функция будет вызвана. Более того, одна и та же инструкция в программе может в разные моменты выполнения приводить к вызову различных функций.

Такие вызовы бывают двух типов:

1) виртуальные – вызов метода, помеченного ключевым словом virtual; в этом случае может быть вызван как сам метод, так и метод одного из наследников класса, в зависимости от объекта, метод которого вызывается;
2) косвенные – вызов по хранимому в памяти указателю на функцию; в этом случае вызываемая функция полностью определяется тем, какой указатель был записан в данную ячейку памяти.

Консервативный подход к статическому анализу предполагает, что виртуальные и косвенные вызовы трактуются так же, как и вызовы неизвестной функции, однако на практике это сильно ограничивает полноту анализа.

Альтернативой является поиск всех возможных функций-кандидатов для таких вызовов, с предположением, что вызываться может любой из найденных кандидатов. Это может приводить к появлению новых ложных срабатываний, поскольку IFDS анализ нечувствителен к путям и не может проверить, что такая комбинация вызываемых кандидатов действительно возможна, но повышает полноту анализа в тех проектах, где часто используются виртуальные или косвенные вызовы.

Для виртуальных вызовов мы восстанавливаем иерархию классов при помощи метаданных, имеющихся в биткоде LLVM, после чего добавляем в список кандидатов все реализации вызываемого метода из классов-наследников.

Для косвенных вызовов мы осуществляем отдельный IFDS анализ, в котором истоками являются взятия адресов функций в программе, а стоками – их использование в вызовах по указателю. Все адреса, достигшие места косвенного вызова, добавляются в качестве его кандидатов.

4.3.4 Снятие помеченности с целочисленных переменных

Хотя Irbis не позволяет задавать санитайзеры в явном виде, в нём была реализована эвристика, позволяющая снимать помеченность с целочисленных переменных. Если включена опция `--integer-sanitization`, то такие переменные перестают считаться помеченными, если в данной точке программы их значение ограничено и сверху, и снизу путём выполнения соответствующих проверок.

Чтобы определить наличие таких ограничений, на предварительном этапе выделяются все сравнения целочисленных переменных с константами. В зависимости от типа сравнения (<, >, ==) и того, является ли тип переменной знаковым или беззнаковым (беззнаковые ограничены снизу нулём), мы указываем, что в базовых блоках, соответствующих началу истинной и ложной веток условия, её значения ограничены сверху и/или снизу. Затем эта информация распространяется на те следующие за ними блоки, которые доминируются одновременно самим условием и началом выбранной ветки.

Во время анализа, при попадании помеченного факта в новый базовый блок, сначала проверяется, соответствует ли он одной из переменных, на которые в данном блоке существуют ограничения. Если это так, и ограничения есть одновременно и сверху и снизу, то помеченный факт стирается.

Данная эвристика работает только с условиями, расположенными недалеко друг от друга и только внутрипроцедурно, однако её оказывается достаточно для многих практических случаев, когда проверяется значение индекса массива или размера выделяемого буфера. В частности, именно она позволила избавиться от всех ложных срабатываний на тестовом наборе Juliet Test Suite [28].

5. Результаты

Для демонстрации результатов анализа использовались несколько проектов.

Для тестирования было выбрано четыре анализатора, реализующих статический анализ помеченных данных: Irbis, Svace, Clang Static Analyzer (CSA) и Infer Static Analyzer. Поскольку большинство из перечисленных инструментов являются анализаторами общего назначения и ищут различные ошибки, включая синтаксические, для большинства анализаторов учитывалась только часть срабатываний:

- для Irbis использовались все имеющиеся детекторы.
- Для Svace сравнение проводилось по всем срабатываниям, содержащим в своём названии слово *TAINT*: *TAINTED_INT*, *TAINTED_INT.LOOP*, *TAINTED_ARRAY_INDEX*, *TAINTED_PTR.FORMAT_STRING* и другим.
- Для Clang Static Analyzer сравнение проводилось по результатам типа *Use of Untrusted Data* и *Out-of-bound access* со включёнными детекторами *alpha.security.taint.TaintPropagation* и *alpha.security.ArrayBoundV2*.
- Для инструмента Infer Static Analyzer сравнение проводилось по результатам работы детекторов *InferBO* и *Quandary*.

Первым анализируемым проектом является подмножество тестового набора Juliet 1.3 test suite for C/C++ [28], включающее в себя тесты на потенциальные уязвимости из следующих классов:

- CWE-124 — Buffer Underwrite;
- CWE-126 — Buffer Overread;
- CWE-127 — Buffer Underread;
- CWE-134 — Uncontrolled Format String;
- CWE-194 — Unexpected Sign Extension;
- CWE-195 — Signed to Unsigned Conversion Error;
- CWE-400 — Resource Exhaustion;
- CWE-680 — Integer Overflow to Buffer Overflow;
- CWE-789 — Uncontrolled Mem Alloc.

Из подмножества были исключены тесты, содержащие в названии слова w32 или wchar, поскольку они предназначены для сборки только в ОС Windows.

Среди оставшихся были отобраны 4656 тестов, в заголовках которых поле `BadSource` содержит слово `read`. В остальных тестах ошибка проявляется на каждом запуске, а не на специальным образом сформированных пользовательских данных, поскольку определяется некорректными константами в коде – обнаружение таких ошибок не имеет отношения к анализу помеченных данных, потому такие тесты в данной работе не рассматриваются.

Большинство тестов включает в себя один содержащий уязвимость тестовый пример и несколько тестовых примеров, на которых уязвимость отсутствует или не осуществима. Чтобы облегчить классификацию срабатываний на ложные и истинные, тестовый набор поддерживает два макроса: `OMIT_GOOD` и `OMIT_BAD`, которые, будучи объявлены во время сборки, убирают тестовые примеры соответствующей группы.

В этом случае, после базового просмотра полученных предупреждений, все срабатывания на проекте, собранном с макросом `OMIT_GOOD`, можно считать истинными, а все срабатывания на проекте, собранном с макросом `OMIT_BAD`, можно считать ложными.

Результаты запусков на Juliet Test Suite приведены в табл. 2.

Табл. 2. Результаты анализа выбранного подмножества Juliet Test Suite
Table. 2. Results of the analysis of the selected subset of Juliet Test Suite

Анализатор	**TP**	**FN**	**FP**	**RAM, Гб**
Irbis	4656	0	0	3
Svace	3666	990	248	4
CSA	1197	3459	17	<1
Infer	753	3903	437	<1

Все инструменты продемонстрировали сравнимое время анализа (меньше 1 часа в сумме по двум запускам), однако поскольку на этом проекте время сборки сопоставимо или превышает время анализа, а для инструментов CSA и Infer их нельзя разделить – сравнивать этот показатель напрямую нецелесообразно. Для анализа проекта инструментом Infer пришлось отредактировать имеющиеся сборочные файлы, так как перехват сборки этого анализатора не поддерживает абсолютные пути к компилятору.

Вторым тестовым проектом является библиотека OpenSSL версии 1.0.1f, содержащая уязвимость CVE-2014-0346 (Heartbleed) [2], которую должен обнаруживать инструмент. Учитываются те же типы срабатываний, что и для предыдущего проекта.

Результаты анализа приведены в табл. 3, с более подробным рассмотрением типов срабатываний инструмента Irbis в табл. 4. Срабатывания Irbis с «ослабляющими» тегами игнорировались.

Табл. 3. Результаты анализа OpenSSL 1.0.1f
Table 3. Analysis' results of OpenSSL 1.0.1f

Анализатор	Срабатываний	TP-rate	Время (мин.)	RAM (Гб)
Irbis	3996 (279)	12% (41%)	87	3
Svace	14	71%	13	6
CSA	22	14%	8	<1
Infer	24 (469)	15%	17	<1

Будем трактовать срабатывание как истинное в том случае, если по нему пользователь может предположить, что помеченные данные действительно достигают указанного стока – эксплуатируемость найденной ошибки безопасности не проверялась. Для каждого типа предупреждений мы вручную разметили не менее 20 случайным образом выбранных срабатываний.

Табл. 4. Оценка процента истинных срабатываний Irbis на OpenSSL (по 20 размеченных срабатываний на каждый тип предупреждения)
Table 4: Estimates of the percentage of true Irbis hits on OpenSSL (20 labeled hits per alert type)

Тип предупреждения	Всего	TP-rate
TAINTED_ARG	172	45%
TAINTED_ARG.BUFFER_LENGTHS	1	100%
TAINTED_ARG.MALLOC	5	40%
TAINTED_LOOP_CONDITION	64	40%
TAINTED_PTR.LOAD	895	25
TAINTED_PTR.STORE	262	50%
RESOURCE_INJECTION	6	100%
PROCESS_CONTROL	2	100%
HARDCODED_PASSWORD_EXTRA	1	0%
USE_AFTER_FREE	1182	0%
PASSED_TO_PROC_AFTER_FREE	1378	0%
DOUBLE_FREE	28	0%

Как можно заметить по этим таблицам, хотя инструмент и производит значительно большее количество срабатываний, однако 93% из них приходится на всего 4 типа предупреждений: в основном на детекторы использования освобождённой памяти и на детектор чтения и записи по помеченному указателю, которые пользователь может проигнорировать. Более того, все просмотренные ложные срабатывания, связанные с освобождённой памятью, являлись следствием того, что отсутствие чувствительности к путям мешает анализатору проверить корректность использования функции *CRYPTO_realloc_clean* – в теории, все эти срабатывания можно пометить новым тегом при помощи добавления ещё одной эвристики, либо убрать добавлением аннотации для этой функции. Большинство предупреждений о чтении и записи по помеченному указателю произошли в криптографических функциях, потому их проверка и классификация на истинные или ложные затруднена.

Для удобства в скобках мы указали статистику без этих 4 типов предупреждений, внёсших наибольший вклад в количество срабатываний. Для инструмента Infer в скобках приведена статистика по срабатываниям уровня L1, обладающим наибольшей достоверностью.

Из четырёх представленных инструментов только Irbis продемонстрировал срабатывание, соответствующее искомой уязвимости. Это предупреждение имеет тип *TAINTED_ARG*, его истоком является чтение данных функцией *BIO_read в s3_pkt.c:239*, а стоком — вызов memcpy с размером, задаваемым помеченной переменной `payload` в *d1_both.c:1487*.

Также, ранее мы демонстрировали возможность обнаружения уязвимости CVE-2018-15209 в проекте LibTIFF версии 4.0.9 в работе [25].

6. Заключение

Мы представили свой статический анализатор помеченных данных для программ на языках C/C++ Irbis, работающий в инфраструктуре Svace. Он реализует 4 основных детектора, решающих задачу IFDS для разных типов помеченных данных, что позволяет ему искать такие потенциальные уязвимости и ошибки безопасности, как выход за границы буфера, обращение к освобождённой памяти, использование константных паролей, утечки чувствительных данных и другие.

Из-за отсутствия чувствительности к путям и других ограничений выбранного метода анализа, этот анализатор обладает более высоким процентом ложных срабатываний на реальных проектах, чем инструменты на основе символьного выполнения, однако он способен распространять помеченность даже по длинным межпроцедурным путям в программах, что делает возможным обнаружение реальных уязвимостей – таких как Heartbleed.

Возможности анализатора были продемонстрированы как на тестовом наборе Juliet Test Suite, так и на реальных проектах, содержащих уязвимости – он показал высокую степень покрытия, при этом время выполнения оказалось сопоставимым с индустриальными инструментами.

Список литературы / References

[1] CVE - CVE-2014-0160. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160, accessed 01.11.2022.

[2] Heartbleed bug. Available at: https://nvd.nist.gov/vuln/detail/CVE-2014-0160, accessed 01.11.2022.

[3] Nethercote N., Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation. SIGPLAN Notices, vol. 42, issue 6. 2007, pp. 89-100.

[4] Haldar V., Chandra D., Franz M. Dynamic taint propagation for java. In Proc. of the 21st Annual Computer Security Applications Conference (ACSAC'05), 2005, pp. 303-311.

[5] Newsome J., Song D.X. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In Proc. of the Network and Distributed System Security Symposium, 2005, 17 p.

[6] Sutton M., Greene A., Amini P. Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley Professional, 2007, 576 p.

[7] American fuzzy lop. Available at: https://lcamtuf.coredump.cx/afl/, accessed 01.11.2022.

[8] Godefroid P., Levin M.Y., Molnar D. SAGE: whitebox fuzzing for security testing: SAGE has had a remarkable impact at Microsoft. Queue, vol. 10, issue 1, 2012, pp. 20-27.

[9] Cadar C., Dunbar D., Engler D. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008, pp. 209-224.

[10] Chipounov V., Kuznetsov V., Candea G. S2E: a platform for in-vivo multi-path analysis of software systems. SIGPLAN Notices, vol. 46, issue 3, 2011, pp.:265-278.

[11] Cha S.K., Avgerinos T. et al. Unleashing mayhem on binary code. In Proc. of the IEEE Symposium on Security and Privacy, 2012, pp. 380-394.

[12] Arzt S., Rasthofer S. et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. ACM SIGPLAN Notices, vol. 49, issue 6, 2014, pp 259-269.

[13] Бородин А.Е., Горемыкин А.В. и др. Поиск уязвимостей небезопасного использования помеченных данных в статическом анализаторе Svace. Труды ИСП РАН, том 33, вып. 1, 2021 г., стр. 7-32. DOI: 10.15514/ISPRAS–2021–33(1)–1 / Borodin A., Goremykin A. et al. Searching for Taint Vulnerabilities with Svace Static Analysis Tool. Programming and Computer Software, vol. 47, issue 6, 2021, pp. 466-481.

[14] Schubert P.D., Hermann B., Bodden E. PhASAR: an inter-procedural static analysis framework for C/C++. Lecture Notes in Computer Science, vol. 11428, 2019, pp. 393-410.

[15] D'Silva V., Kroening D., Weissenbacher G. A survey of automated techniques for formal software verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, issue 7, 2008, pp. 1165-1178.

[16] Иванников В.П., Белеванцев А.А. и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 231-250. DOI: 10.15514/ISPRAS-2014-26(1)-7 / Ivannikov V.P., Belevantsev A.A. et al. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, issue 5, 2014, pp. 265-275.

[17] Arroyo M., Chiotta F., Bavera F. An user configurable clang static analyzer taint checker. In Proc. of the 35th International Conference of the Chilean Computer Science Society (SCCC), 2016, pages 1-12.

[18] Calcagno C, Distefano D. Infer: an automatic program verifier for memory safety of C programs. Lecture Notes in Computer Science, vol. 6617, 2011, pp. 459-465.

[19] Беляев М.В., Шимчик Н.В. и др. Сравнительный анализ двух подходов к статическому анализу помеченных данных. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 99-116. DOI: 10.15514/ISPRAS-2017-29(3)-7 / Belyaev M.V., Shimchik N.V. et al. Comparative analysis of two approaches to static taint analysis. Programming and Computer Software, vol. 44, issue 6, 2018, pp. 459-466.

[20] Bodden E. Inter-procedural data-flow analysis with IFDS/IDE and Soot. In Procю of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis, 2012, pp. 3-8.

[21] Reps T., Horwitz S., Sagiv M. Precise interprocedural dataflow analysis via graph reachability. In Proc. of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1995, pp. 49-61.

[22] Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40. DOI: 10.15514/ISPRAS-2016-28(1)-2 / Koshelev V.K., Ignatiev V.N. et al. SharpChecker: Static analysis tool for C# programs. Programming and Computer Software, vol. 43, issue 4, 2017, pp. 268−276.

[23] Lattner C., Adve V. LLVM: a compilation framework for lifelong program analysis & transformation. In Proc. of the International Symposium on Code Generation and Optimization, 2004, pp. 75‑86.

[24] Reps T., Horwitz S., Sagiv M. Precise interprocedural dataflow analysis via graph reachability. In Proc. of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1995, pp. 49-61.

[25] Shimchik N.V., Ignatyev V.N. Vulnerabilities Detection via Static Taint Analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 3, 2019. pp. 177-190. DOI: 10.15514/ISPRAS-2019-31(3)-14.

[26] Shimchik N., Ignatyev V., Belevantsev A. Improving accuracy and completeness of source code static taint analysis. In Proc. of the 2021 Ivannikov Ispras Open Conference (ISPRAS), 2021, pp. 61-‑68.

[27] A wrapper script to build whole-program LLVM bitcode files. Available at: https://github.com/travitch/whole-program-llvm, accessed 01.11.2022.

[28] Juliet C/C++ 1.3. Available at: https://samate.nist.gov/SARD/test-suites/112, accessed 01.11.2022.

Информация об авторах / Information about authors

Никита Владимирович ШИМЧИК – младший научный сотрудник ИСП РАН. Его научные интересы включают статический анализ программного обеспечения.

Nikita Vladimirovich SHIMCHIK is a researcher at ISP RAS. His research interests include static analysis of programs.

Валерий Николаевич ИГНАТЬЕВ, кандидат физико-математических наук, старший научный сотрудник ИСП РАН, доцент кафедры системного программирования факультета ВМК МГУ. Научные интересы включают методы поиска ошибок в исходном коде ПО на основе статического анализа.

Valery Nikolayevich IGNATYEV, PhD in computer sciences, senior researcher at Ivannikov Institute for System Programming RAS and associate professor at system programming division of CMC faculty of Lomonosov Moscow State University. He is interested in techniques of errors and vulnerabilities detection in program source code using static analysis.

Андрей Андреевич БЕЛЕВАНЦЕВ – доктор физико-математических наук, ведущий научный сотрудник ИСП РАН, профессор МГУ. Сфера научных интересов: статический анализ программ, оптимизация программ, параллельное программирование.

Andrey Andreevich BELEVANTSEV – Dr.Sc., Leading Researcher at ISP RAS, Professor at MSU. Research interests: static analysis, program optimization, parallel programming.

DOI: 10.15514/ISPRAS-2022-34(6)-5

Introducing Programming Language Metrics

T.R. Fayzrakhmanov, ORCID: 0000-0001-5013-4523 <tim.fayzrakhmanov@gmail.com>
Innopolis University,
1, Universitetskaya Str., Innopolis, 420500, Russia

Abstract. We introduce possibly the first approximation of *programming language metrics* that represent a spectrum over 70 unique and carefully gathered *dimensions* by which any two programming languages can be compared. Based on those metrics, one can evaluate her own “best” language, and to demonstrate how complex feelings such as “simplicity” and “easy to use”, often found as arguments in language debates and advertisements, can be decomposed into clear and measurable pieces. We put the collection as a completely separate open-source file (here as an appendix) so that everyone can participate in eliciting new and interesting dimensions used in programming languages research, development, and use. Metrics can find their use to compare languages, define requirements, create rankings, give tips for language designers, and simply provide a bird’s-eye view on existing languages features found in the wild.

Keywords: programming languages; metrics; comparison; analysis

For citation: Fayzrakhmanov T.R. Introducing Programming Language Metrics. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 67-84. DOI: 10.15514/ISPRAS-2022-34(6)-5

Acknowledgements. We thank Eugene Zouev (Innopolis University), Nikolay Shilov (Innopolis University), Nikolay Kudasov (Innopolis University), and an anonymous reviewer from Ivannikov ISP RAS for providing useful suggestions, additional resources and corrections.

Система метрик для языков программирования

Т.Р. Файзрахманов, ORCID: 0000-0001-5013-4523 <tim.fayzrakhmanov@gmail.com>
Университет Иннополис,
420500, Россия, г. Иннополис, ул. Университетская, д.1

Аннотация. Мы представляем, возможно, первое приближение метрик языков программирования, которые представляют собой спектр из более чем 70 уникальных и тщательно собранных измерений, по которым можно сравнивать любые два языка. Основываясь на метриках, человек может самостоятельно определить "лучший" для него язык и продемонстрировать, как сложные чувства, такие как "простота" и "легкость в использовании", часто встречающиеся в продвижении и спорах о том какой язык лучше, могут быть разложены на четкие и измеримые части. Мы разместили коллекцию в виде отдельного файла с открытым исходным кодом (здесь в качестве приложения), чтобы каждый мог принять участие в поиске новых и интересных измерений, используемых в практике, исследованиях, и разработке языков программирования. Метрики могут найти свое применение для сравнения языков, определения требований, создания рейтингов, советов разработчикам языков, а также просто для получения представления о возможностях в существующих языках программирования.

Ключевые слова: языки программирования; метрики; сравнение; анализ

Для цитирования: Файзрахманов Т.Р. Система метрик для языков программирования. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 67-84. DOI: 10.15514/ISPRAS-2022-34(6)-5

Благодарности. Мы благодарим Евгения Зуева (Университет Иннополис), Николая Шилова (Университет Иннополис), Николая Кудасова (Университет Иннополис) и анонимного рецензента ИСП РАН за полезные предложения, дополнительные ресурсы и исправления.

1. Introduction

The hot debates in comparison of programming languages have been known for years. "What is the *best* programming language?" – is one of the most frequently asked questions when one encounters a plethora of available options: 100 languages with thousands to millions of active users worldwide up to 8,945 in total (Table 1). Despite such a precious freedom to choose, in practice, leads to the *paradox of choice* where the more options you have, the more time it takes to settle on a final decision.

Scientific community in the analysis and comparison of programming languages have tried to "nail down" this question multiple times giving an *objective* answer to both: "Who is the best?" in general [1, 2] or for a particular area [3, 4]. However, besides inaccessible scientific jargon for an ordinary language user, no single research can cover so many languages with so many purposes at once.

In this work we propose an alternative approach. Instead of searching for an abstractly best language among an ever-growing number of options and realizing in advance that the choice still depends on numerous factors (user preferences, current infrastructure, etc.), we simply suggest collecting *top language aspects*. In other words, the aspects (further "*programming language metrics*") that have always influenced our positive and anti-choices, and proven to be useful over a long period of time.

The definition of *best*, then, (Section 4) will be a simple formula: *subset* of aspects that must be necessarily included in a language plus their *weights* that distribute the consideration importance within.

Табл. 1. Количество языков программирования
Table 1. The number of programming languages

Total	
(the total number of languages ever created)	
8,945	HOPL Historical Encyclopedia (till 2005) [5]
4,217	Programming Language DataBase [6]
Notable	
(languages that are influential or proved their existence)	
878	Rosetta Code's List of Programming Languages [7]
690	Wikipedia's List of Programming Languages [8]
560	Available in GitHub's Advanced search [9]
370	"Primary" in the annual GitHub report [10]
Popular	
("top" languages with thousands to millions of active users)	
52	IEEE Spectrum index [11]
50-100	TIOBE Programming Community index [12]
42	StackOverflow Developer Survey [13]
28	PYPL Popularity index [14]

2. Related works

Our filter out criteria for works in the comparison and analysis of programming languages were studies that (1) directly identify the list of programming languages metrics, (2) describe possible ways for an ordinary user to measure them, and (3) give methods to define "best" in terms of given metrics and measured scores. Within those constraints, we found no previous work. However, [15] written by Jean E. Sammet 50 years ago is the closest published study.

Although Sammet has not provided a generalized list of metrics with methods for applying them in defining the best option, we admit that (1) she stressed the importance of a language assessment to be *dependent* on numerous factors, e.g. on a viewpoint of a user, implementor, or application area [15, p. 245]; (2) even though her work was found *retrospectively*, the method presented in **Metrics use** (Section 4) is already implicitly used in evaluation of languages for "a user wishing to write a payroll program" [15, tables I, V, and VI].

The rest of the studies were primarily focused on:

- Taking a limited number of metrics (e.g., “running time” or “memory usage”) and evaluating the *best* language among the existing few. For example, within a specific area (bioinformatics [3], robotics [4], economics [16]) or generically ([1, 2, 17-19);
- Introducing a *specific* metric/s (e.g., “popularity”, “impact” [20], “syntactic complexity” [21], “structural complexity” [22]) without further *generalization* with other metrics.

In this work, we considered all types of studies to make the table of metrics as complete as possible.

3. Method

In collecting metrics, we followed no particular method or order. We tried to scrutinize as much literature and resources as we could, which can effectively cooperate in eliciting useful and easy-to-distinguish metrics from an end-user perspective. Besides referenced literature, it involves taking a list of notable languages (Table 1) going through the websites of each, reading the advertising text, specifications, language references, and they-provided comparison with other competitors.

3.1 Terminology

There are many words to describe *dimensions* by which objects, be they programming languages or apples, can be *differentiated* with each other. To make our mappings between words and meaning clearer throughout the text, we want to explicitly distinguish the following terms:

- ***Dimension*** (*aspect, property, indicator, attribute, characteristic, criteria, parameter)* – a *quality* associated with an object
- ***Metric*** – a set of qualities *and* a method of quantifying (measuring) it
- ***Feature*** – a quality that beneficiary distinguishes one object *from others* in its class

We use “dimension” (and its synonyms) to signify the most atomic aspects of a language, “metric” as a set of dimensions that can be represented *numerically*, and “feature” to simply provide a colloquial language used in languages advertising or keywords to search solutions in the web.

Our terminology implies that *quality* of an object is taken for granted and means what common sense suggests us. *Quantity* represents the state of being in a certain quality. For example, “five apples” can be considered as immeasurable qualitative state of having “five” (not used here) or as a *measurable quantitative* state of being in 5 pieces. Binary states (used in the metrics of a type “language supports *X*”) are also accounted as measurable quantitative states of the amount of two (“red apple” is 1 and lacking the “red” is 0).

3.2 Metrics vs. features

A common practice to think of any consumer product, which we believe any programming language essentially is, is in terms of “features”. Features can be considered as a *common language* that are used by both product creators in advertising as well as end-users in product perception.

In this work, it was tempting to present neutrally-oriented list of metrics in terms of features as it is the most popular way for a computer language to be promoted and picked up in the wild. However, if we decided to do so, the table would have become *suggestive*, implying that those features are rather *requirements* for a “perfect language” we are seeking for, than the *dimensions* by which we simply want to compare. Table 2 shows the difference. Thus, we decided to keep the list neutral and add features (whenever possible) right under the metric names to simply provide an additional information.

Табл. 2. Разница между метрикой и «фитчей» языка
Table 2. The difference between a language metric vs. feature

Language Metric	**Language Feature**
Definition and Purpose	
A *method* to measure (quantify) a quality of an object in a neutral manner	A *quality* of an object that demonstrates an advantage(s) over others in its class
The main difference	
A metric cannot be introduced without a method of measuring it	A feature, similar to a feeling, can be introduced even if there is no clear way to measure it
Examples	
Performance (mostly nouns)	*Fast* (mostly adjectives)
Number of seconds required to compile and/or run a program	Whatever the numbers are, it feels really fast comparing to others
Compiler size	*Lightweight*
Lines of code or size in bytes of a language compiler or VM	We may not know exactly but the language weights really nothing comparing to others

4. Result

We introduce the full list of metrics in Appendix A. We have made the appendix *self-contained*. It is a completely separate document with its own description, legend, references, and contributors list. We wanted to make it easily printed, shared, and updated independently of the article itself. As such, some of the parts that are already present in this article might be duplicating in the appendix.

4.1 Disclaimer

We do not pretend the list to be *complete* nor we believe it is reasonable to do so. As the field progresses, there will be always new unique ways to measure language aspects, similar to those of software metrics. The attempt is to make at least a *first* approximation of what programming language metrics might be, what one can measure in general, and how they can be used in practice.

5. Metrics use

In this section, we introduce a simple method of how to define and compute your “best” language using a simple table, the list of metrics, and the measurement data.

5.1 Background

Being able to compare similar objects around us and picking the “best one” among available options is one of the essential cornerstones for an *effective* everyday life. Being able to compare *programming languages* and choosing the best one for a particular problem is probably the essential cornerstone for an effective *programmer* life.

Languages are often advertised and perceived in terms of intuitive *feelings* such as “simple” (Python [23]), “fast” (C [24]), “delightful” (Elm [25]) or even “magical” and “sacred” (LISP [26]). Those feelings, collectively, make us prefer one language over another and, thus, shape our *favorite choices*. However, when it comes to the precise definition of what those feelings actually *mean*, how they can be assessed in practice, and how they affect our final choice(s), the details always elude from the scene. Programming languages are very versatile inventions, and to understand why the same language can be considered as the *best* for one and the *worst* for another, we need a *method* of

dismantling complex feelings, features, and the notion of "best" into something that can be effectively *measured*.

5.2 Procedure

To demonstrate the method, we will be using a simple example. Suppose our goal is to find the "best fit" language out of the three: L1, L2, L3 (names are chosen deliberately abstract to eliminate language affections). The question is "How do we know what language is the best among selected?" To do so, we first identify *dimensions* by which they can be compared.

Step 1. *Skim through programming language metrics and pick the ones that "feels" right, essential, or important*

Suppose we selected Popularity, Documentation, Standard Library, Performance, and Expressivity (further as Popl, Docs, Stdlib, Perf, and Expr for brevity). Then,

Step 2. *Create a table where rows are languages and columns are metrics*

In our case, the table will look like the following:

	Popl	**Docs**	**Stdlib**	**Perf**	**Expr**
L1	-	-	-	-	-
L2	-	-	-	-	-
L3	-	-	-	-	-

Before we start fulfilling the table,

Step 3. *Distribute the importance (weights) per each of the metric*

We do so *before* fulfilling the table because metric importance affects not only the computing procedure for the final choice but how *carefully and precisely* should we measure the scores. For example, according to some ranking, we may find out that L1 has 2nd place in Popl, and L2 – 20th. If we decided a place in Popl isn't *that* important for us, we may no longer waste our time trying to refine the scores by other rankings, we can simply move on to measuring something else that is *more* important. So, let us say we decided to make the distribution as follows:

	Popl	**Docs**	**Stdlib**	**Perf**	**Expr**
Weight	5%	15%	30%	40%	10%
L1	-	-	-	-	-
L2	-	-	-	-	-
L3	-	-	-	-	-

As we can see the sum of all weights is equal to 100%. In practice, however, when we, say, have 15 metrics, it becomes difficult to distribute importance manually to sum them back to 100%. Instead, we suggest simply giving metrics a "place" or "points" (say, from 1 to 10), and compute the corresponding percentages automatically. For example:

	Popl	**Docs**	**Stdlib**	**Perf**	**Expr**	
Weight	2	4	5	8	3	
Sum:	2 +	4 +	5 +	8 +	3	= 22
Normalize:	2/22	4/22	5/22	8/22	3/22	
Weight (result):	=0.09	=0.18	=0.23	=0.36	=0.14	
In %	9%	18%	23%	36%	14%	

Despite using points, we were able to "normalize" them back to percentages so that they sum up to 100% and roughly correspond to our previous manual distribution.

As a result, the table may look as follows:

	Popl	**Docs**	**Stdlib**	**Perf**	**Expr**
Weight	2	4	5	8	3
In %	9%	18%	23%	36%	14%
L1	-	-	-	-	-
L2	-	-	-	-	-
L3	-	-	-	-	-

The second (grayed out) row is optional and can be removed completely. However, we recommend to keep it and, with the help of spreadsheet software, use it to "interactively" adjust points so that the computed weights in percentage looks *desirable*.

Step 4. *Measure metric scores for each of the language and fulfill the table. Make sure all scores have a numeric value*

Step 4 must be the most difficult and important one as everything else depends on its data. However, measurements details are out of scope of this article. We will simply assume we were able to get the results that are more or less "accurate":

	Popl	**Docs**	**Stdlib**	**Perf**	**Expr**
Weight	2	4	5	8	3
In %	9%	18%	23%	36%	14%
L1	2pl	7p	82pkg	3.1ms	300L
L2	20pl	3p	117pkg	1.7ms	155L
L3	13pl	5p	63pkg	0.5ms	170L

We do not need to fit our metric scores into a particular system of units or scale. Scores can be completely "raw". What is important is that they are *numerical*. For example, Popl can be a place in some ranking as TIOBE [12]; Docs can be a sum of abstract points (say, +1 for coverage, readability, nice-looking, etc.); Stdlib – a number of available packages in it; Perf – time in milliseconds needed to run a test-bench program; and Expr could be the lines of code for the program we run in Perf.

Step 5. *Set the polarity for each of the metric (e.g., "higher is better" or "lower is better", where binary metrics are not the exception) and place them on a separate line or near the names*

We used higher ↑ and lower ↓ is better at the end of the names:

	Popl$^{\downarrow}$	**Docs**$^{\uparrow}$	**Stdlib**$^{\uparrow}$	**Perf**$^{\downarrow}$	**Expr**$^{\downarrow}$
Weight	2	4	5	8	3
In %	9%	18%	23%	36%	14%
L1	2pl	7p	82pkg	3.1ms	300L
L2	20pl	3p	117pkg	1.7ms	155L
L3	13pl	5p	63pkg	0.5ms	170L

Before we continue, we may do an additional step:

Step 5.1 (optional) *Identify best scores per each of the column, and write them out on a separate "Best [score]" line*

This step is completely optional and serves rather as an intermediate phase. It shows how close *visually* (by counting highlights) each language approximates to the best sampled scores:

	Popl$^{\downarrow}$	**Docs**$^{\uparrow}$	**Stdlib**$^{\uparrow}$	**Perf**$^{\downarrow}$	**Expr**$^{\downarrow}$
Weight	2	4	5	8	3
In %	9%	18%	23%	36%	14%
Best	**2pl**	**7p**	**117pkg**	**0.5ms**	**155L**
L1	**2pl**	**7p**	82pkg	3.1ms	300L
L2	20pl	3p	**117pkg**	1.7ms	**155L**
L3	13pl	5p	63pkg	**0.5ms**	170L

It's time, however, to calculate how *actually* close each language approximates to the best sampled scores considering the weights. In other words, "Who is the best?" among our three. Depending on

how we would define “best”, we might have two strategies. First is to compute proximity relative to the *unreal* best scores (0th place in Popl and 0ms in Perf). Second – relative to the *real* “Best” scores taken from the previous table (2nd place in Popl and 0.5ms in Perf). We will take the first strategy and remove “Best” line altogether. We do so because (1) taking the second strategy doesn't change the order of “winners”, (2) we found it simpler to compute, and (3) we want the highlighted line, which is now is used by “Best”, to be taken by the real winner (L1, L2, or L3).

Step 6. *Calculate the final score for each of the languages using the following algorithm 1:*

```
For each row [language]:
   For each column [metric in a language]:
      Take score value v
      Take maximum value in column max
      Take metric weight w
      If column polarity is 'higher is better':
         Compute v/max × w
      Otherwise ('lower is better'):
         Compute |v/max − 1| × w
         Add the result to language score S
   [After all metrics are processed]
   Language score S is ready
   Put S on a separate column 'Score'
[After all languages are processed]
Evaluation is completed
Best language is the one with the biggest S
```

Algorithm 1. Best Language Evaluation

Procedure visually:

	Popl↓	**Docs**↑	**Stdlib**↑	**Perf**↓	**Expr**↓	**Score**
L1	2pl	7p	82pkg	3.1ms	300L	0.42
	|2/20 – 1| ×0.09 =0.081 +	7/7 ×0.18 =0.18 +	82/117 ×0.23 =0.16 +	|3.1/3.1 – 1| ×0.36 =0 +	|300/300 – 1| ×0.14 =0 =	0.421
L2	20pl	3p	117pkg	1.7ms	155L	0.54
	|20/20 – 1| ×0.09 =0 +	3/7 ×0.18 =0.077 +	117/117 ×0.23 =0.23 +	|1.7/3.1 – 1| ×0.36 =0.16 +	|155/300 – 1| ×0.14 =0.068 =	0.535
L3	13pl	5p	63pkg	0.5ms	170L	**0.64**
	|13/20 – 1| ×0.09 =0.032 +	5/7 ×0.18 =0.128 +	63/117 ×0.23 =0.124 +	|0.5/3.1 – 1| ×0.36 =0.3 +	|170/300 – 1| ×0.14 =0.06 =	0.644

Procedure formally:

$$S_{\text{best}} \begin{cases} S_1 = m_1[w_1] + m_2[w_2] + \ldots + m_j[w_j] \\ S_2 = m_1[w_1] + m_2[w_2] + \ldots + m_j[w_j] \\ \vdots \\ S_n = m_1[w_1] + m_2[w_2] + \ldots + m_j[w_j] \end{cases},$$

where

$$m_j = \begin{cases} \dfrac{v_j}{max}, \text{ if polarity is "higher is better"} \\ \left|\dfrac{v_j}{max} - 1\right|, \text{otherwise ("lower is better")} \end{cases} \quad (1)$$

and

- S_{best} is the best language score among n languages;

- S_n is the final score for the language *n*;
- w_j is the weight (importance) of the metric *j*;
- m_j is the computed score value of the metric *j*;
- v_j is the "raw" score value of the metric *j*;
- *max* is the biggest numeric value for *j* among $S_{1...n}$.

We call (1) as the "Formula of Choice". We read it as following: the best language S_{best} among available $S_{1...n}$ is the one which has the biggest sum of metric scores $m_{1...j}$ given their weights $w_{1...j}$.

So, after we have computed all the language scores, we can finalize our table with:

Step 6. *Sort the table by Score in descending order, and add a Place column enumerating languages from 1*

	Popl↓	**Docs**↑	**Stdlib**↑	**Perf**↓	**Expr**↓	**Place**	Score
Weight	2	4	5	8	3		
In %	9%	18%	23%	36%	14%		
L3	**13pl**	**5p**	**63pkg**	**0.5ms**	**170L**	**1**	0.64
L2	20pl	3p	117pkg	1.7ms	155L	2	0.54
L1	2pl	7p	82pkg	3.1ms	300L	3	0.42

Column "Place" will give us an ability to keep/see language places even if we decide to sort the table by other columns (e.g., shuffle languages by the largest number of packages).

5.3 Discussion

When we originally looked at the data, we were expecting L2 to become our "top" language. However, the calculation gave it the 2nd, which made us suspect an error in calculations. After a closer look (and double-checking estimates), we understood that L2 is simply *3x times slower* than the winner in Perf, which we decided to be the *most* important aspect in the table. Even though, Stdlib of L2 is larger *almost twice*, its importance is still *lower*. When it comes to the rest of the metrics, they seemed simply compensating each other: L2 is slightly better at Expr, however, slightly worse at Docs, whereas Popl felt completely discarded due to its very low importance.

These slightly unexpected results led us to draw the following conclusions:

1) Weights have to match the *actual expectations* of the author (originally, they have been put artificially without author's internal agreement).
2) Even if weights were in a perfect harmony with us, such cases cannot be excluded, which would require us to start *metric refinement*. The latter means what we have said at the very beginning – the more important metric is, the more effort one should invest into its score elicitation.

5.4 Metric composition

For the sake of simplicity, at the very beginning of the subsection 4.2 (Step 1 and 3), we used metric only as *independent* variables to form our comparison. However, using Step 6 and the "Formula of Choice", we can elaborate the method. We can *compose* metrics as if they are *building blocks* for defining other "high-order" metrics, or (as we are interested in this section) *features* and *feelings*. For example:

"Easy to learn" = Expressiveness↑⟨6⟩ + Documentation↑⟨4⟩ + REPL↑⟨2⟩;

"Easy to write" = Syntactic complexity↓⟨5⟩ + Code formatting↑⟨3⟩;

"Easy to run" = Compiler portability↑ + Supporting platforms↑;

"Easy to debug" = Error hints↑⟨7⟩ + Compilation speed↓⟨3⟩;

"Easy to find help" = *max* (Popularity↑, Technical support↑);

"Fast" = Runtime $speed_{Rust}$ / Runtime $speed_{C}$ (How fast is C relative to Rust)

where

- *"Aspect"* = $\text{metric}_1^p\langle\text{weight}_1\rangle$ *op* $\text{metric}_2^p\langle\text{weight}_2\rangle$ ...;
- *"Aspect"* can be a new (composite) metric, feeling, or feature;
- Angle brackets designate *weight* of the metric (in relative units, say, from 1 to 10);
- Lack of brackets means all metrics share the same importance;
- *p* is the polarity of the metric: higher ↑ or lower ↓ is better;
- *op* is how we want to combine metrics to produce new aspect (e.g., by summation, division, taking max, etc.)

For example, we defined an *"Easy to learn"* as a sum of three aspects: Expressiveness of code, available Documentation, and the presence of REPL. The importance within was distributed using relative points to get percentages automatically (as we did in Step 3). In our case, they come to 50%, 33%, 16% accordingly. *"Easy to run"* we defined as the sum of Compiler portability and Supporting platforms. We missed weights, which mean they are distributed equally: 50% and 50%. Finally, *"Easy to find help"* is simply the metric that is best manifested in the language: either Popularity or direct Technical support, where the weight of whichever metric is chosen will be 100%. The rest of examples should be self-explanatory.

These simple rules of composition give us unlimited power in defining feelings, features, and other high-order metrics that would otherwise be difficult to express. We believe that the idea of combining metrics could be an interesting tool for making *sound arguments* in the endless emotionally-driven language debates. This perspective could make metric composition to be uncharted territory for further research and exploration.

6. Conclusion

The current work presents possibly the first approximation of programming language metrics. We provided an open-source document (to which we welcome to contribute) with over 70 unique programming language aspects that can be used to pragmatically compare languages, define requirements, create rankings, and have an overview of available language features. We have presented the "Formula of Choice" to determine "best language" for your own needs using a simple table with few calculations. We have presented the method of *metrics composition* to decompose complex feelings and features, such as "simplicity" and "easy to use", into more measurable pieces.

We hope that this information can serve as a useful guidance for the analysis and comparison of programming languages in the never-ending debates and constantly emerging options.

References / Список литературы

[1] Nanz S., Furia C.A. A Comparative Study of Programming Languages in Rosetta Code. In Proc. of the IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 778-788.

[2] Prechelt L. An Empirical Comparison of Seven Programming Languages. Computer, vol. 33, issue 10, 2000, pp. 23-29.

[3] Fourment M., Gillings M.R. A comparison of common programming languages used in bioinformatics. BMC Bioinformatics, vol. 9, issue 1, 2008, article no. 82, 8 p.

[4] Pembeci İ., Hager G. A Comparative Review of Robot Programming Languages, Report CIRL-Johns Hopkins University, 2003, 29 p.

[5] Pigott D.J., Axtens B.M. HOPL; Online Historical Encyclopaedia of Programming Languages. Available at: https://hopl.info/, accessed Aug. 11, 2020.

[6] Programming Language DataBase. Available at: https://pldb.com/, accessed Aug. 13, 2022.

[7] Programming Languages. Rosetta Code. Available at: https://rosettacode.org/wiki/Category:Programming_Languages, accessed Nov. 04, 2022.

[8] List of programming languages. Wikipedia. Available at: https://en.wikipedia.org/w/index.php?title=List_of_programming_languages&oldid=972639589, accessed Aug. 13, 2020

[9] GitHub | Advanced Search.GitHub. Available at: https://github.com/search/advanced, accessed Nov. 04, 2022.
[10] The state of open source software | Top Languages. Available at: https://octoverse.github.com/2022/top-programming-languages, accessed Nov. 16, 2022.
[11] Top Programming Languages 2022. IEEE Spectrum. Available at: https://spectrum.ieee.org/top-programming-languages-2022, accessed Dec. 20, 2022.
[12] TIOBE Programming Community index. Available at: https://www.tiobe.com/tiobe-index/, accessed Dec. 20, 2022.
[13] Stack Overflow Developer Survey 2022. Available at: https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022, accessed Dec. 20, 2022.
[14] Carbonnelle P. PYPL PopularitY of Programming Language index. Available at: http://pypl.github.io/PYPL.html, accessed Jul. 16, 2020.
[15] Sammet J.E. Problems in, and a pragmatic approach to, programming language measurement. In Proc. of the Fall Joint Computer Conference, 1971, pp. 243-251.
[16] Aruoba S.B, Fernández-Villaverde J.A Comparison of Programming Languages in Economics. Working Paper 20263. National Bureau of Economic Research, 2014, 20 p.
[17] Boom H.J., de Jong E. A critical comparison of several programming language implementationsю Software: Practice and Experience, vol. 10, issue 6, 1980, pp. 435-473,
[18] Alomari Z., Halimi O.E. et al. Comparative Studies of Six Programming Languages. arXiv:1504.00693, 2015, 71 p.
[19] Al-Qahtani S.S., Pietrzynski P. et al. Comparing Selected Criteria of Programming Languages Java, PHP, C++, Perl, Haskell, AspectJ, Ruby, COBOL, Bash Scripts and Scheme. Revision. arXiv:1008.3434, 2010, 149 p.
[20] Delorey D.P., Knutson C.D., Giraud-carrier C. Programming language trends in open source development: An evaluation using data from all production phase sourceforge projects, In Proc. of the Second International Workshop on Public Data about Software Development, 2007, 5 p.
[21] MacLennan B.J. Simple metrics for programming languages. Information Processing & Management, vol. 20, no. 1, 1984, pp. 209-221.
[22] MacLennan B.J. The Structural Analysis of Programming Languages. Report NPS52 81-009, Naval Postgraduate School, 1981, 37 p.
[23] van Rossum G. Python reference manual. Technical Report CS-R9525. NCWI (Centre for Mathematics and Computer Science), 1995, 59 p.
[24] Kernighan B.W., Ritchie D.M. The C programming language. 2nd edition. Pearson, 1988, 272 p.
[25] Elm - delightful language for reliable web applications. Available at: https://elm-lang.org/, accessed Nov. 04, 2022.
[26] G. Steele. Common LISP: the language. 2nd updated edition. Digital Press, 1990, 1029 p.

A. Appendix. Programming language metrics

This appendix is a collection of over 70 unique *programming language metrics*. The purpose of this section is to provide *dimensions* (features, properties, aspects) by which any two computer languages can be qualitatively and quantitatively compared. These metrics can be used to analyze languages, define requirements, create rankings, provide tips for language designers, or simply give a bird's-eye view on existing language features. The list is based on metrics commonly used in programming language research, development and use, as well as the years of author and contributors personal experience. The collection is open-source and can be downloaded as a separate PDF file at https://github.com/timfayz/language-metrics.

A.1 Contribution

To contribute new metrics, typo fixes, or suggest any other improvements, please send an email to tim.fayzrakhmanov@gmail.com, or make a pull request / open an issue at https://github.com/timfayz/language-metrics. Please, specify your full name, public email, and affiliation if necessary.

A.2 Legend

Metrics are grouped into nine basic categories:

1) *User experience* – a user background affecting the language use;
2) *Language recognizability* – how popular the language is;
3) *Language infrastructure* – surrounding documentation and libraries;
4) *Language development and support* – maintenance, user support, and tooling;
5) *Language special features* – coding experience and special-purpose features;
6) *Language implementation and programs* – compiler and its generated executable files;
7) *Language specialization and design* – focus and syntactic/semantic design decisions;
8) *Language definition* – specification, formalization, and standardization;
9) *Language origin* – by whom, when, and why the language was originally conceived.

Each metric has an ordinal number, name, indicators for measuring score, and examples of a user feedback. The order of categories and metrics within is by potential "impact factor" for an *ordinary end-user* rather than by the impact factor for a potential language designer.

First column contains:

1) **Metric name** with a polarity sign: ↑ "higher or support is better", ↓ "lower or absence is better", and ○ "neutral or depends";
2) *Feature names* found in the advertising descriptions of languages (should be read as "*Language is / has / supports ...*");
3) Typical examples of languages with a good demonstration score (based on public information, author/contributors experience, with no supporting references).

Second column contains a set of indicators for measuring metric "score". If many, indicators can be added together or used individually to adjust the desired accuracy.

Third column describes typical positive "+" or negative "–" end-user perception (usually emotional ones) that have been found "in the wild" (forums, comments, contributors/author experience). Sometimes we put content of the third column in the second (after a long dash "—") to save some vertical space.

A.3 Metrics table

v0.4 (Updated 22 Dec, 2022)

	Metric name(polarity) *Feature name* Typical representative	**How to measure** Common › indicators for measuring metric score	**Typical end-user perception** Positive + or negative – comments found in the wild, when the score is high/low according to metric's polarity
	User Experience		
1.	**Familiarity**↑	› *N* of years coding in the language	+ "It is easier to code in because I already know the language"
2.	**Similarity**↑ C, C++, C# Pascal, Modula, Oberon	› Language is similar to other languages known by the user	+ "The language is really easy to grasp because it looks similar to others"
	Language Recognizability		
3.	**Popularity**↑ *Popular* *Mainstream* *Rich set of libraries* *Community support* Python, JavaScript	› Rank of the language in popularity ranks/surveys: TIOBE [12], PYPL [14], IEEE Spectrum [11], StackOverflow Developer Survey [13], GitHub's State of Octoverse [10] If manually: (the order reflects an ease of checking)	+ "Language must be safe to learn because a lot of people already use it and there must be a reason for it" + "Language must be actively developed* and its development won't be abandoned soon" + "There are plenty of tutorials, examples, snippets, answers to get started"

3.	**Popularity**↑ (cont.) *Popular* *Mainstream* *Rich set of libraries* *Community support* Python, JavaScript	› Wiki page is available › Is in "Popular" category at GitHub's search [9] › Reddit community is available + *N* of members › *N* of questions at StackOverflow [27] › *N* of packages at GitHub [9] › *N* of references/tutorials in web searches › *N* of books written › YouTube videos are available › Job openings are available	+ "It's probably easier to find a job" * Language development might be stagnating even if it is still actively used or considered popular. That is why we included "Development" as a separate metric
4.	**Trendiness**↑ *Trendy* *"Rising star"* Haskell, Python, Rust	› N of stars in public repository compared to the date of the project inception › Language has a surge of interest in newsgroups, conference talks and media	+ "The language seems promising. If I start using it now, it may payoff in the future (new jobs, niches, technological advantage)"
	Language Infrastructure		
5.	**Documentation**↑ *Easy to read* *Comprehensive* *Full of examples* PHP, C#, Go	› Language has "official documentation", "reference manual", or "programmer's guide" that: › Clearly describes how to get started › Written in a clear/informal manner › Has a wide coverage › Contains illustrative code examples › Well-linked with other parts of documentation › Loads quickly	+ "With good examples in documentation I can easily start prototyping my own project" + "I can easily find an answer to my questions concerning the language" – "It is almost impossible to use and learn language without a well-written documentation"
6.	**3rd-party Resources**↑ *Rich community support*	› *N* of textbooks available (for various kind of users; from novices to experienced developers) › Online resources: tutorials, articles, posts › Q&A websites › Videos	+ "It is great when language has a lot of additional resources, tutorials, etc. that explain the same language from different angles, and for different users"
7.	**Standard Library**↑ *Rich/Clean stdlib* *"Batteries included"* Go, Python, Java, C++	› *N* of packages available in standard library › Language is following exhaustive vs minimalistic standard library approach	+ "Rich standard library means I can build many applications without switching to unreliable 3rd-party libraries that might be buggy or no longer supported"
8.	**3rd-party Libraries**↑ *Rich ecosystem* JavaScript, Python, C++	› *N* of packages available on GitHub or language's own repository network	+ "The more packages available in the wild, the faster I can create my own solution, just by using someone else's work"
	Language Development and Support		
9.	**Development**↑ *Actively Developed* Python, C++	(the overall language development dynamics) › How recently was the stable release › *N* of releases per month/year › *N* of commits per month/year	+ "If the language is actively developed, then it's not going to *"die"* soon, and so we can rely on it" + "Bugs reported in the previous version(s) are to be fixed in the next one"
10.	**IDE support**↑ *Supported by many IDEs* Java	› *N* of 3rd-party IDEs supporting the language IDE support = syntax highlight, syntax checker, code formatter, auto-completion, refactoring, code search, debugging, linter, etc. (each feature gives "point")	+ "I can use language in my favorite IDE" – "Without IDE support (like syntax, error highlighting, autocompletion, and such), the modern use of language is almost impossible" (if ↓)
11.	**Milestone**↑ *Stable*	› Language has reached version 1.0 (ie. its library API, syntax, and language constructs became fixed)	+ "Language API isn't in complete flux, so we can rely on it without worrying of breaking changes in the next update"
12.	**Backward-compatibility**○ *Backwards-compatible* C++, JavaScript	› Every new release keeps language API, syntax, and language constructs backward compatible with previous release(s)	+ "My codebase can rely on the API it was originally written in and yet keep updating compiler for possible performance improvements"

13.	**Technical support**↑ *24/7 Technical support*	› Language provides a service with direct human-based technical support	—
	Language Special Features		
14.	**Garbage collection**° *Automatic memory management* Go, Java, Python, C#	› Language provides a garbage collector (GC) —— + “Language takes care of my resources so I don’t need to think about manual memory allocation and deallocation”	– “Programs in the language with automatic memory control are memory hungry and probably cannot be used for embedded systems” – “Language does not give me manual memory control to do my own (unsafe) stuff”
15.	**Type safety**↑ *Strong typing* *Static type-checking* Rust, Go, Haskell	› Language provides any form of runtime or/and compile-time type checking (ie. prevents a program to perform illegal operations on values that do not have appropriate data type)	+ “Programs written in this language are reliable, less error-prone, and always behave the way I defined them to behave” – “I’m so annoyed with the constant type errors that I simply cannot write programs”
16.	**Memory safety**↑ *Safe/Memory-safe* Rust, Go, Kotlin	› Language provides any form of mechanisms to prevent illegal memory access in a program (runtime/compile-time checks for buffer/stack overflows, dangling pointers, double freeing, etc.)	+ “Programs written in this language are more safe and less prone to memory leaks”
17.	**Type richness**↑ *Rich types* Haskell, Scala, Typescript	› Language has high descriptive power in its type system (eg. support for interfaces, generics, algebraic, high-order, dependent types, etc.)	+ “Language allows me to define complex types, data and program behaviour as well as verify them prior execution”
18.	**Exception handling**↑ *Exception handling* C++, Python, Java	› Language provides mechanisms for handing unexpected runtime errors without immediate crash / resuming execution	+ “Language allows me to handle runtime errors such that I am able to recover execution flow or exit properly”
19.	**Concurrency**↑ *Parallel computing* *Multithreaded* *Coroutines* C/C++, Go, Erlang	› Language supports any form of parallel execution and multithreaded computing: › Heavyweight threads (also native, or operating system threads) › Lightweight coroutines (also fibers, generators, or “green threads”)	+ “Language allows me to do parallel computing (ie. utilize as much computing power as possible) in a manageable way”
20.	**Instruction-level parallelism**↑ *Parallel computing* *SIMD programming* C/C++	› Language supports any form of vectorized operations or "SIMD programming” (eg. explicit directives for vectorized/”streaming” data structures, operations, loop unrolling, etc.)	+ “Language allows me to do professional optimization of my code to get the maximum performance and efficiency of my programs”
21.	**GPU computing**↑ *Parallel computing* *Scientific computing* C/C++	› Language provides well-supported libraries or primitives to dispatch execution onto GPU(s)	+ “Language allows me to accelerate my programs with the power of GPU”
22.	**Distributed computing**↑ *Distributed computing* C/C++, Julia, Erlang	› Language provides mechanisms to distribute a single program or execution flow upon several physically separated machines (incl. separated by network)	+ “Language allows me to do highly scalable computation across multiple machines”
23.	**Message passing**↑ *Distributed computing* Erlang, Smalltalk, Java	› Language supports sending messages between abstract objects which can be objects, parallel processes, subroutines, functions or threads	+ “Language gives me a single model of objects that simply communicate with each other, no matter whether they are functions or parallel processes”
24.	**Reflection**↑ *Reflective* Go, Julia, JavaScript	› Language provides constructs to “see” and modify its own code (normally, at runtime; eg. accessing variable names, function signatures, etc.)	+ “I can dynamically (at run-time) access meta-data of language constructs (eg. get a name of a class, variable, function, etc.), which allows me to write a more generic code and do all kinds of static/dynamic code analysis”

25.	**Lazy evaluation**° Haskell, Io, Clojure, Scala	› Language supports holding up the evaluation of an expression until its value is needed › Language allows switching back to or explicitly forcing (normal) "eager evaluation" when needed —— + "My code can be more efficient in terms of memory and performance because values don't need to be computed if they aren't going to be used"	+ "In lazy language it is possible to define infinite lists and elegantly handle streams of data" – "Lazy evaluation brings a certain amount of memory bloat, and requires too much knowledge of the program and algorithms to get the benefits" – "It is not clear when exactly side effects are going to happen and so it is hard to debug"
26.	**Lambda expressions**↑ Haskell, Scheme, many...	› Language supports anonymous functions	+ "I can construct higher-order functions or use them as values to return from other functions"
27.	**Package manager**↑ *Package manager* C# NuGet, Python pip	› Language allows to download and install packages and dependencies using one of its (built-in) CLI commands	+ "Language comes with its own package manager so I don't need to install some 3-rd party packages to get things up and running"
28.	**Doc generator**↑ *Doc comments* Java, C#	› Language supports "documentation comments" (formatting tags) and is able to generate (HTML) pages based on these annotations	+ "I can embed parts of program documentation directly into my source code and get nice-looking pages for free"
29.	**Build system**↑ *Native build system* Zig	› Language allows to write build scripts in itself without using external tools or other languages (such as Bash, make, CMake, Maven, etc.)	+ "It is great that I don't need to learn other building tools and their cryptic languages in order to automate my project building routines"
30.	**Error hints**↑ *Smart compiler* *Helpful debug messages* Elm	› Language compiler or run-time environment provides error messages that are instructive enough to understand how to fix them	+ "Language is really good in helping to fix my code. I get not only an error message but also a hint how to fix it"
31.	**Code formatting**↑ *No more formatting wars* Go fmt, C clang-format	› Language compiler can automatically reformat code to follow default/user-defined coding standards	+ "I don't need to spend time following numerous and over-complicated coding styles to format my code. Let the language do it instead"
32.	**Macros**↑ *Metaprogramming* C/C++, Zig, Nim	› Language supports any form of metaprogramming or defining "macros" to execute logic during compile time	+ "I can do a lot of prepossessing during compile time so that runtime is not occupied by unnecessary computations"
33.	**Native IDE**↑ *Built-in IDE* Eiffel → EiffelStudio	› Language offers its own integrated development environment —— + "Native IDE may give much better integration than 3rd-party alternatives"	– "I don't want to change my environment just because of the language" (if only native IDE available) – "It's unlikely that build-in IDE is better than my current"
34.	**REPL**↑ *Interactive* Python, Scala	› Language has interactive Read-Eval-Print-Loop mode	+ "It is easy to play with the language and test code snippets"
35.	**Embedding**↑ *Embeddable* Lua, Tcl, Red, Lisp	› Language (as "guest") can run in *N* of ("host") languages or applications	+ "Language can be used as a *scripting* language to automate repetitive tasks in my favorite app or a host language (eg. Bash in shell, Python in Blender, Lua in World of Warcraft)"
36.	**Bindings**↑ *FFI support** Python/Go ← C/C++ Kotlin ⇄ Java	› Language supports direct function calls (bindings) from *N* other languages without wrappers or special API	+ "My code can easily use the libraries of other language(s)" *FFI (Foreign Function Interface) – a language is capable to call functions written in another language providing so called "bindings" (primarily to C)
37.	**Transpilation**↑ *Transpiled* Haxe, TypeScript, Elm	› Language is able to compile code into source code of other *N* (high-level) languages	+ "I can keep writing my code in the language to the benefits of which I already get used to but also benefit from other language(s) infrastructure, libraries, performance, etc."

38.	**IR access**↑ *Open interface* *Deep language integration* C# or VB (using Roslyn)	› Language, for each compilation step, provides internal intermediate representation (IR) export (eg. pre-processed source code, parse tree, syntax tree, intermediate code, etc.)	+ “Probably language provides a good amount of data for implementing advanced IDE features (debuggers, static analyzers, code formatters, dependency checkers, visualizers, etc.)”
39.	**Unicode support**↑ *UTF-8 support* Java, C#, Go, Swift	› Language supports Unicode Standard for representing characters in strings or identifiers	+ “I can work with special characters such as emoji in my strings or use foreign language identifiers”
40.	**GOTO support**° C/C++, Go, Fortran	› Language supports “goto” statements for unconditional jumps to specific program locations (usually by means of labels)	+ “I can create custom control structures where the built-in ones do not satisfy my (professional/low-level) needs” – “GOTO statements can be easily abused by unskilful programmer and lead to notorious Spaghetti code”
	Language Implementation and Programs		
41.	**Compilation speed**↑ *Fast compilation* C, Go, Zig	› How fast compiler compiles programs in s/ms/ns	+ “Recompilation time in this language is really short, which allows me to make the feedback loop between code changes and results short”
42.	**Runtime speed**↑ *Fast* C/C++, Rust, Zig	› How fast programs run in s/ms/ns	+ “Language is blazingly fast, programs written it run really quickly”
43.	**Compile-time memory footprint**↓ *Low memory usage* C, Pascal, Forth	› The amount of memory in bytes needed to compile a program (or *while* compiling the program)	+ “I can compile big projects without thinking that I will run out of memory on my machine”
44.	**Runtime memory footprint**↓ *Low memory footprint* C/C++, Fortran, Rust	› The amount of memory in bytes that a program uses *while* running	+ “Programs written in this language hardly use any RAM (compared to others), which means the compiler does good optimizations, emits efficient code and probably suitable for embedded systems”
45.	**Compiler/VM size**↓ *Lightweight* Lua	› Size of language compiler or VM in LOC/bytes	+ “Language is lightweight, minimalistic and (possibly) embeddable”
46.	**Executable size**↓ *Compact programs* *Slim binaries* C, Oberon, Zig	› Size of executables, including the ones for VM, in bytes (eg. with default compiler options)	+ “Programs are small, possibly fast, and may fit into embedded systems”
47.	**Compiler/VM portability**↑ *Portable* C/C++, Java	› *N* of platforms the language compiler/VM can run on	+ “I can compile my code on many platforms” or “I can run compiler/VM on many platforms”
48.	**Executable portability**↑ *Cross-compiled* *Portable, Transpiled* C/C++, Java	› *N* of “target platforms” the language programs can be run on	+ “I can write code once and run it anywhere (WORA)”
49.	**CLI complexity**↓ *Simple to use* Go	› *N* of `$ language commands` › *N* of command line `--options`	+ “Command Line Interface of the language is easy and simple to use and remember”
50.	**Self-hosting**↑ *Self-hosted* Zig, Go, Rust	› Language implementation is written in itself —— + “If language is self-hosted, it can be considered “serious”, “production ready” and independent from others”	+ “Language can get more contributions to its compiler by people who before would only work on the standard library”

51.	**Open-source**↑ *Open Source* *OSI-approved* Python, Go	› Language (compiler) source code is open-source and available for download, modification, recompilation, distribution, static linking and commercialization	+ “Open-source is good because anyone can contribute to language development: do code reviews, fix bugs, write modules, documentation, etc.” – “If open-source, it is not clear who is responsible for the project and fixing bugs. It can be abandoned at any time”
52.	**License**° *MIT license*	› Language license type (MIT, GPL, BSD etc.)	+ “Nonrestrictive license types give a language freedom to be not confined to any single ownership, and prevent attempts to be company or technology specific”
	Language Specialization and Design		
53.	**Paradigm**°	› Language presents itself as following a particular or multiple paradigms (eg. procedural, object-oriented, functional)	+ “I like when the language mix different paradigms because I can approach problems using a paradigm that is the most effective for the solution”
54.	**Visual language**° *Visual Programming* DRAKON, Scratch	› Language has a graphical representation and can be used as a visual modeling or programming language	+ “I like the visual expression of my code to better understand and manipulate my program”
55.	**Esoteric language**° Brainfuck	› Language is considered as “esoteric” (esolang)	+ “I can use the language as a form of software art to show off my skills”
56.	**Educational language**° Logo	› Language is specifically designed or can be used to introduce pure computer science ideas (also known as “tiny”, “small”, or “first”)	+ “I can use the language to concentrate on pure ideas without being distracted with unnecessary infrastructural details”
57.	**Domain-specialization**° *Used by professionals* *Hardened by industry* R in statistics Matlab/Python in scientific computations	› Language became one of the standard tools used in a certain domain —— – “Language is not safe to invest time because if I use it, I’ll stuck in its domain” + “I’ll be able to do what other professionals do”	+ “Language is safe for time investment because other people in my domain use it already” + “Typical problems have been solved already” + “It will be easier to find a job (or simply, you don’t find any without having skills in it)”
58.	**Platform-orientation**° *Deep integration* Apple –› Swift Microsoft –› C#	› Language is primarily driven by or developed for a certain platform and its infrastructure	+ “Language provides the best integration experience for this platform” – “If I use this language I will probably *stuck* in its infrastructure”
59.	**Expressiveness**↑ *Expressive, Powerful* Python	› Length of program in LOC to express a typical problem comparing to the same task written in another language [8]	+ “Language is easy to write, it is concise, short and elegant; code do not repeat itself” (if ↑) – “Language is difficult to write, read, and maintain; code grows fast” (otherwise)
60.	**Syntactic complexity**↓ *Laconic, Concise* *Elegant, Simple* Lisp ↓, C++ ↑	› *N* of production rules language grammar has › *N* of keywords	+ “Language is simple, elegant, concise and has a small learning curve” (if ↓) – “Language is bulky, complex, bloated and has a steep learning curve” (otherwise)
61.	**Syntactic coherence**↑ *Clean syntax* APL, Brainfuck ↓ Elm ↑	› Ratio between word- vs ASCII-based operators, keywords, and constructs › Keywords are in/distinguishable › Use of ASCII in identifiers is not/allowed › Lack/use of underscores in reserved identifiers	– “Code is cryptic, noisy, ripples in eyes and difficult to follow” (if ↑) + “Code is clean, consistent and easy to follow” (otherwise)
62.	**Semantic complexity**↓ *Simple* Go	› *N* of language constructs › *N* of built-in operators	+ “The less construct language has, the less I need to remember”

63.	**Semantic coherence**↑ *Consistent design* *Easy to learn* *Coherent* Lisp	› Language constructs are composable with each other › Language follows a paradigm "everything is an expression"	+ "Language feels well-designed, coherent, and easy to learn. It has a small number of constructs, everything is composable with each other, and there are little/no special rules or exceptions"
64.	(Syntactic/Semantic) **Homoiconicity**° *Code as data* Lisp, Scheme	› Code can be directly interpreted as data (ie. as language built-in structures), and inversely, data can be executed as code	+ "Language feels magical and self-referential" + "I can easily generate programs or do program analysis written in that language"
65.	**Design independence**° *Inspired by X* *Designed from scratch* X is a well-known language	› Language design is "inspired" by other languages, or it is a continuation of "language family"	+ "If the language is inspired by *X*, and *X* wasn't bad, then the new one is going to be at least as good as its predecessor(s)" + "If a language designed from scratch, it is probably fresh and ambitious enough to give a good "punch" to others"
	Language Definition		
66.	**Specification**↑ C/C++, Java	› Language has a normative Specification with a complete in-/semi-/formal definition of its form (syntax) and behaviour (semantics) › Specification includes the specification of standard library	– "If specification is too big, the language is probably over-complicated to hold in one's programmer head and so, difficult to learn" + "If specification is simple/short, the language can be probably easily re-implemented or ported to new architectures"
67.	**Standardization**° *Standardized* C/C++	› Specification is based on the consensus of different parties that may include firms, interest groups, standards organizations or governments	+ "It is good that I can have independent compilers for the same code base and switch them if there is performance or development stagnating issues" – "Language has become huge, bulky and slow-moving because its design is now dispatched to (big) standardization committee rather than (small group of) individual(s)"
68.	**Formal syntax**↑ SQL, C#, Go, Python	› Specification includes the formal grammar of language syntax (normally in EBNF)	+ "I can use it to write a parser for language analysis or as a basis for its reference implementation"
69.	**Formal semantics**↑ *Formalized* Standard ML, PL/I	› Specification includes the definition of language semantics in some theory or formal system (eg. Set theory + First-order logic, Category theory, etc.)	+ "Behaviour of my programs can be verified with mathematical rigour" + "Language can be used for mission- and safety-critical software systems"
	Language Origin		
70.	**Origin**° *Came from X* X is a well-known company or eminent university	› Language was born as an academic, industry, or a hobby project —— + "If the language was born in industry, it is probably battle-tested, pragmatic, and understandable by a normal human being"	– "If the language was born in academia, probably it is *not* well suited for the real industrial software development" + "If it was born in academia, it is well-designed, has a mathematical rigour, formally defined behaviour, and potentially verifiable programs"
71.	**Author**° *Designed by X* X is a prominent person	› Author name(s) who designed, implemented or gave rebirth to the language	+ "If the author is well-known developer/researcher, then the language should be well-designed too"
72.	**Initial purpose**° *Designed for X*	› The problem domain the language was originally(historically) designed for	+ "If the language was created for X, then it should probably do it well"
73.	**Age**° *Developed since X* X suggests maturity	› Date or *N* of years from the first release or exposition	+ If the language is developed over many years, then it must be mature, has comprehensive documentation, and vast infrastructure" – "If the language is too old, then it is slow developed, its design overloaded with special

			cases and exceptions, and it is overall conservative towards new advancements”

Information about the author / Информация об авторе

Timur Rasimovich FAYZRAKHMANOV – software developer, PhD student, researcher. Research interests: programming languages, knowledge representation and processing, knowledge organization and reuse, formalization, generic systems modeling, World Digital Mathematics Library, graphics, alternative development environments, block-based approach.

Тимур Расимович ФАЙЗРАХМАНОВ – разработчик ПО, аспирант, исследователь. Сфера научных интересов: языки программирования, представление и обработка знаний, организация и повторное использование знаний, формализация, обобщённое моделирование систем, Всемирная цифровая математическая библиотека, графика, альтернативные среды разработки, блочный подход.

DOI: 10.15514/ISPRAS-2022-34(6)-6

Математические и программные модели задач технического зрения робототехнических комплексов на основе микропроцессоров “Эльбрус”

[1] *Н.А. Бочаров, ORCID: 0000-0002-8504-2060 <bocharov.na@phystech.edu>*
[1] *Н.Б. Парамонов, ORCID: 0000-0002-6999-0968 <paramonov_n_b@mail.ru>*
[2] *О.А. Славин, ORCID: 0000-0002-9541-469X <oslavin@isa.ru>*
[1] *К.А. Суминов, ORCID: 0000-0002-3759-6026 <suminov_k@mcst.ru>*

[1] *Институт электронных управляющих машин им. И.С. Брука,*
119334, Россия, г. Москва, ул. Вавилова, д. 24
[2] *Федеральный исследовательский центр "Информатика и управление" РАН,*
117312, Москва, проспект 60-летия Октября, 9

Аннотация. Создание новых поколений автономных робототехнических комплексов, систем распознавания и систем технического зрения в целом невозможно без использования современных компьютерных технологий. В данной статье представлены модели системы технического зрения роботов на базе микропроцессоров "Эльбрус". Были разработаны модели задач обнаружения, классификации и сегментации. Теоретические и экспериментальные результаты были получены на существующих и перспективных микропроцессорах "Эльбрус". Показано, что микропроцессоры "Эльбрус" могут быть основой бортовой системы технического зрения. Полученные авторами результаты свидетельствуют о перспективах импортозамещения в области робототехники.

Ключевые слова: моделирование; Эльбрус; e2k; VLIW; бортовой вычислитель; техническое зрение; робототехника

Для цитирования: Бочаров Н.А., Парамонов Н.Б., Славин О.А., Суминов К.А. Математические и программные модели задач технического зрения робототехнических комплексов на основе микропроцессоров “Эльбрус”. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 85-100. DOI: 10.15514/ISPRAS-2022-34(6)-6

Mathematical and software models of technical vision tasks of robotic complexes based on “Elbrus” microprocessors

[1] *N.A. Bocharov, ORCID: 0000-0002-8504-2060 <bocharov.na@phystech.edu>*
[1] *N.B. Paramonov, ORCID: 0000-0002-6999-0968 <paramonov_n_b@mail.ru>*
[2] *O.A. Slavin, ORCID: 0000-0002-9541-469X <oslavin@isa.ru>*
[1] *K.A. Suminov, ORCID: 0000-0002-3759-6026 <suminov_k @mcst.ru>*

[1] *PJSC “INEUM named after I.S. Brook”,*
24, Vavilova st., Moscow, 119334, Russia
[2] *ISA FRC RAS,*
9, 60-letiya Oktyabrya av., Moscow, 117312, Russia

Abstract. The creation of new generations of autonomous robotic complexes, recognition systems and vision systems in general is impossible without the use of modern computer technologies. This article presents models

of the robot vision system based on Elbrus microprocessors. Models of detection, classification and segmentation tasks were developed. The models are based on the number of arithmetic operations required to perform a forward pass. The models take into account such features of Elbrus microprocessors as: number of executing devices, pipeline, data pre-pumping, clock frequency, etc. Theoretical and experimental results were obtained on existing and promising "Elbrus" microprocessors. It is shown that Elbrus microprocessors can be the basis of an on-board vision system. The results obtained by the authors indicate the prospects of import substitution in the field of robotics.

Keywords: modeling; Elbrus; e2k; VLIW; onboard computer; technical vision; robotics

For citation: Bocharov N.A., Paramonov N.B., Slavin O.A., Suminov K.A. Mathematical and software models of technical vision tasks of robotic complexes based on “Elbrus” microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 85-100 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-6

1. Введение

Задачи технического зрения в настоящее время являются важным направлением развития области искусственного интеллекта [1]. Создание новых поколений автономных робототехнических комплексов (РТК), систем распознавания и систем технического зрения в целом невозможно без использования современной вычислительной техники. При этом для решения подобного рода задач в настоящее время активно применяются и разрабатываются [2-4] вычислительные комплексы с использованием специализированных ускорителей. Использование таких ускорителей обусловлено неспособностью микропроцессоров (МП) общего назначения решить такие задачи за поставленное время вследствие большой вычислительной нагрузки.

Тем не менее, граница применимости таких специализированных ускорителей при проектировании вычислительных комплексов для решения подобных задач часто определяется эмпирически, в особенности для вычислительных комплексов (ВК) на основе МП серии «Эльбрус» [5], поскольку, в силу особенности архитектуры МП «Эльбрус» (Very Long Instruction Word**,** VLIW), сложно и не всегда возможно оценить сложность и ресурсоемкость решения на базе имеющихся решений подобных задач, реализованных для систем с МП других архитектур.

Одним из важных и актуальных применений бортовых систем с МП серии «Эльбрус» являются бортовые вычислители и системы технического зрения. В ходе проведенных в МЦСТ работ показано, что вычислители на основе МП серии «Эльбрус» могут и успешно используются для решения задач технического зрения как в серверном [6] так и в бортовом [7] режимах. В 2017 году в МЦСТ разработан и внедрен бортовой вычислитель на базе МП Эльбрус-4С, обеспечивающий достаточную производительность для автономного движения робота на скоростях до 40 км\ч. В работе [8] было обосновано, что для обеспечения корректной работы на такой скорости достаточным условием является работа системы технического зрения с производительностью не менее 10 кадров в секунду.

Появление новых МП серии «Эльбрус», таких как Эльбрус-8СВ, Эльбрус-2С3 и Эльбрус-16С [9], а также средств вычислительной техники на их основе [10] открывает новые перспективы перед разработчиками РТК. Высокая производительность, обеспечиваемая новыми МП серии «Эльбрус», позволит создавать бортовые вычислительные комплексы, способные решать задачи технического зрения на РТК с использованием отечественной программно-аппаратной платформы, а появление МП «Эльбрус» шестого поколения должно еще больше повысить производительность существующих решений и открыть возможности для решения новых задач в этой области.

Одними из самых частых задач в области технического зрения являются задачи сегментации, обнаружения и классификации объектов на изображении. Для обнаружения объектов одним из популярных методов, находящих широкое применение, является метод Виолы-Джонса (Viola–Jones object detection) [11]. Для решения задачи классификации, как и задач

сегментации изображений, как правило, с большим успехом используются сверточные нейронные сети различных архитектур.

Специальные ускорители активно создаются с целью ускорения решения, в том числе, именно таких задач, но в силу дороговизны, большей сложности и ограничений в использовании систем со специализированными ускорителями встает вопрос об определении условий, в которых для решения рассматриваемых задач достаточно использовать МП общего назначения, например, из ряда «Эльбрус», без специальных ускорителей.

2. Особенности аппаратно-программной платформы «Эльбрус» для решения задач технического зрения

Архитектура процессоров VLIW является альтернативой для архитектур OOOSS (Out-Of-Order SuperScalar), главным отличиями которой является использование так называемых широких команд, позволяющих выразить параллельность множества операций в ассемблере, а также использование оптимизирующих компиляторов [12], переупорядочивающих команды во время компиляции программы, а не во время ее выполнения.

Компилятор для VLIW обладает гораздо большим окном операций для перемешивания, чем имеется на этапе исполнения у аппаратуры OOOSS. Это позволяет в некоторых случаях лучше выявлять независимые операции для их параллельного исполнения. С другой стороны, OOOSS обладает дополнительной информацией о параллелизме, доступной в динамике исполнения, например, значения адресов операций чтения и записи. Это позволяет лучше выявлять параллелизм в некоторых других ситуациях.

Широкие команды в МП серии «Эльбрус» содержат набор элементарных операций, которые можно запустить на исполнение в одном процессорном такте. Для широких команд процессоров «Эльбрус» с системой команд версии не ниже 4 доступны 6 арифметико-логических устройств (АЛУ), поддерживающих операции с вещественными числами, устройство передачи управления, 3 устройства для работы с предикатами, 6 квалифицирующих предикатов, 4 устройства для команд асинхронного чтения данных – APB (Array Prefetch Buffer), 4 32х-битных литерала для хранения константных значений – LIT. Состав широкой команды представлен на рис. 1 [13].

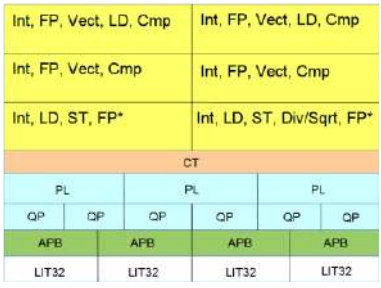

Рис. 1. Парк устройств широкой команды МП «Эльбрус»
Fig. 1. The list of devices of the wide command of MP "Elbrus"

Ядро поддерживает большое количество операций – 25 в скалярном и 41 в векторном режимах. Также стоит отметить, что АЛУ в составе ядра поддерживают выполнение зацепленных друг за другом операций в качестве одной трехаргументной операции, например, операции вида a * b + c. При этом промежуточный результат первой операции не записывается в регистр, а сразу используется в качестве аргумента второй операции. Наличие такой операции является важным при работе со сверточными нейронными сетями, в которых такая комбинация операций является основополагающей для работы всего алгоритма.

Архитектура «Эльбрус» включает в себя многие решения, которые обеспечивают улучшение производительности при работе со сверточными нейронными сетями, в том числе:

- конвейеризация циклов позволяет наиболее эффективно исполнять циклы с

независимыми (или слабо зависимыми) итерациями; в программно-конвейеризированном цикле последовательные итерации выполняются с наложением - одна или несколько следующих итераций начинают выполняться раньше, чем заканчивается текущая;

- методы предварительной подкачки данных – в том числе устройство асинхронной подкачки массивов APB; устройство применяется для асинхронной предподкачки необходимых элементов массива заранее перед использованием, что ускоряет процесс обращения в память во время исполнения программы и уменьшает время ее работы делая более эффективной работу при большом количестве итераций; APB особенно актуально при работе со сверточными нейронными сетями, например, для заблаговременной подкачки очередной порции весовых коэффициентов (ядер сверток).

Также для оптимизации выполнения команд применяются такие техники, как unroll&fuse (unroll&jam) [14], применяемые компилятором для попытки развернуть и слить выполнение цикла там, где это возможно. При этом развертывание цикла повторяет тело нескольких циклов и объединяет необходимые итерации в один развернутый цикл. Использование объединенного цикла может свести к минимуму необходимое количество итераций и снизить частоту промахов в кэш, что актуально при больших объемах параметров-весов нейронных сетей.

Кроме архитектурных решений и механизмов, для МП архитектуры «Эльбрус» реализованы и портированы программные решения и библиотеки, в их числе библиотека EML – Elbrus Media Library, высокопроизводительная математическая и мультимедийная библиотека, представляющая из себя набор разнообразных функций для обработки сигналов, изображений, видео, математических вычислений. Благодаря эффективной реализации библиотеки EML, на некоторых задачах с матрицами с ее использованием ускорение составляет более 20 раз по сравнению с процессором Intel и до 85 раз по сравнению с процессором «Эльбрус» без использования библиотеки EML [15].

Библиотека EML содержит в своем составе разделы:

- Core – для работы с памятью;
- Vector – для работы с векторами;
- Algebra – раздел линейной алгебры включающий пакеты для работы с матрицами и векторами BLAS 1, BLAS 2, BLAS 3, LAPACK.

Это позволяет эффективно выполнять операции линейной алгебры, в том числе с матрицами, что необходимо при работе со сверточными нейронными сетями.

Кроме библиотеки EML, для МП архитектуры «Эльбрус» портирована библиотека Opencv 3.2.0. В настоящее время находится на завершающем этапе работа по портированию для архитектуры «Эльбрус» актуальной версии библиотеки OpenCV 4.5.2, которая также включает в себя модули Opencv_dnn для работы с глубокими нейронными сетями.

На завершающем этапе портирования находится фреймворк для работы с машинным обучением PyTorch для языка Python. Фреймворк PyTorch является одним из самых популярных в мире и активно применяется, в том числе, для решения задач с применением сверточных нейронных сетей различных архитектур. Благодаря его наличию, для МП архитектуры «Эльбрус» становится возможным использование существующих решений задач, а также создание универсальных решений для различных архитектур на его базе.

Кроме того, для архитектуры «Эльбрус» закончена стадия практического тестирования отечественного фреймфорка для обучения и работы с глубокими нейронными сетями – платформа ГНС [16]. Платформа представляет собой клиент-серверное приложение, позволяющее пользователям работать с эффективными реализациями нейронных сетей различных архитектур. Совместно с платформой разработчиком поставляется библиотека, учитывающая архитектурные особенности процессоров семейства «Эльбрус».

Таким образом, большой парк вычислительных устройств (АЛУ) в составе процессоров «Эльбрус», большое количество операций – 25 в скалярном и 41 и векторном режимах [17] за такт на одно ядро (в Эльбрус-8С), такие механизмы, как предварительная подкачка данных, возможность выполнять совмещенные операции и эффективная работа с матрицами, а также большое количество ядер МП дают возможность поставить процессоры «Эльбрус» в ряд между процессорами общего назначения и специализированными процессорами, использующими SIMD-инструкции. А наличие оптимизированных библиотек и фреймворков, обеспечивающих эффективную работу с матрицами и другими математическими вычислениями в совокупности с большой степенью параллельности многих задач, обеспечивающих функционирование систем технического зрения, дают основание полагать, что задачи подобного класса могут эффективно решаться и использованием современных МП серии «Эльбрус».

3. Модель задачи обнаружения

Алгоритмы выделения и распознавания образов на изображении, работающие на основе дерева классификаторов, способны эффективно обнаружить на изображении объекты, совпадающие по своим свойствам, например, форме и цвету, с заранее заданными программистом объектами. Таким образом, эти алгоритмы могут использоваться для поиска заранее известных объектов на изображении. Одним из наиболее распространенных подобных алгоритмов является алгоритм Виолы-Джонса.

Основными операциями при работе алгоритма являются операции умножения и сложения. Произвести оценку количества выполняемых операций сложения и умножения S и M соответственно, можно согласно формуле:

$$S = M = \frac{A * B}{w * h * s^2} * \left(\frac{w * h}{w_0 * h_0}\right)^2 * \sum_{i=0}^{N} S_i * f_i,$$

где A – ширина входного изображения, A – высота входного изображения, w – ширина поискового окна, h – высота поискового окна, w_0 – ширина базового поискового окна, h_0 – высота базового поискового окна, s – относительный размер шага поискового окна, S_i – результат i-го классификатора, представленный значениями 0 и 1 при отсутствии и наличии объекта соответственно, f_i – количество признаков на i-м этапе классификатора N – количество классификаторов в каскаде.

При оценке самого вычислительно нагруженного случая, когда все классификаторы на изображении обнаруживают объект и проводятся все вычисления, количество рассчитываемых признаков составляет более 6000 для каждого из них. При оценке самого вычислительно ненагруженного случая, когда все классификаторы отбрасывают окно на первом этапе, и останавливают дальнейшие вычисления, количество рассчитанных признаков составляет 10 для каждого из них.

Для изображения размером 1280x720 пикселей теоретическая оценка производительности составляет от 1,7 кадра в секунду в наиболее нагруженном и до 1100 кадров в наименее нагруженном случаях. В действительности, за счет каскадной архитектуры классификатора количество выполняемых операций значительно варьируется в зависимости от содержания изображения. Вследствие очень широкого диапазона теоретической оценки производительности для более точной оценки требуются экспериментальные исследования на различных видеорядах.

Для моделирования задачи обнаружения были разработаны программы обнаружения объектов в видеопотоке с использованием метода Виолы-Джонса. Разработаны программы для использования на процессорах архитектуры «Эльбрус», на процессорах Intel, а также для использования на процессорах семейства Intel совместно с видеокартой Nvidia GTX 960 в качестве ускорителя. Программы написаны с использованием языка C++, библиотеки

OpenCV версии 4.5.2 для архитектуры Intel x86-64 и архитектуры «Эльбрус». Также использовались некоторые идеи, опубликованные в [18, 19]. Для версии, использующей видеокарту Nvidia в качестве ускорителя, также использовался фреймворк Nvidia CUDA toolkit версии 10.0 [20].

В библиотеке OpenCV применялся модуль objdetect в версиях, ориентированных на исполнение на универсальных процессорах Intel и «Эльбрус», а также модуль cudaobjdetect содержащий реализации алгоритмов и функций на основе модели программирования CUDA, использующей в качестве вычислителя видеокарту. В указанных модулях для обнаружения объектов использовался класс CascadeClassifier, в том числе, метод этого класса detectMultiScale, применяющий алгоритм Виолы-Джонса к изображению и выполняющий обнаружение объектов разных размеров на изображении. Результатом работы метода является список прямоугольников, заданных координатами на исходном изображении и ограничивающих обнаруженные объекты, если таковые нашлись.

При разработке программной модели для оценки эффективности решения были проведены тесты для различных размеров скользящих окон с различным соотношением сторон, в том числе, квадратных. В результате для дальнейших экспериментов были выбраны окна различных размеров с соотношением сторон 23:28, как дающие наилучший результат. Также были протестированы каскадные классификаторы с различными наборами признаков, в результате чего были выбраны классификаторы haarcascade_frontalface_alt и cuda_haarcascade_frontalface_alt для реализации на универсальном процессоре и с использованием CUDA соответственно.

4. Модель задачи классификации

Среди архитектур сверточных нейронных сетей, дающих хороший результат top 5 accuracy (более 92%) на соревновании ImageNet [21], являются сети VGG16 и VGG19 [22]. Кроме того, сверточная часть этих сетей, в особенности VGG16, часто используется для предварительной обработки (feature detector) для выделения признаков, которые в дальнейшем используются в других приложениях.

Для теоретического обоснования времени выполнения вычислений нейронной сети с архитектурой VGG16 и VGG19, на примере МП серии «Эльбрус», разработана математическая модель вычислений, учитывающая количество операций, производимых при расчетах сети. В табл. 1 представлено необходимое количество параметров и их объем, а также количество и тип необходимых операций при выполнении основных расчетов для VGG19 и VGG16. Слои, помеченные (*), не входят в VGG16.

Табл. 1. Количество параметров и операций для слоев сети VGG19 (VGG16)
Table 1. Number of parameters and operations for network layers VGG19 (VGG16)

№ слоя	слой	размерность входа			количество умножений в слое	количество сложений в слое	количество сравнений в слое	количество параметров в слое
		ш	в	г				
1	conv3-64	224	224	3	89915392	86704128	0	1792
2	conv3-64	224	224	64	1852899328	1849688064	0	36928
	pool2	224	224	64	0	0	2408448	0
3	conv3-128	112	112	64	926449664	924844032	0	73856
4	conv3-128	112	112	128	1851293696	1849688064	0	147584
	pool2	112	112	128	0	0	1204224	65664
5	conv3-256	56	56	128	925646848	924844032	0	295168
6	conv3-256	56	56	256	1850490880	1849688064	0	590080
7	conv3-256	56	56	256	1850490880	1849688064	0	590080
8(*)	conv3-256	56	56	256	1850490880	1849688064	0	590080
	pool2	56	56	256	0	0	602112	0
9	conv3-512	28	28	256	925245440	924844032	0	1180160

10	conv3-512	28	28	512	1850089472	1849688064	0	2359808
11	conv3-512	28	28	512	1850089472	1849688064	0	2359808
12(*)	conv3-512	28	28	512	1850089472	1849688064	0	2359808
	pool2	28	28	512	0	0	301056	0
13	conv3-512	14	14	512	462522368	462422016	0	2359808
14	conv3-512	14	14	512	462522368	462422016	0	2359808
15	conv3-512	14	14	512	462522368	462422016	0	2359808
16(*)	conv3-512	14	14	512	462522368	462422016	0	2359808
	pool2	14	14	512	0	0	75264	0
17	fc4096	1	1	25088	102760448	102760448	0	102764544
18	fc4096	1	1	4096	16777216	16777216	0	16781312
19	fc1000	1	1	4096	4096000	4096000	0	4097000
		Всего VGG19			19646914560	19632062464	4290048	143732904
		Всего VGG16			15483811840	15470264320	4591104	138423208

В табл. 2 представлены значения теоретической оценки времени выполнения расчетов сетей (inference) при рассмотрении идеальной модели процессоров, в которой отсутствуют задержки по памяти, длительность вычислений ограничивается лишь скоростью работы и степенью конвейеризации АЛУ. В действительности, при наличии не идеальности процессов вычисления и существующих потерь, а также ограниченного размера кэш-памяти использование механизма предподкачки, реализованного в МП серии «Эльбрус» [23] механизма APB, а также эффективная реализация умножения матриц блоками, использование схемы unroll and fuse и возможность АЛУ выполнять зацепленные операции умножения и сложения позволяют получить скорость предподкачки, обеспечивающую на ~90% эффективную загруженность АЛУ. Время выполнения T в этом случае можно оценить, как:

$$T = \frac{(Nu + Nm + Ns + Nc) * R * k}{F * C * A * S},$$

где Nu – количество совмещенных операций, Nm, Ns и Nc – количество отдельных операций умножения, сложения и сравнения соответственно, R – разрядность чисел, k – коэффициент эффективности, F – тактовая частота процессора, C – количество ядер, A – количество АЛУ включающих FPU (Floating Point Unit), S – разрядность FPU.

Табл. 2. Теоретическое время выполнения вычислений для сетей VGG16 и VGG19 на МП «Эльбрус»
Table 2. Theoretical execution time for VGG16 and VGG19 networks on MP «Elbrus»

	VGG16				**VGG19**			
количество совмещенных операций * +	15470264320				19632062464			
количество операций *	13547520				14852096			
количество операций +	0				0			
количество операций <>	4290048				4591104			
процессор	Эльбрус-8С	Эльбрус-8СВ	Эльбрус-16С	Эльбрус-2С3	Эльбрус-8С	Эльбрус-8СВ	Эльбрус-16С	Эльбрус-2С3
расчетное время расчета прямого прохода (мс)	137,9	59,7	22,4	179,2	174,9	75,8	28,4	227,4

расчетное количество кадров в секунду	7,3	16,7	44,6	5,6	5,7	13,2	35,2	4,4

Для моделирования задачи классификации были разработаны программы, реализующие вычисления нейронных сетей различных архитектур. Для задачи классификации с использованием архитектуры нейронной сети VGG16 были разработаны следующие программы: универсальная программа на языке С без внешних и архитектурных зависимостей, программа на языке Python 3 с использованием фреймворка машинного обучения с открытым кодом PyTorch для вычислений на универсальных процессорах и видеокарты Nvidia GTX 960 в качестве ускорителя. Также разработана программа с использованием платформы ГНС разработки ГосНИИАС в которой были реализованы вычисления нейронных сетей архитектуры VGG19, а также некоторых других распространенных архитектур. При реализации программ использовались некоторые идеи, опубликованные в [17, 24].

Универсальная программа на языке С осуществляет вычисления прямого прохода (inference) нейронной сети архитектуры VGG16 для изображений размером 224 x 224 x 3. Для реализации версии программы на языке С были использованы только стандартные библиотеки из состава языка программирования С без архитектурных зависимостей, в результате чего программа пригодна для вычислений на универсальных процессорах разных архитектур, т.е. является полностью кроссплатформенной. Также были использованы файл предобученных весов нейронной сети с соревнований ImageNet и файл расшифровки классов. Классификация выполняется среди 1000 классов соревнований ImageNet. При реализации программы, ввиду отсутствия использования архитектурных зависимостей библиотеки EML, позволяющих получить ускорение, было получено большее время обработки одного изображения, чем ожидалось.

Также была разработана модель задачи классификации с использованием Платформы-ГНС. Файлы обученных нейронных сетей для архитектур VGG19, AlexNet, LeNet, ResNet18, ResNet50, MobileNetV1 были обучены и предоставлены ГосНИИАС. Для работы с обученными файлами в комплекте поставляется библиотека, использующая архитектурные зависимости.

5. Модель задачи сегментации

Предложенная Адамом Пашке (Adam Paszke) и др. в работе [25] архитектура нейронной сети с названием ENet была создана для решения задачи сегментации с малыми затратами вычислительных ресурсов. Архитектура сети ENet представляет собой архитектуру кодер-декодер, и ее работа делится на 6 этапов, строящихся из блоков нескольких типов – это initial block, asymmetric bottleneck block, upscale bottleneck block, bottleneck block в трех вариантах, а также fullconvolution block.

Согласно расчетам, для изображения размерностью 640x360x3 необходимы базовые операции в количестве, указанном в табл. 3. Количество указано в миллионах операций.

Табл. 3. Количество основных необходимых операций
Table 3. Number of basic operations required

умножения	сложения	сравнения	деления	извлечение корня
2386	2472	58	30	30

Таким образом, всего необходимо около 5 миллиардов операций. Учитывая возможность выполнять зацепленные совмещенные операции, а также временную зависимость некоторых данных, в общей сложности получается 3.8 млрд операций [25] и 370 тысяч параметров.

Аналогично расчетам для модели сетей VGG19, VGG16, в табл. 4 представлены теоретические оценки времени выполнения расчетов сети (inference) при рассмотрении

идеальной модели процессоров, а также с учетом не идеальности процессов вычисления и существующих потерь, а также размера кэш-памяти, с использованием механизма предподкачки, реализованного в МП серии «Эльбрус», механизма APB, а также при эффективной реализации умножения матриц блоками, использования схемы unroll and fuse и возможности АЛУ выполнять зацепленные операции умножения и сложения. Скорость предподкачки в таком случае достигает значения, обеспечивающего на ~90% эффективную загруженность АЛУ. Время выполнения T в этом случае можно оценить, как:

$$T = \frac{(Nu + Nm + Ns + Nc + Nd + Nq) * R * k}{F * C * A * S},$$

где Nu –количество совмещенных операций, Nm, Ns, Nc, Nd и Nq – количество отдельных операций умножения, сложения, сравнения, деления и вычисления квадратного корня соответственно, R – разрядность чисел, k – коэффициент эффективности, F – тактовая частота процессора, C – количество ядер, A – количество АЛУ, включающих FPU, S – разрядность FPU.

Также в табл. 4 представлена оценка времени для видеокарты GTX 960 и ВК Nvidia tx1 используемого авторами архитектуры ENet в своих тестах.

Табл. 4. Теоретическое время выполнения вычислений для сети ENet
Table 4. Theoretical execution time for the ENet

количество совмещенных операций * +	752					
количество отдельных операций *	441					
количество отдельных операций +	1720					
количество отдельных операций < >	58					
количество отдельных операций /	30					
количество отдельных операций √	30					
вычислитель	Эльбрус-8C	Эльбрус-8CB	Эльбрус-16C	Эльбрус-2C3	GTX 960	Nvidia TX 1
теоретическое время расчета прямого прохода сети (мс)	15,6	6,8	2,5	20,3	1,6	7,6
теоретическое количество кадров в секунду	64,0	147,7	393,8	49,2	618,7	131,3

Для моделирования задачи сегментации изображений была разработана программа сегментации изображений с использованием нейронной сети архитектуры ENet. Программа написана с использованием языка C++, библиотеки OpenCV версии 4.5.2 для архитектуры Intel x86-64 и архитектуры «Эльбрус». Также использовались некоторые идеи, опубликованные в [26-28].

В библиотеке OpenCV использовался модуль opencv_dnn для работы с глубокими нейронными сетями. Также использовались файлы весов предобученной сети Enet.

6. Общая модель технического зрения робототехнических комплексов

На основе разработанных программ для задач обнаружения, классификации и сегментации была разработана единая программная модель, объединяющая в себе все указанные модели, общую программу тестирования, скрипты сборки для Cmake и примеры входных данных. Общая модель написана на языке python и позволяет провести общее тестирование системы

на соответствие необходимым параметрам производительности. Общий объем тестирующей системы с примерами исходных данных для программ составил 1.5 Гб.

Для тестирования разработанных программ применялись различные процессоры семейства «Эльбрус», в том числе 2, 8 и 16 ядерный, а также МП Intel core i7 2600k, серверные МП Intel Xeon 4110 и Xeon e5 2620, выпуска 2016-2017 года, 8 ядерные, 16 поточные и мобильный процессор Intel core i7-8565U производства 2018 года. В качестве спец. ускорителя была использована видеокарта Nvidia GeForce GTX 960 совместно с МП Intel core i7 2600k.

Для проведения эксперимента с обнаружением был выбран один видеоряд со следующими разрешениями: 424 x 240, 640 x 360, 854 x 480, 1280 x 720, 1920 x 1080, 2560 x 1440, 3840 x 2160, также производилось тестирование, и отладка на видеопотоке с веб-камеры в реальном времени.

В качестве входных данных для всех программ, моделирующих вычисления нейронной сети с архитектурой VGG16, использовались изображения из базы данных ImageNet с разрешением 224x224x3. Для нейронных сетей с архитектурами VGG19, AlexNet, LeNet, ResNet18, ResNet50, MobileNetV1 кроме изображений с разрешением 224x224x3, также были использованы изображения в разрешении 32x32x3.

Для задачи сегментации в качестве исходных данных было выбрано видео, снятое на видеорегистратор автомобиля и содержащее движение автомобиля по дорогам общего пользования, в разрешении 1280x720 и наборы кадров, полученные из этого видео. Также производилось тестирование и отладка на видеопотоке с веб-камеры в реальном времени.

При моделировании задачи обнаружения объектов в видеопотоке были получены результаты производительности в зависимости от разрешения изображения для различных размеров поискового окна. На рис. 2 представлены графики зависимости производительности, выраженной в количестве кадров в секунду от разрешения входного видеопотока для вычислительных комплексов на базе различных процессоров, в том числе для ВК, включающего ускоритель GTX 960. Результаты представлены для поискового окна размером 92 x 112.

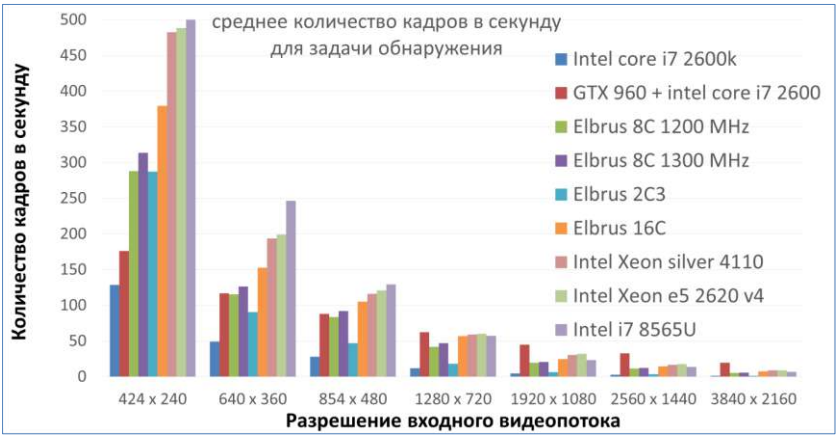

Рис 2. Количество кадров в секунду при решении задачи обнаружения
Fig 2. The number of frames per second when solving the detection problem

Из полученных данных видно, что все протестированные процессоры семейства «Эльбрус» стабильно превосходят Intel core i7 2600k, в том числе двухядерный Эльбрус-2С3. Из результатов видно, что на малых разрешениях использование совместно с процессором ускорителя в виде видеокарты не только не дает выигрыша по времени, но и обеспечивает худшие результаты в отличии от использования только процессора, что связано с особенностью распределенных вычислений с использованием спецускорителей, а именно, с длительной и частой пересылкой маленьких для расчета объемов данных между процессором

и ускорителем. Однако стоит заметить, что в ходе исследования использовалась относительно устаревшая видеокарта, и при использовании современных спецускорителей типа Модуль [30], Элвис [31], Nvidia разница в результатах будет отличаться. Также в ходе эксперимента установлено, что Эльбрус-8С превосходит вычислитель на основе Intel core i7 2600k совместно с видеокартой GTX 960 на разрешениях вплоть до 854x480, а на разрешении 1280x720 Эльбрус-8С показал результат до 47 кадров в секунду, при том, что Эльбрус-2С3 обеспечивает до 18, а Эльбрус-16С до 57 кадров в секунду, в то время как Intel core i7 2600 – всего около 11 кадров, а совместно с видеокартой – 62 кадра в секунду. Эльбрус-16С показал сравнимый с серверными процессорами Intel Xeon результат на больших разрешениях входного видеопотока, а Эльбрус-8С показал отставание от них на 10-30% при вдвое меньшем количестве потоков.

При моделировании задачи классификации изображений для всех программ моделирования расчетов нейронных сетей и используемых архитектур нейронных сетей были получены значения времени выполнения. Для программы реализации вычислений сети VGG16 на языке С были получены результаты, представленные в табл. 6.

Табл. 6. Время выполнения задачи классификации VGG16
Table 6. VGG16 classification task execution time

Процессор	**Эльбрус-2С3**	**Эльбрус-8С**	**Эльбрус-16С**	**Intel i7-2600k**	**Intel xeon silver 4110**	**Intel xeon e5 2620 v4**
однопоточный режим (мс)	34,2	56,4	31,3	16,3	15,7	15,7
многопоточный режим (мс)	18,7	9	3,6	4,4	1,9	2,5

Для реализации сети VGG16 на PyTorch для процессоров из семейства «Эльбрус» были получены результаты для МП Эльбрус-8С. Для других процессоров линейки на данный момент отсутствует реализация фреймворка PyTorch. Также был получен результат для процессора Intel i7 2600k и Intel i7 2600k совместно с ускорителем Nvidia GTX 960. Результаты представлены в табл. 7, значения выражены в миллисекундах.

Таблица 7. Время выполнения задачи классификации VGG16 на Python
Table 7. Execution time of the VGG16 classification task in Python

	Intel i7 2600	**Intel i7 2600 + GeForce GTX 960**	**Эльбрус-8С**
среднее время выполнения	327,1	73,6	194,6
среднее количество кадров в секунду	3,1	13,6	5,1

Для реализации нейронных сетей с архитектурами VGG19, AlexNet, LeNet, ResNet18, ResNet50, MobileNetV1 с использованием платформы ГНС были получены результаты для процессора Эльбрус-8С. Результаты представлены в табл. 8.

Табл. 8. Время выполнения задачи классификации с платформой ГНС
Table 8. Time to complete the classification task with the GNS platform

		AlexNet	LeNet	ResNet18	ResNet34	ResNet50	VGG19	MobileNetV1
224x224x3	время выполнения (мс)	25,1	12,3	40,6	259,2	316,1	266,7	40,6
	кадров в секунду	39,8	81,2	24,6	3,9	3,2	3,7	24,6
32 x 32 x 3	время выполнения (мс)	5,2	3,3	18,7	27,0	28,4	16,5	4,0
	кадров в	191,3	301,5	53,5	37,1	35,2	60,5	252,1

	секунду							

В табл. 9 представлены результаты производительности некоторых нейронных сетей для вычислителей с использованием современных ускорителей, позволяющие произвести относительную оценку [31]. Для видеокарт использовалась CUDA 5.0.05.

При разнице в количестве ядер в 400 раз и их производительности в 20 раз, производительность на задаче меньше всего в 4 – 15 раз.

Табл. 9. Сравнение практических результатов различных вычислителей
Table 9. Comparison of practical results of various computers

Вычислитель	ядер	fp32 Gflops	дата выхода	ALexNet	VGG-19	ResNet-18	ResNet-34	ResNet-50
GTX 1080	2560	8870	05.2016	7,4	79,8	14,8	24,8	50,8
Maxwell Titan X	3072	6140	03.2015	7,6	93,5	17,1	28,8	56,3
Pascal Titan X	3584	10160	08.2016	5,3	55,8	10,1	16,9	35,0
Эльбрус-8С	8	512	10.2018	25,1	266,7	40,6	259,2	316,1

При моделировании задачи сегментации изображений были получены результаты времени выполнения сегментации видео с разрешением 1280x720 для архитектуры сети ENet.

Среднее время выполнения для одного кадра при разрешении входа нейронной сети 640 x 360 на процессоре Эльбрус-8С оказалось 235 миллисекунд или 4.25 кадра в секунду и 310 миллисекунд или 3.2 кадра в секунду для процессора Intel i7 2600k, для NVidia TX1 практический результат составляет 69 миллисекунд или 14.5 кадра в секунду.

7. Заключение

В ходе данного исследования разработаны математические модели вычислений каскадного классификатора, сверточных нейронных сетей с архитектурой VGG16, VGG19, а также ENet. Получено теоретическое обоснование минимального времени выполнения расчетов на процессорах Эльбрус-2С3, Эльбрус-8С, Эльбрус-8СВ и Эльбрус-16С. Разработаны программные модели для решения задачи обнаружения объектов в видеопотоке с использованием OpenCV. Разработаны программные модели для решения задач классификации с использованием программы на Си, программы на Python, а также с использованием ПО «Платформа-ГНС». Разработаны программные модели для решения задачи сегментации с использованием языка C++.

Проведены эксперименты по обнаружению объектов в видеопотоке с различным разрешением, а также по классификации изображений с разрешением 224 x 224 x 3, 32 x 32 x 3 и сегментации изображений с входом нейронной сети 640 x 360. Результаты, полученные в ходе экспериментов, согласуются с теоретическими результатами, основанными на разработанной математической модели.

В результате проведенных экспериментов показано, что использование микропроцессоров Эльбрус-2С3, Эльбрус-8С, Эльбрус-8СВ и Эльбрус-16С без дополнительных ускорителей обеспечивает достаточную производительность при задаче обнаружения для использования в бортовых вычислителях и системах технического зрения автономных роботов вплоть до разрешения 2560x1440, где обеспечивается частота обработки до 14-15 кадров в секунду. Показано, что обеспечивается производительность до 80 кадров в секунду на задачах классификации изображений 224x224x3 с применением ПО ГосНИИАС. Также показана производительность процессора Эльбрус-8С при решении задачи сегментации изображений в 4.3 кадра в секунду. На основе полученных результатов ожидаемый на практике результат для МП Эльбрус-8СВ составляет более 10 кадров в секунду.

Были продемонстрированы результаты и перспективы использования аппаратно-программной платформы «Эльбрус» для решения задач технического зрения, что позволило согласовать требования к вычислителям перспективных автономных роботов.

Список литературы / References

[1]. Смолин В.С., Соколов С.М. Методы использования искусственного интеллекта в составе систем управления с компьютерным видением. Сборник трудов Всероссийской научно-технической конференции «Техническое зрение в системах управления», 2020 г., стр. 37-38 / Smolin V.S., Sokolov S.M. Methods of using artificial intelligence as part of control systems with computer vision. In Proc. of the All-Russian Scientific and Technical Conference on Technical Vision in Control Systems, 2020, pp. 37-38 (in Russian)

[1] Gondimalla A., Chesnut N. et al. Sparten: A sparse tensor accelerator for convolutional neural networks. In Proc. of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 151-165.

[2] Deng L., Li G. et al. Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE, vol. 108, issue 4, 2020, pp. 485-532.

[3] Auten A., Tomei M., Kumar R. Hardware acceleration of graph neural networks. In Proc. of the 57th ACM/IEEE Design Automation Conference (DAC), 2020, pp. 1-6.

[4] Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства "Эльбрус". СПб, Питер, 2013 г., 272 стр. / Kim A.K., Perekatov V.I., Ermakov S.G. Microprocessors and computer systems of the "Elbrus" family. St. Petersburg, Peter, 2013, 272 p. (in Russian).

[5] Лимонова Е.Е., Бочаров Н.А. и др. Оценка быстродействия системы распознавания на VLIW архитектуре на примере платформы Эльбрус. Программирование, том 45, вып. 1, 2019 г., стр. 15-21 / Limonova E.E., Bocharov N.A. et al. Performance Evaluation of a Recognition System on the VLIW Architecture by the Example of the Elbrus Platform. Programming and Computer Software, vol. 45, issue 1, 2019, pp. 12-17. (in Russian).

[6] Бочаров Н.А., Зуев А.Г., Славин О.А. Производительность микропроцессора Эльбрус-8СВ для решения задач технического зрения в условиях ограничений энергопотребления. Известия Южного федерального университета. Технические науки, вып. 1, 2021 г., стр. 259-271 / Bocharov N.A., Zuev A.G., Slavin O.A. Performance of the Elbrus-8SV microprocessor for technical vision tasks under power constraints. Izvestiya SFedU. Engineering Sciences, issue 1, 2021, pp. 259-271 (in Russian).

[7] Бочаров Н.А., Парамонов Н.Б. и др. Производительность вычислительной техники с процессором «Эльбрус-8С» на задачах робототехнического комплекса. Наноиндустрия, вып. 5(82), 2018 г., стр. 79-84. / Bocharov N.A., Paramonov N.B. et al. Performance of computer systems with Elbrus-8C processor for robotic systems tasks. Nanoindustry, issue 5(82), 2018, pp.79-84 (in Russian).

[8] Кожин А.С. Основные проектные решения для процессора «Эльбрус-16С». Наноиндустрия, том 13, вып. S4, 2020 г., стр. 74-75 / Kozhin A.S. Elbrus-16C processor architecture decisions. Nanoindustrija, vol. 13, №. S4, 2020, pp. 74-75 (in Russian)

[9] Бычков И.Н., Лобанов И.Н., Молчанов И.А. Вычислительная техника на основе аппаратно-программной платформы «Эльбрус» для перспективных информационных систем. Приборы, вып. 8, 2018 г., стр. 14-20 / Bychkov I.N., Lobanov I.N., Molchanov I.A. Computer equipment based on «Elbrus» architecture platform for advanced information systems. Instruments, issue. 8, 2018, pp. 14-20 (in Russian)

[10] Viola P.A., Jones M. Robust real-time object detection. International journal of computer vision, vol. 57, issue 2, 2004, pp. 137-154.

[11] Четверина О.А. Методы коррекции профильной информации в процессе компиляции. Труды ИСП РАН, том 27, вып. 6, 2015 г., стр. 49-68 / Chetverina O.A. Methods of Profile Information Correction during Compilation. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 6, 2015, pp. 49-66 (in Russian). DOI: 10.15514/ISPRAS-2015-27(6)-4.

[12] Нейман-заде М.И., Королев С.Д. Руководство по эффективному программированию на платформе «Эльбрус». М., АО «МЦСТ», 2020 г., 178 стр. / Nejman-zade M.I., Korolev S.D. A Guide to Effective Programming on the Elbrus Platform // M., MCST, 2020, 178 p. (in Russian).

[13] Carr S., Guan Y. Unroll-and-jam using uniformly generated sets. In Proc. of 30th Annual International Symposium on Microarchitecture, 1997, pp. 349-357.

[14] Ишин П.А., Логинов В.Е., Васильев П.П. Ускорение вычислений с использованием высокопроизводительных математических и мультимедийных библиотек для архитектуры Эльбрус. Вестник воздушно-космической обороны, вып. 4 (8), 2015 г., стр. 64-68 / Ishin P.A., Loginov V.E., Vasilyev P.P. Computational speed up with use of high efficiency mathematical and multimedia functions. Aerospace Defense Herald, issue 4(8), 2015, pp. 64-68 (in Russian)

[15] Визильтер Ю.В., Желтов С.Ю. Использование глубоких нейронных сетей для анализа данных, управления и оптимизации в перспективных авиационных приложениях. Труды XII мультиконференции по проблемам управления (МКПУ-2019), том 4, 2019 г., стр. 17-20 / Vizilter Yu.V., Zheltov S.Yu. Using deep neural networks for data analysis, management and optimization in promising aviation applications. In Proc. of the XII Multiconference on Problems of Control, 2019, pp. 17-20 (in Russian)

[16] Кожин А.С., Нейман-заде М.И., Тихорский В.В. Влияние подсистемы памяти восьмиядерного микропроцессора «Эльбрус-8С» на его производительность. Вопросы радиоэлектроники, вып. 3, 2017 г., стр. 13-21 / Kozhin A.S., Neiman-zade M.I., Tikhorskiy V.V. Memory subsystem impact on the 8-core «Elbrus-8C» processor performance. Issues of radio electronics, issue 3, 2017, pp. 13-21 (in Russian).

[17] OpenCV Tutorials. Available at: https://docs.opencv.org/3.4/d9/df8/tutorial_root.html, accessed 07.05.2022.

[18] Cascade classifier. Available at: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html, accessed 07.05.2022.

[19] CUDA Toolkit Documentation v10.0.130. Available at: https://docs.nvidia.com/cuda/archive/10.0/, accessed 07.05.2022

[20] Сикорский О.С. Обзор сверточных нейронных сетей для задачи классификации изображений. Новые информационные технологии в автоматизированных системах, вып. 20, 2017 г., стр. 37-42 / Sikorsky O.S. An overview of convolutional neural networks for an image classification problem. New information technologies in automated systems, issue 20, 2017, pp. 37-42 (in Russian)

[21] Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv: 1409.1556, 2014, 14 p.

[22] Limonova E.E., Neimanzade M.I., Arlazarov V.L. Special aspects of matrix operation implementations for low-precision neural network model on the Elbrus platform. Bulletin of the South Ural StateUniversity. Series Mathematical Modelling, Programming & Computer Software, vol. 13, issue 1, 2020, pp. 118-128.

[23] Krizhevsky A., Sutskever I., Hinton G.E. ImageNet classification with deep convolutional neural networks. Communications of the ACM, vol. 60, issue 6, 2017, pp 84-90.

[24] Paszke A., Chaurasia A. et al. Enet: A deep neural network architecture for real-time semantic segmentation. arXiv preprint arXiv:1606.02147, 2016, 10 p.

[25] Deep Neural Networks (dnn module). Available at: https://docs.opencv.org/4.x/d2/d58/tutorial_table_of_content_dnn.html, accessed 07.05.2022.

[26] ENET CPP. Available at: https://github.com/zm0612/ENet-version-CPP, accessed 07.05.2022.

[27] Semantic Segmentation on PyTorch. Available at:: https://github.com/Tramac/awesome-semantic-segmentation-pytorch, accessed 07.05.2022.

[28] Бирюков А.А., Таранин М.В., Таранин С.В. Процессор 1879ВМ6Я. Реализация глубоких сверточных нейронных сетей. DSPA: Вопросы применения цифровой обработки сигналов, том 8, вып. 4, 2018 г., стр. 191-195 / Birjukov A. A., Taranin M. V., Taranin S. V. Processor 1879VM6Ya. Implementation of deep convolutional neural networks. DSPA: Digital Signal Processing Applications, vol. 8, issue 4, 2018, pp. 191-195 (in Russian).

[29] Петричкович Я. Солохина Т. и др. RoboDeus-50-ядерная гетерогенная СнК для встраиваемых систем и робототехники. Электроника: Наука, технология, бизнес, вып. 7, 2020, pp. 52-63 / Petrichkovich Ya., Solokhina T et al. Robodeus: 50-Core Heterogeneous Soc for Embedded Systems And Robotics. Electronics: Science, Technology, Business, issue 7, 2020, pp. 52-63 (in Russian).

[30] cnn-benchmarks. Available at: https://github.com/jcjohnson/cnn-benchmarks, accessed 07.05.2022.

Информация об авторах / Information about authors

Никита Алексеевич БОЧАРОВ – кандидат технических наук, начальник отдела. Сфера научных интересов: вычислительная техника, техническое зрение, робототехника.

Nikita Alekseevich BOCHAROV – Candidate of Technical Sciences, Head of Department. Research interests: computer engineering, technical vision, robotics.

Николай Борисович ПАРАМОНОВ – доктор технических наук, профессор, главный научный сотрудник. Сфера научных интересов: вычислительная техника, техническое зрение, робототехника.

Nikolay Borisovich PARAMONOV – Doctor of Technical Sciences, Professor, Chief Researcher. Research interests: computer engineering, technical vision, robotics.

Олег Анатольевич СЛАВИН – доктор технических наук, главный научный сотрудник. Сфера научных интересов: вычислительная техника, техническое зрение, распознавание образов.

Oleg Anatolyevich SLAVIN – Doctor of Technical Sciences, Chief Researcher. Research interests: computer engineering, technical vision, document recogntition.

Константин Александрович СУМИНОВ – инженер-программист. Сфера научных интересов: вычислительная техника, техническое зрение, робототехника.

Konstantin Alexandrovich SUMINOV is a software engineer. Research interests: computer engineering, technical vision, robotics.

DOI: 10.15514/ISPRAS-2022-34(6)-7

Влияние трансформаций на успешность состязательных атак для классификаторов изображений Clipped BagNet и ResNet

[1] *Е.О. Курденкова, ORCID: 0000-0001-5871-8179 <kurdenkova@ispras.ru>*
[2] *М.С. Черепнина, ORCID: 0000-0003-1186-6718 <i.knaz@yandex.ru>*
[1,3] *А.С. Чистякова, ORCID: 0000-0003-4896-4418 <a.chistyakova@ispras.ru>*
[1] *К.В. Архипенко, ORCID: 0000-0002-8699-889X <arkhipenko@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Мюнхенский технический университет, Германия, 80333 Мюнхен, Арсисштрассе 21*
[3] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*

Аннотация. В нашей статье сравнивается точность классической модели ResNet-18 с точностью моделей Clipped BagNet-33 и BagNet-33 с состязательным обучением в разных условиях. Мы провели эксперименты для изображений, атакованных состязательной наклейкой, в условиях трансформаций изображений. Состязательная наклейка представляет из себя небольшую область атакуемого изображения, внутри которой значения пикселей можно неограниченно менять, что может порождать ошибки в предсказании модели. Трансформации атакованных изображений в данной статье моделируют искажения, появляющиеся в физическом мире, когда смена ракурса, масштаба или освещения изменяет распознаваемое изображение. Наши эксперименты показывают, что модели из семейства BagNet плохо справляются с изображениями в низком качестве. Также мы проанализировали влияние разных видов трансформаций на устойчивость моделей к состязательным атакам и переносимость этих атак.

Ключевые слова: состязательная атака; состязательная наклейка; архитектура BagNet; состязательное обучение; проектируемый градиентный спуск

Для цитирования: Курденкова Е.О., Черепнина М.С., Чистякова А.С., Архипенко К.В. Влияние трансформаций на успешность состязательных атак для классификаторов изображений Clipped BagNet и ResNet. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 101-116. DOI: 10.15514/ISPRAS-2022-34(6)-7

Effect of transformations on the success of adversarial attacks for Clipped BagNet and ResNet image classifiers

[1]*E.O. Kurdenkova, ORCID: 0000-0001-5871-8179 <kurdenkova@ispras.ru>*
[2] *M.S. Cherepnina, ORCID: 0000-0003-1186-6718 <m.cherepnina@ispras.ru>*
[1,3] *A.S. Chistyakova, ORCID: 0000-0003-4896-4418 <a.chistyakova@ispras.ru>*
[1]*K.V. Arkhipenko, ORCID: 0000-0002-8699-889X <arkhipenko@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Technical University of Munich Arcisstraße 21, 80333 München, Germany*
[3] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*

Abstract. Our paper compares the accuracy of the vanilla ResNet-18 model with the accuracy of the Clipped BagNet-33 and BagNet-33 models with adversarial learning under different conditions. We performed experiments on images attacked by the adversarial sticker under conditions of image transformations. The adversarial sticker is a small region of the attacked image, inside which the pixel values can be changed indefinitely, and this can generate errors in the model prediction. The transformations of the attacked images in this paper simulate the distortions that appear in the physical world when a change in perspective, scale or lighting changes the image. Our experiments show that models from the BagNet family perform poorly on images in low quality. We also analyzed the effects of different types of transformations on the models' robustness to adversarial attacks and the tolerance of these attacks.

Keywords: adversarial attack; adversarial patch; BagNet architecture; adversarial training; projected gradient descent

For citation: Kurdenkova E.O., Cherepnina M.S., Chistyakova A.S., Arkhipenko K.V. Effect of transformations on the success of adversarial attacks for Clipped BagNet and ResNet image classifiers. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 101-116 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-7

1. Введение

Предыдущие работы [1, 2] показали, что несмотря на хорошую способность современных нейронных моделей решать задачи компьютерного зрения, эти модели остаются уязвимыми для атак злоумышленников. В данной статье рассматриваются атаки состязательной наклейкой, которая представляет из себя ограниченную область изображения, к которой может быть применено неограниченное возмущение пикселей. Пример атакованных изображений приведен на рис. 1.

Атака состязательной наклейкой находит свое применение в физическом мире. Злоумышленник может распечатать такую наклейку и поместить её в область видимости классификатора (например, во время детекции) и тем самым атаковать систему. Существуют атаки, которые могут генерировать универсальные состязательные наклейки, которые не зависят от сцены на изображении и даже от распознаваемого предмета [3]. Самым популярным примером применения состязательной наклейки является нанесения таковой на дорожный знак «стоп», после чего такой знак распознается классификатором как знак «ограничение скорости». Такая атака может привести к серьезным последствиям в эпоху тотальной автоматизации процессов. Таким образом, защита от подобного вида атак состязательной наклейкой является крайне важной задачей.

Существуют архитектуры моделей, которые лучше других справляются с атаками наклейкой. Например, в статье [4] авторами была предложена модель BagNet, которая более устойчива к атакам наклейкой в сравнении с другими моделями. Затем в статье [5] была предложена модификация этой модели – Clipped BagNet (CBN), которая улучшила устойчивость модели.

Авторы статьи [6] пошли еще дальше и предложили универсальные маски для моделей с небольшими размерами рецептивных полей для повышения устойчивости моделей к атакам наклейкой. Также в статье [6] предложен метод состязательного обучения модели Clipped BagNet с использованием одной из масок.

Рис. 1. Изображения, атакованные состязательной наклейкой PGD на примере модели CBN. В первом ряду изображения до применения трансформаций, второй ряд – гауссовский шум, третий ряд – затемнение, четвертый – поворот с масштабированием, пятый – комбинация всех трансформаций. Над изображениями указаны истинные метки и предсказанные на соответствующем этапе. Число итераций: 40, размер стороны наклейки 40 пикселей, что эквивалентно 3,2% от площади изображения

Fig. 1. Images attacked with a PGD adversarial patch using the CBN model as an example. In the first row of the image before applying the transformations, the second row is Gaussian noise, the third row is darkening, the fourth is rotation with scaling, the fifth is a combination of all transformations. Above the images, the true labels and those predicted at the corresponding stage are indicated. Number of iterations: 40, patch side size 40 pixels, equivalent to 3.2% of the image area

В перечисленных статьях атаки состязательной наклейкой рассматривались статичными, то есть без моделирования эффектов из физического мира. В физическом мире атаки наклейкой могут рассматриваться как непосредственное нанесение наклейки на предмет (например, на дорожный знак), уже после этого получается изображение, которое необходимо классифицировать. В таком случае изображение и наклейка вместе будут подвержены одинаковым изменениям, связанным с ракурсом, выбранным для фотографии, изменениям

освещения, а также шумам, возникающим в связи с техническими особенностями фотоаппарата. Чтобы получить более объективную оценку эффективности моделей при атаке, необходимо учитывать эти изменения изображения, поэтому в данном исследовании рассматриваются трансформации над состязательными изображениями.

Модели на основе архитектуры BagNet обладают маленьким размером рецептивного поля, что в совокупности с эффектами физического мира может привести к низкой точности предсказаний. Поэтому, одна из основных задач данной статьи – это исследовать влияние таких эффектов на устойчивость модели к атакам.

В рамках нашей статьи мы:

- провели и систематизировали более 200 вычислительно трудоемких экспериментов по состязательным атакам наклейкой;
- показали, что модели из семейства BagNet плохо справляются с изображениями в низком качестве;
- проанализировали, насколько сильно четыре разных типа трансформаций физического мира влияют на устойчивость моделей к состязательным атакам и переносимость этих атак.

2. Типы состязательных атак

2.1 Атака состязательной наклейкой

Атака состязательной наклейкой в данном исследовании основана на атаке проектируемого градиентного спуска (PGD – Projected Gradient Descent), которая является атакой белого ящика, то есть предполагается, что злоумышленник имеет доступ к модели, ее весам и градиентам [7]. Успеха в такой атаке часто можно добиться с помощью невидимых человеку изменений, что является несомненным плюсом для злоумышленника.

Идея PGD атаки заключается в решении задачи оптимизации с ограничениями. Атака пытается найти такие возмущения изображения, которые бы максимизировали потери модели на выходе (в случае целевой атаки – на конкретном выходе), при этом искомое возмущение обычно ограничено некоторой величиной ϵ. Ограничение задается с помощью нормы ℓp, которая определяется формулой (1). В данном исследовании рассматривается $\ell 0$-норма, которая измеряет, какое количество пикселей были изменены:

$$\left|\left|x\right|\right|_p = \left(\sum_i |x_i|^p\right)^{1/p}. \quad (1)$$

Если ставится задача изменить предсказание модели на любое неверное, то атака называется нецелевой. Формальная запись преобразований для нецелевой PGD атаки представлена формулой (2):

$$x'_{i+1} = \prod_{x+\epsilon}\left[x'_i + \alpha \cdot \mathrm{sign}\left(\nabla_x \mathcal{L}(x, y, \theta)\right)\right]. \quad (2)$$

Если же цель – обмануть модель, получив от модели метку заранее определенного класса, то атака называется целевой. Преобразования для целевой PGD атаки описываются формулой 3:

$$x'_{i+1} = \prod_{x+\epsilon}\left[x'_i - \alpha \cdot \mathrm{sign}\left(\nabla_x \mathcal{L}(x, \hat{y}, \theta)\right)\right]. \quad (3)$$

В этих формулах θ – параметры модели, x – изображение, y – истинная метка, $\hat{y}$ – целевая метка, $\mathcal{L}$ – функция потерь, i – номер итерации, $\Pi x + \epsilon$ – оператор проекции, который обрезает входные данные в позициях вокруг предопределенного диапазона возмущений, α – шаг, ϵ – максимально возможное возмущение, ℓp – норма.

Атака состязательной наклейкой отличается от обычной PGD атаки тем, что область возмущения на изображении ограничена, при этом значение возмущения ϵ не ограничено [3]. В данном эксперименте рассматриваются квадратные наклейки для атаки, реализованной в библиотеке Adversarial Robustness Toolbox, и для классической PGD-атаки, реализованной авторами в рамках данного исследования и описанной в статье [2].

В классической атаке PGD подбор наклейки, которая бы изменила метку изображения при предсказании, происходит следующим образом, аналогично методу из статьи [4]. Изображение разделяется на участки с помощью квадратной сетки, где длина ячейки сетки равна длине стороны квадратной наклейки. Далее наклейка подбирается отдельно на каждом участке сетки по порядку до тех пор, пока не найдется такое место и такая наклейка, при которой атака успешна. Если после перебора всех мест по сетке не удалось найти наклейку, которая бы меняла метку изображения, то атака считается неуспешной.

В рамках данного исследования реализация указанных методов атаки включала в себя расчёт градиента по всему изображению, а не только по области с состязательной наклейкой. Наклейку, полученную описанным образом, будем называть состязательной наклейкой PGD.

2.2 Состязательные атаки в физическом мире

Многие атаки генерируют состязательные изображения, основываясь на готовом классифицируемом изображении, добавляя к нему некоторые возмущения. Такие атаки обычно используются в случае классификации после получения фотографии, т.е. атака не изменяет реальные объекты. Но если распечатать состязательную наклейку и поместить ее на реальный предмет, то успешность атаки обычно снижается. Это происходит, поскольку добавленные возмущения для создания состязательного изображения изменятся из-за различных эффектов физического мира, несовершенства или особенностей сенсоров камеры, ракурса съёмки и т. д.

В статье [8] предлагается метод генерации состязательных примеров, которые были бы устойчивыми к трансформациям физического мира. Авторы статьи назвали предложенный метод Expectation over transformation (EOT), т.е. ожидание над трансформацией. EOT основывается на моделировании трансформаций физического мира в рамках процедуры генерации состязательной атаки. Задача, которая решается в данном методе, представлена формулой (4):

$$\arg\max_{x'} \mathbb{E}_{t\sim T}\left[\log P\left(y_t|t(x')\right)\right]$$
$$\text{при условии } \mathbb{E}_{t\sim T}\left[d\left(t(x'),t(x)\right)\right] < \varepsilon, \text{x} \in [0,1]^D. \quad (4)$$

В этой формуле обозначено за T – распределение возможных трансформаций, $t(\cdot)$ – функция преобразования из T, $d(\cdot,\cdot)$ – функция расстояния, x – исходное изображение, x' – состязательное изображение, D – размерность пространства изображений, yt – целевой класс, P – вероятность, а ϵ – максимальное возмущение.

Данный метод вместо оптимизации вероятности предсказания целевого класса на одном исходном изображении использует выбранное распределение трансформаций и оптимизирует математическое ожидание по этому распределению. Также вместо обычного расстояния между исходным и состязательным изображениями в данном методе рассматривается ожидаемое расстояние по всем трансформациям. Авторы статьи [8] рассматривают двумерные и трёхмерные трансформации, среди которых повороты, масштабирование, затемнение и гауссовский шум.

2.3 PGD-атака, реализованная в Adversarial Robustness Toolbox

В дополнение к классической PGD атаке в нашей статье исследуется поведение нейронных сетей ResNet и BagNet при PGD атаке наклейкой, реализованной в библиотеке Adversarial

Robustness Toolbox (ART) [9] и описанной в статье [3]. Данная атака является модификацией рассмотренной ранее PGD атаки, отличаясь от неё инициализацией состязательной наклейки и ее местоположением на изображении. В реализованной в ART атаке наклейка инициализируется однотонной картинкой – средним между максимальным и минимальным возможными значениями пикселя. Затем на каждой итерации наклейка помещается в случайное место на атакуемом изображении.

Указанный выше способ генерации состязательной наклейкой позволяет делать её более универсальной, не зависящей от освещения, угла камеры, места и т. д.

Наклейку, полученную в процессе этой атаки, будем называть состязательной наклейкой ART.

3. Защита от состязательных атак на основе изменения архитектуры сети

В этом разделе мы рассмотрим методы защиты от состязательных наклеек с помощью изменения архитектуры нейронной сети.

3.1 Метод защиты на основе модели BagNet

Архитектура BagNet [4] основана на идее модели Bag-Of-Local-Features (мешок слов), которая применяется для задач, связанных с обработкой текстов. Её идея состоит в представлении текста в виде словаря из слов и количеством их вхождений в текст. Так, текст можно представить в виде числового вектора.

Для изображений в качестве визуальных слов используются отдельные кусочки изображения. По каждому из кусочков в отдельности делается предсказание. Итоговые выходные значения модели определяются голосованием по всем кускам изображения.

Таким образом, первым шагом в модели BagNet является получение 2048-мерного представления для каждого фрагмента исходного изображения размером $q \times q$ пикселей. Для этого используется несколько блоков ResNet и линейный классификатор. Так получается значение выхода на каждом классе для каждого фрагмента изображения. Затем значение выходов на всех фрагментах усредняются для получения итогового значения для каждого класса.

BagNet отличается от классического ResNet только заменой большинства свёрток 3×3 на 1×1, что позволяет ограничить размер рецептивного поля самой верхней свёртки до размера $q \times q$.

В экспериментах к данной статье в качестве q для модели BagNet было выбрано q=33.

3.2 Clipped BagNet (CBN)

Слабой стороной BagNet является то, что большое изменение одного выходного значения для хотя бы одного фрагмента изображения приведёт к увеличению среднего значения, в результате чего модель неустойчива к атакам.

Для решения этой проблемы можно использовать модификацию модели – Clipped BagNet (CBN). Данная модификация реализуется с помощью ограничения выходов модели на каждом фрагменте изображения. В статье [5] показано, что в качестве ограничения можно выбрать функцию $f(x)$=tanh(ax+b) с параметрами a=0.05 и b=−1.

На рис. 2 изображена тепловая карта выходов модели BagNet (до ограничения выходных значений) и CBN до трансформаций изображений для случая, когда CBN предсказывает метку верно, а BagNet ошибается. По этому рисунку можно понять, как изображение видит модель BagNet, а как CBN.

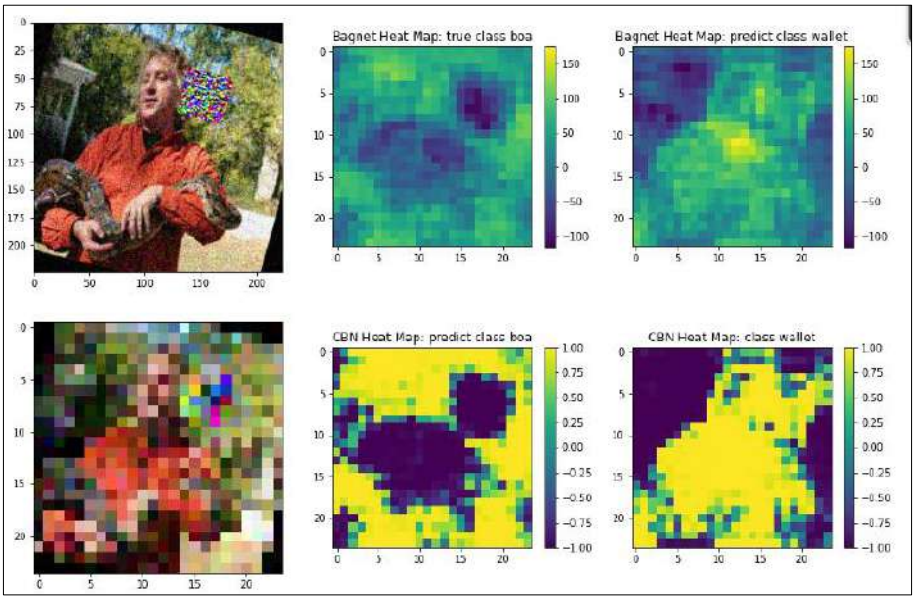

Рис. 2. Тепловая карта выходов моделей для истинного и предсказанного классов на каждом участке рецептивного поля для состязательного изображения с трансформациями. Случай, когда CBN предсказывает верно, а BagNet ошибается. Для модели CBN в нижнем ряду справа изображена тепловая карта для класса, который предсказала модель BagNet

Fig. 2. Heat map of model outputs for true and predicted classes at each region of the receptive field for an adversarial image with transformations. The case when CBN predicts correctly, but BagNet is wrong. For the CBN model, the bottom row on the right shows the heat map for the class predicted by the BagNet model

3.3 Clipped BagNet с состязательным обучением (ADV CBN)

В статье [6] авторами было предложено улучшить устойчивость модели к атакам с помощью PatchGuard масок. Идея одной из масок (маска «m») состоит в том, что необходимо до агрегирования выходов модели пропускать свидетельства для каждого класса через окно определенного размера и маскировать (убирать) максимальное значение, после чего уже агрегировать выходы и делать предсказание. Размер окна зависит от структуры модели и от максимально возможного размера атакующей наклейки. Вторая маска (маска «cbn») заключается в добавлении ограничивающей свидетельства класса функцией перед финальной агрегацией. В качестве ограничивающей функции была выбрана аналогичная [5] функция $f(x)=\tanh(ax+b)$ с параметрами $a=0.05$ и $b=-1$. Авторы статьи [6] предложили использовать такие маски для состязательного обучения модели CBN. Обученная таким способом модель в данной статье обозначается как ADV CBN (Adversarial training for Clipped BagNet).

4. Влияние трансформаций на успешность состязательных атак

В экспериментах, описываемых в данной статье, рассматривались изображения размера 224×224 пикселя, в качестве максимального возмущения для атак было выбрано $\epsilon=1$. Значения компонент пикселей оригинальных изображений заданы числами от 0 до 1.

В данном исследовании проводилась классическая PGD атака наклейкой на модели ResNet, CBN и ADV CBN, а также проводились атаки, реализованные в библиотеке Adversarial Robustness Toolbox [9]. В результате этих атак были получены состязательные изображения, над которыми проводились некоторые трансформации. Эти трансформированные состязательные изображения снова подавались на вход модели. Таким образом проверялась

устойчивость моделей ResNet и CBN на атаках с трансформациями состязательных изображений.

Параметры состязательной атаки ART настраивались подобно параметрам атаки PGD. В эксперименте рассматривались квадратные наклейки без дополнительных поворотов и каких-либо трансформаций при генерации атаки (таким образом, стороны наклейки всегда параллельны сторонам изображения).

В качестве дополнительного результата были получены точности модели на состязательных изображениях для других моделей. Трансформации изображений к ним также применялись.

Помимо обычных моделей, в экспериментах рассматривались модели с масками из статьи [6]. Для модели ResNet рассматривались обе маски, а для моделей CBN и ADV CBN только маска «m», поскольку применение маски «cbn» заключается в ограничении свидетельств модели гиперболическим тангенсом, которое и так содержится в этих моделях.

4.1 Модели и данные

В экспериментах использовались модели ResNet-18 (ResNet), Clipped BagNet-33 (CBN) и Adversarial Clipped BagNet-33 (ADV CBN). В качестве данных была выбрана часть датасета ImageNet [10], состоящая из 10 классов по 1000 изображений в каждом. Тестовая выборка составляла 33% от используемых данных, остальная часть данных использовалась для обучения моделей.

CBN и ADV CBN обучались с помощью SGD оптимизатора с параметром learning rate =0.001, а ResNet с помощью SGD оптимизатора и с параметром learning rate =0.01.

Код к данному исследованию, а также ссылки на используемые данные и модели представлены в репозитории https://github.com/kekaterina/transformation_adversarial_attack.

4.2 Трансформация изображений

Трансформации изображений помогают проверить надежность атаки наклейкой в физическом мире, когда материальная наклейка нанесена на реальный предмет. В экспериментах использовались 3 типа трансформаций: затемнение, гауссовский шум и поворот с масштабированием, а также комбинация из всех этих трансформаций одновременно.

Табл. 1. Значения параметров трансформаций
Table 1. Values of transformation parameters

Трансформация	*Минимум*	*Максимум*
Масштаб (№1)	0.9	1.4
Масштаб (№2)	0.8	1.0
Поворот (№1)	$-22.5°$	$22.5°$
Поворот (№2)	$-10.0°$	$10.0°$
Осветвление и затемнение	-0.05	0.05
Трансформация	*Среднее значение*	*Отклонение*
Гауссовский шум	0.0	0.1

Параметры трансформаций (для масштаба и поворота варианты №1) взяты из статьи [8] и находятся в табл. 1. В указанных границах строились равномерные распределения параметров, из которых для каждого изображения случайным образом подбирались конкретные параметры для трансформаций.

Для экспериментов с состязательной наклейкой ART использовались масштаб №1 и поворот №1, а для экспериментов с состязательной наклейкой PGD использовались масштаб №2 и поворот №2. Это обусловлено тем, что в состязательных наклейках ART местоположение

наклейки обычно находится в середине изображения, поэтому трансформировать такое изображение без большого изменения площади наклейки можно сильнее. Для состязательной наклейки PGD параметры масштабирования и поворота уменьшены, чтобы не обрезать большую часть изображения с состязательной наклейкой. В последующих таблицах трансформации типа осветления/затемнения будут обозначаться как затемнение.

5. Результаты экспериментов

В этом разделе мы сравним устойчивость моделей CBN, ADV CBN и ResNet к состязательным атакам в условиях трансформаций и исследуем переносимость атак.

5.1 Трансформации

Эксперименты над трансформированными состязательными изображениями проводились с помощью двух типов состязательных атак квадратной наклейкой:

- наклейки PGD разных размеров: 60×60, 50×50, 40×40, 32×32, 20×20 пикселей;
- наклейки ART при разных значениях параметра $patchScale$≈0.446,0.268,0.223,0.179,0.142,0.079; параметр $patchScale$ рассчитывался таким образом, чтобы наклейка ART соответствовала доли равной доли наклейки PGD от площади исходного изображения.

На рис. 1 представлены изображения атакованные состязательными PGD наклейками, до и после трансформаций.

Точность моделей рассчитывалась на изображениях из тестового набора изображений – 3273 изображения из 10 классов.

Табл. 2. Атака состязательной наклейкой PGD. Точность моделей на чистых и атакованных изображениях с трансформациями и без них. Размер наклейки указан в пикселях. Красными и зелеными цифрами в скобках обозначена разница в точности CBN и ADV CBN по сравнению с аналогичным экспериментом на ResNet. ***Жирным*** *шрифтом выделен лучший результат в каждом столбце. Результаты для модели ADV CBN получены на 1000 тестовых изображениях, результаты остальных моделей представлены для всего тестового набора*

Table 2. Attack with the adversarial PGD patch. Accuracy of models on clean and attacked images with and without transformations. The patch size is in pixels. The red and green numbers in parentheses indicate the difference in CBN and ADV CBN accuracy compared to the same experiment on ResNet. ***Bold*** *indicates the best result in each column. The results for the ADV CBN model were obtained on 1000 test images, the results of other models are presented for the entire test set*

Модель	*Трансформация*	*Исходное изображение + трансформации*	*Наклейка 20×20*	*Наклейка 32×32*	*Наклейка 40×40*	*Наклейка 50×50*	*Наклейка 60×60*
ResNet-18	Без трансформации	**0.790**	0.128	0.027	0.013	0.009	0.007
	Поворот	0.696	**0.620**	**0.615**	**0.560**	**0.500**	**0.428**
	Шум Гаусса	0.715	0.317	0.324	0.060	0.022	0.010
	Затемнение	0.787	0.218	0.236	0.025	0.011	0.073
	Комбинация	0.626	0.563	0.569	0.512	0.475	0.406
	Без трансформации	**0.675** (−0.115)	0.601 (+0.473)	0.555 (+0.528)	0.520 (+0.507)	0.456 (+0.447)	0.398 (+0.391)

CBN-33	Поворот	0.635 (−0.061)	0.628 (+0.008)	0.609 (−0.006)	**0.609** (+0.049)	**0.588** (+0.088)	**0.567** (+0.139)
	Шум Гаусса	0.262 (−0.453)	0.255 (−0.062)	0.246 (−0.078)	0.241 (+0.181)	0.234 (+0.014)	0.219 (+0.209)
	Затемнение	0.668 (−0.119)	**0.643** (+0.425)	**0.616** (+0.380)	0.565 (+0.540)	0.508 (+0.497)	0.452 (+0.379)
	Комбинация	0.263 (−0.363)	0.262 (−0.301)	0.259 (−0.310)	0.257 (−0.255)	0.250 (−0.225)	0.247 (−0.159)
ADV CBN-33	Без трансформации	**0.696** (−0.094)	0.639 (+0.511)	0.612 (+0.585)			
	Поворот	0.642 (−0.054)	0.616 (−0.004)	**0.613** (−0.002)			
	Шум Гаусса	0.373 (−0.342)	0.377 (−0.062)	0.346 (+0.060)			
	Затемнение	0.690 (−0.097)	**0.642** (+0.424)	0.610 (+0.374)			
	Комбинация	0.390 (−0.236)	0.385 (−0.187)	0.378 (−0.191)			

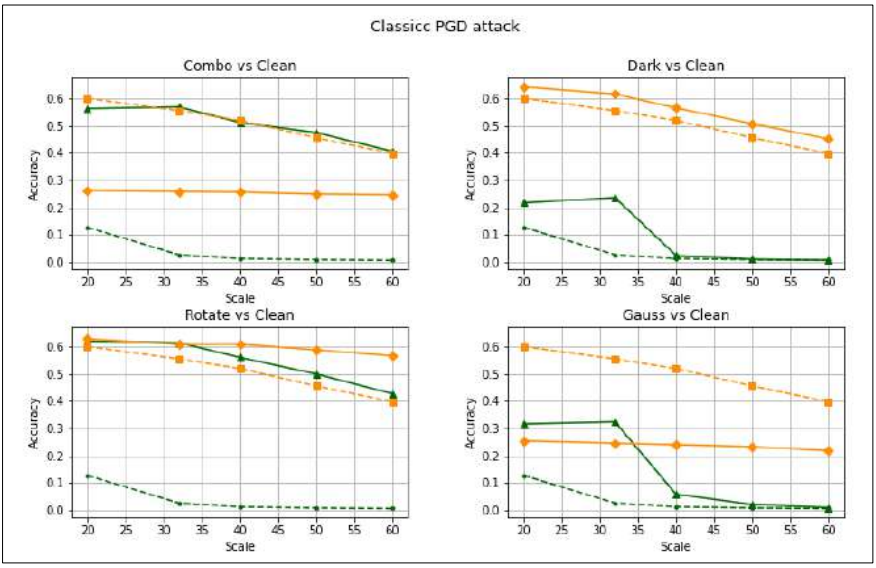

Рис. 3. Изменение точности моделей на атакованных классической PGD-атакой изображениях с трансформациями и без. По горизонтальной оси указан размер состязательной наклейки в пикселях
Fig. 3. Changing the accuracy of models on images attacked by the classical PGD attack with and without transformations. The horizontal axis shows the size of the contest sticker in pixels

Результаты экспериментов с трансформациями и состязательными наклейками PGD представлены в табл. 2 и на рис. 3. Данные результаты показывают, что:

- модели CBN и ADV CBN ожидаемо лучше защищены от атак без трансформаций, чем модель ResNet. Это совпадает с выводами в [4, 5, 11];
- из всех трансформаций без состязательной атаки гауссовский шум больше всех снижает точность BagNet: CBN – на 45%, ADV CBN – на 34%, тогда как точность ResNet упала лишь на 7,5%;
- при всех размерах состязательной наклейки самые низкие результаты CBN и ADV CBN показывают на изображениях с гауссовским шумом;

- при комбинированной трансформации CBN и ADV CBN оказались хуже ResNet во всех проведенных экспериментах: как с состязательными наклейками, так и без;
- поворот и затемнение/осветление состязательных изображений значительно повышает точность моделей ResNet и CBN;
- модель ADV CBN имеет точность выше, чем у обычного CBN, на изображениях без трансформаций. Но в половине экспериментов с трансформированными состязательными изображениями ADV CBN проигрывает CBN;
- гауссовский шум чуть меньше влияет на ADV CBN, чем на CBN – точность на зашумленных состязательных изображениях у ADV CBN выше.

Таким образом, модели CBN и ADV CBN показывают лучшую точность, чем ResNet на атаках наклейками. Однако при атаках с комбинацией трансформаций ResNet оказывается лучше CBN. Добавление состязательного обучения к модели CBN (для получения ADV CBN) дает значимое преимущество только при работе с состязательными изображениями, не подвергшимся трансформациям.

Заметим, что для модели ADV CBN-33 представлены результаты не для всех рассматриваемых размеров наклейки, поскольку исследование этой модели не было основной целью данной работы и проводилось по остаточному принципу, насколько хватило ресурсов. Также, результаты проведенных над ADV CBN-33 экспериментов показывают, что эта модель во многом похожа на модель CBN-33.

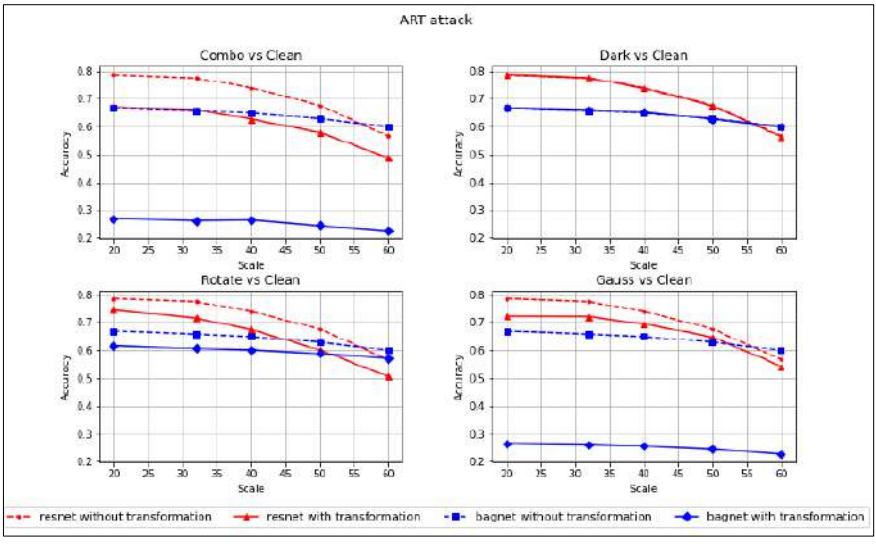

Рис. 4. Изменение точности моделей на атакованных состязательной наклейкой ART изображениях с трансформациями и без. По горизонтальной оси указан размер состязательной наклейки в пикселях
Fig. 4. Changing the accuracy of models on images attacked by an ART adversarial sticker with and without transformations. The horizontal axis shows the size of the contest sticker in pixels

Результаты экспериментов для состязательных наклеек ART показаны в табл. 3 и на рис. 4. При выбранных нами гиперпараметрах атака ART оказалась слабее PGD, т.к. при одинаковых размерах наклейки PGD снижала точность моделей сильнее. Результаты в табл. 3 показывают, что:

- только при размерах наклейки ART 60×60 пикселей CBN показывает результат выше, чем ResNet, в 3 из 5 экспериментах; при более маленьких наклейках CBN всегда остается менее точным;
- лучшие результаты обе модели показывают на изображениях без трансформаций и на изображениях с затемнением;

- среди всех трансформаций гауссовский шум больше всего снижает точность CBN; аналогичный результат получен и в экспериментах с PGD наклейкой.

Табл. 3. Атака состязательной наклейкой ART. Точность моделей на чистых и атакованных изображениях с трансформациями и без них. Размер наклейки указан в пикселях. Красными и зелеными цифрами в скобках обозначена разница в точности CBN по сравнению с аналогичным экспериментом на ResNet. ***Жирным*** *шрифтом выделен лучший результат в каждом столбце*
Table 3. Attack with the adversarial patch ART. Accuracy of models on clean and attacked images with and without transformations. The patch size is in pixels. The red and green numbers in brackets indicate the difference in CBN accuracy compared to the same experiment on ResNet. Bold indicates the best result in each column

Модель	*Трансформация*	*Исходное изображение + трансформации*	*Наклейка 20×20*	*Наклейка 32×32*	*Наклейка 40×40*	*Наклейка 50×50*	*Наклейка 60×60*
ResNet-18	Без трансформации	**0.790**	0.786	0.775	**0.740**	**0.677**	**0.566**
	Поворот	0.735	0.745	0.716	0.675	0.602	0.506
	Шум Гаусса	0.714	0.722	0.722	0.695	0.647	0.543
	Затемнение	0.787	**0.787**	**0.778**	0.738	0.675	0.564
	Комбинация	0.665	0.672	0.680	0.628	0.580	0.488
CBN-33	Без трансформации	**0.675** (−0.115)	0.669 (−0.117)	0.658 (−0.117)	0.650 (−0.090)	**0.629** (−0.048)	**0.601** (+0.035)
	Поворот	0.623 (−0.112)	0.617 (−0.128)	0.607 (−0.109)	0.601 (−0.074)	0.588 (−0.014)	0.570 (+0.064)
	Шум Гаусса	0.261 (−0.453)	0.266 (−0.456)	0.262 (−0.460)	0.255 (−0.440)	0.245 (−0.402)	0.228 (−0.315)
	Затемнение	0.670 (−0.117)	**0.670** (−0.117)	**0.660** (−0.118)	**0.652** (−0.086)	0.628 (−0.047)	**0.601** (+0.037)
	Комбинация	0.266 (−0.399)	0.269 (−0.403)	0.262 (−0.418)	0.264 (−0.364)	0.244 (−0.336)	0.225 (−0.263)

Таким образом, при слабых атаках, несильно снижающих точность ResNet, модель с архитектурой CBN остается менее точной.

Точность моделей на оригинальных и состязательных изображениях при использовании масок, описанных в статье [6], приводится в табл. 4. По представленным данным видно, что:

- для модели ResNet маски в среднем увеличивают устойчивость модели к атаке, но маска «cbn» в данном случае показывает себя лучше маски «m». ResNet в совокупности с маской «m» имеет более низкую точность на оригинальных изображениях в сравнении с маской «cbn»; зато для моделей CBN и ADV CBN маска «m» улучшает устойчивость к атаке и к трансформациям;
- применение поворота к состязательным изображениям на модели ResNet с применением маски и на обеих версиях CBN с применением маски действует на точность по-разному; для ResNet с маской это повышает точность модели, а для CBN и ADV CBN с маской наоборот понижает.

Заметим, что многих случаях PGD наклейка располагается в верхнем левом углу изображения (рис. 1). Это связано с тем, что алгоритм и гиперпараметры PGD-атаки зачастую позволяют успешно атаковать модель с первой итерации и, следственно, не перебирать другие локации для наклейки. Мы предполагаем, что любое другое положение наклейки

(например, в центре изображений) не изменило бы описанных нами закономерностей. Однако доказательство этой гипотезы мы оставляем для дальнейших исследований.

Табл. 4. Точность моделей на чистых и атакованных изображениях с трансформациями и без них для моделей с масками PatchGuard. Размер наклейки указан в пикселях. Синим цветом отмечена точность на «родных» состязательных изображениях для модели. Результаты для 1000 тестовых изображений. Для состязательных изображений размер наклейки равен 32×32

Table 4. Accuracy of models on clean and attacked images with and without transformations for models with PatchGuard masks. The patch size is in pixels. The blue color marks the accuracy on the "native" adversarial images for the model. Results for 1000 test images. For competitive images, the sticker size is 32×32

Модель	*Трансформация*	*Исходное изображение + трансформации*	*Состязательные изображения для ResNet*	*Состязательные изображения для CBN*	*Состязательные изображения для ADV CBN*
ResNet-18 + маска «cbn»	Без трансформации	0.792	0.567	0.785	0.784
	Поворот	0.730	**0.667**	0.724	0.709
	Шум Гаусса	**0.793**	0.579	**0.788**	**0.788**
	Затемнение	0.731	0.482	0.730	0.732
	Комбинация	0.653	0.610	0.654	0.655
ResNet-18 + маска «m»	Без трансформации	0.382	**0.637**	**0.638**	**0.662**
	Поворот	0.516	**0.427**	0.490	0.478
	Шум Гаусса	0.661	0.377	0.627	0.632
	Затемнение	0.585	0.374	0.584	0.566
	Комбинация	0.289	0.269	0.278	0.278
CBN-33 + маска «m»	Без трансформации	**0.688**	**0.681**	**0.655**	**0.679**
	Поворот	0.652	0.652	0.628	0.642
	Шум Гаусса	0.667	0.667	0.638	0.669
	Затемнение	0.269	0.273	0.271	0.274
	Комбинация	0.278	0.289	0.269	0.278
ADV CBN-33 + маска «m»	Без трансформации	**0.694**	**0.696**	**0.677**	**0.633**
	Поворот	0.632	0.634	0.627	0.604
	Шум Гаусса	0.683	0.690	0.673	0.617
	Затемнение	0.382	0.388	0.375	0.365
	Комбинация	0.394	0.383	0.386	0.385

5.2 Переносимость атак

В табл. 5 представлены точности моделей на оригинальных изображениях без состязательной наклейки и на «чужих» состязательных изображениях с наклейкой размера 32×32 пикселя. Зеленым цветом обозначена точность моделей на оригинальных изображениях, а черным – на состязательных. В этом эксперименте рассматривалась классическая PGD атака. По результатам видно, что:

- атака переносится слабо; на собственных состязательных изображениях точность моделей (табл. 2) гораздо ниже, чем на чужих;
- на чужих состязательных изображениях без трансформаций точность снижается незначительно, зато при повороте этих изображений разница с точностью на оригинальных изображениях без трансформаций становится существеннее.

Табл. 5. Точность моделей на состязательных изображениях других моделей. Размер наклейки 32×32 в пикселях. По горизонтали отмечены названия моделей, для которых генерировались состязательные изображениях. На диагональных блоках (где название модели совпадает с названием модели, для которой строились состязательные изображения) зелёным цветом выделена точность на оригинальных изображениях без наклейки. Результаты для 1000 тестовых изображений
Tab. 5. Accuracy of models on competitive images of other models. Patch size is 32×32 in pixels. The names of the models for which competitive images were generated are marked horizontally. On the diagonal blocks (where the name of the model matches the name of the model for which the competitive images were built), the accuracy on the original images without a patch is highlighted in green. Results for 1000 test images

Модель	*Трансформация*	*Состязательные изображения для ResNet*	*Состязательные изображения для CBN*	*Состязательные изображения для ADV CBN*
ResNet-18	Без трансформации	**0.794**	**0.785**	**0.789**
	Поворот	0.709	0.708	0.698
	Шум Гаусса	0.791	0.784	0.786
	Затемнение	0.727	0.722	0.725
	Комбинация	0.628	0.632	0.630
CBN-33	Без трансформации	**0.694**	**0.694**	**0.684**
	Поворот	0.643	0.642	0.633
	Шум Гаусса	0.689	0.678	0.669
	Затемнение	0.276	0.269	0.272
	Комбинация	0.273	0.263	0.272
ADV CBN-33	Без трансформации	**0.693**	0.679	**0.696**
	Поворот	0.647	0.635	0.650
	Шум Гаусса	0.687	**0.685**	0.692
	Затемнение	0.378	0.377	0.374
	Комбинация	0.369	0.373	0.380

Аналогичные результаты переносимости атак получены для моделей с масками, описанными в статье [6], представлены в табл. 4. Результаты в данном случае аналогичны эксперименту для моделей без масок.

По результатам можно сделать вывод, что переносимость состязательных изображений одной модели на другую для данных видов атак и моделей есть, но она очень слабая.

6. Заключение

По результатам экспериментов можно сделать следующие выводы:

- добавление гауссовского шума и комбинации из трансформаций особенно сильно понижают предсказательную точность моделей CBN и ADV CBN даже без состязательных наклеек; таким образом, модель CBN плохо справляется с изображениями низкого качества;
- поворот и затемнение состязательных изображений значительно снижает успешность атак; несмотря на то, что рассматриваемое затемнение/осветление не видно для человеческого глаза, при некоторых размерах состязательных наклеек (например, 20×20, 32×32 при обоих методах генерации наклеек) лучше всего смогли нивелировать атаку;
- переносимость состязательных изображений одной модели на другую для данных видов атак и моделей есть, но она очень слабая.

Таким образом, архитектура CBN и ADV CBN может защитить от состязательных атак только при двух условиях:

- качество изображения хорошее, т.е. не подвержено таким трансформациям как зашумление, поворот, затемнение;
- атака значительно снижает точность незащищенной модели.

В случае же работы с изображениями плохого качества или при слабых атаках лучше предпочесть классическую модель классификации изображений или другие методы защиты от состязательных атак.

Список литературы / References

[1] Goodfellow I.J., Shlens J., Szegedy C. Explaining and Harnessing Adversarial Examples. ArXiv 1412.6572, 2014, 11 p.
[2] Madry A., Makelov A. et al. Towards Deep Learning Models Resistant to Adversarial Attacks. ArXiv 1706.06083, 2017, 28 p.
[3] Brown T.B., Mané D. et al. Adversarial Patch. ArXiv 1712.09665, 2017, 6 p.
[4] Brendel W., Bethge M. Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet. ArXiv 1904.00760, 2019, 13 p.
[5] Zhang Z., Yuan B. et al. Clipped BagNet: Defending Against Sticker Attacks with Clipped Bag-of-features. In Proc. of the 2020 IEEE Security and Privacy Workshops (SPW), 2020, pp. 55-61.
[6] Xiang C., Bhagoji A.N. et al. PatchGuard: Provable Defense against Adversarial Patches Using Masks on Small Receptive Fields. ArXiv 2005.10884, 2020, 23 p.
[7] Dong Y., Liao F. et al. Boosting Adversarial Attacks with Momentum. In Proc. of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 9185-9193.
[8] Athalye A., Engstrom L. et al. Synthesizing Robust Adversarial Examples. ArXiv 1707.07397, 2017, 19 p.
[9] Nicolae M., Sinn M. et al. Adversarial Robustness Toolbox v1.0.0. ArXiv, abs/1807.01069, 2018, 34 p.
[10] Russakovsky O., Deng J. et al. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, vol. 115, issue 3, 2015, pp. 211-252.
[11] Uesato J., O'Donoghue B. et al. Adversarial Risk and the Dangers of Evaluating Against Weak Attacks. ArXiv, abs/1802.05666, 2018, 13 p.

Информация об авторах / Information about authors

Екатерина Олеговна КУРДЕНКОВА – стажер-исследователь Центра доверенного искусственного интеллекта ИСП РАН (Научные интересы: исследование и разработка нейросетевых архитектур, устойчивых к состязательным атакам.

Ekaterina Olegovna KURDENKOVA – Graduate Research Trainee at ISP RAS Research Center for Trusted Artificial Intelligence. Research interests: research and development of neural network architectures aimed at robustness to adversarial attacks.

Мария Сергеевна ЧЕРЕПНИНА – студентка магистратуры Мюнхенского технического университета. Научные интересы: объяснимый искусственный интеллект, бизнес-информатика.

Maria Sergeevna CHEREPNINA – Master’s Student at Technical University of Munich. Research interests: explainable artificial intelligence, business informatics.

Анна Сергеевна ЧИСТЯКОВА – студентка бакалавратуры ф-та ВМК МГУ, лаборант Центра доверенного искусственного интеллекта ИСП РАН. Научные интересы: методы регуляризации моделей глубокого обучения, обеспечивающие устойчивость к состязательным атакам.

Anna Sergeevna CHISTYAKOVA – Bachelor’s Student of the faculty of the CMC of Moscow State University, Assistant at ISP RAS Research Center for Trusted Artificial Intelligence. Research interests: regularization methods for deep learning models providing robustness to adversarial attacks.

Константин Владимирович АРХИПЕНКО – младший научный сотрудник Центра доверенного искусственного интеллекта ИСП РАН. Научные интересы: защита моделей машинного обучения от атак, объяснимый искусственный интеллект.

Konstantin Vladimirovich ARKHIPENKO – Junior Research Fellow at ISP RAS Research Center for Trusted Artificial Intelligence. Research interests: defending machine learning models against attacks, explainable artificial intelligence.

DOI: 10.15514/ISPRAS-2022-34(6)-8

Исследование возможности применения нейронных сетей для восстановления изображения лица в системах распознавания

Е.И. Маркин, ORCID: 0000-0003-4848-8757 <evgeniymarkin1@gmail.com>
В.В. Зупарова, ORCID: 0000-0002-7903-083X <zuparova_vv@mail.ru>
А.И. Мартышкин, ORCID: 0000-0002-3358-4394 <mai@penzgtu.ru>
Пензенский государственный технологический университет,
440039, Россия, г. Пенза, проезд Байдукова/ул. Гагарина, д. 1а/11

Аннотация. Идентификация человека на цифровом изображении с помощью компьютерного зрения является важнейшим аспектом этой области. Наличие внешних объектов, таких как медицинские маски, которые закрывают часть лица, может резко снизить точность распознавания и увеличить ошибки от 5% до 50% в зависимости от алгоритма. В данной статье исследуется использование нейронных сетей, в частности генеративной состязательной сети (*GAN*), для решения задачи восстановления изображения лица, закрытого медицинской маской, для повышения точности распознавания лица.

Ключевые слова: компьютерное зрение; нейронные сети, генеративно-состязательные сети, идентификации лиц людей, реконструирование цифрового изображения.

Для цитирования: Маркин Е.И., Зупарова В.В., Мартышкин А.И. Исследование возможности применения нейронных сетей для восстановления изображения лица в системах распознавания. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 117-126. DOI: 10.15514/ISPRAS-2022-34(6)-8

Exploring the application of neural networks for facial image reconstruction in recognition systems

E.I. Markin, ORCID: 0000-0003-4848-8757 <evgeniymarkin1@gmail.com>
V.V. Zuparova, ORCID: 0000-0002-7903-083X <zuparova_vv@mail.ru>
A.I. Martyshkin, ORCID: 0000-0002-3358-4394 < mai@penzgtu.ru>
Penza State Technological University,
1a/11, Baidukova Passage/Gagarin st., Penza, 440039, Russia.

Abstract. Identifying a person in a digital image using computer vision is a crucial aspect of this field. The presence of external objects, such as medical masks that cover part of the face, can drastically reduce recognition accuracy and increase errors from 5% to 50%, depending on the algorithm. This paper investigates the use of neural networks, in particular the generative adversarial network (GAN), to solve the problem of reconstructing an image of a face covered by a medical mask to improve face recognition accuracy.

Keywords: computer vision; neural networks, generative adversarial networks, human face identification, digital image reconstruction.

For citation: Markin E.I., Zuparova V.V., Martyshkin A.I. Exploring the application of neural networks for facial image reconstruction in recognition systems. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 117-126 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-8

1. Введение

Использование передовых технологий и искусственного интеллекта произвело революцию во многих отраслях промышленности, повысив эффективность, точность и производительность. Компьютерное зрение – одна из таких областей, в которой в последние годы наблюдается значительный рост, с разработкой передовых алгоритмов и моделей, позволяющих анализировать и интерпретировать изображения и видео. Идентификация человеческих лиц по фотографическим изображениям - важнейший аспект компьютерного зрения, который находит применение в системах безопасности, контроля доступа, проверки данных и пользователей устройств [1].

Однако наличие посторонних объектов, загораживающих или скрывающих части лица, может сильно повлиять на точность распознавания и повысить коэффициент ошибок на значительную величину [2]. Широкое использование средств индивидуальной защиты во время пандемии коронавируса *2019-nCoV* привело к повсеместным сбоям в системах биометрической идентификации с масками на лице, что подчеркивает необходимость инновационных решений для решения проблемы идентификации лиц со скрытыми участками [3, 4].

В данной работе предлагается нейросетевой подход с использованием глубоких сверточных генеративно-состязательных сетей (*DCGAN*) [5] для восстановления скрытых частей лиц и улучшения качества и детализации изображений для повышения точности распознавания. Использование *DCGAN* для восстановления рисунка лица может значительно повысить точность идентификации личности и снизить коэффициент ошибок, обеспечивая надежное решение проблемы идентификации лиц со скрытыми участками лица.

2. Описание метода решения задачи

По мере развития технологий и разработки новых инноваций, область генерации данных и компьютерного зрения в последние годы претерпела значительные изменения. Одним из наиболее значительных прорывов в этой области стала разработка генеративных состязательных сетей (*GAN*) (рис. 1), которые представляют собой тип нейронных сетей, доказавших свою высокую эффективность в создании фотореалистичных изображений [6].

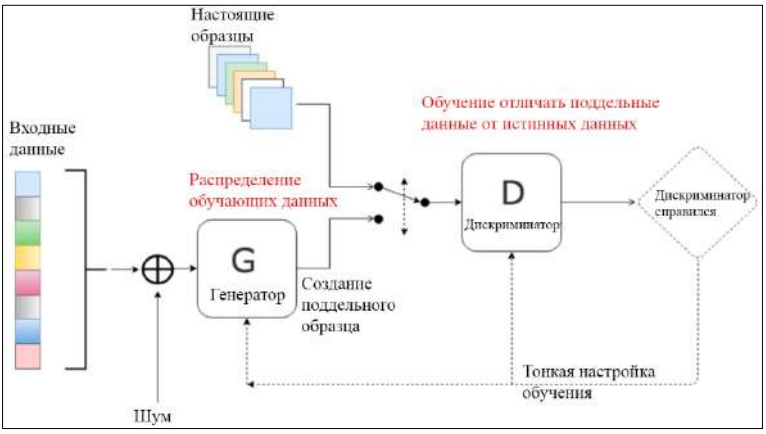

Рис. 1. Архитектура генеративной состязательной сети
Fig. 1. The architecture of the generative adversarial network

Основная концепция *GAN* основана на теории игр и включает две модели, генератор и критик, которые конкурируют друг с другом, чтобы сделать друг друга сильнее [7-9]. Генератор выдает синтетические образцы, которые должны выглядеть как можно ближе к реальным данным, а критик, также известный как дискриминатор, призван определить, является ли образец настоящим или поддельным. Это соревнование между генератором и

критиком приводит к тому, что обе модели со временем совершенствуются и становятся более эффективными в своих задачах.

Дискриминатор (D) должен с высокой точностью отличать реальные данные от поддельных, в то время как генератор (G) стремится производить образцы, которые D трудно отличить от реальных данных. Чтобы сбалансировать эти две цели, между D и G разыгрывается минимаксная игра, в которой функция потерь оптимизируется следующим образом:

$$\min_G \max_D L(D,G) = E_{x\sim p_r(x)}[\log D(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right] = E_{x\sim p_r(x)}[\log D(x)] + E_{x\sim p_g(x)}[\log(1 - D(x)]. \quad (1)$$

$E_{x\sim p_r(x)}[\log D(x)]$ не влияет на G во время обновления градиентного спуска.

Чтобы определить наилучшее значение дискриминатора, нам нужно максимизировать функцию потерь:

$$L(D,G) = \int_x \big(p_r(x)\log\big(D(x)\big) + p_g(x)\log\big(1 - D(x)\big)\big)dx. \quad (2)$$

Оптимальное значение *D(x)* максимизирует *L(G, D)*, его можно представить как:

$$\tilde{x} = D(x), A = p_r(x), B = p_g(x), \quad (3)$$

и оно может быть рассчитана как:

$$f(\tilde{x}) = A\log\tilde{x} + B\log(1-\tilde{x}), \quad (4)$$

$$\frac{df(\tilde{x})}{d\tilde{x}} = A\frac{1}{\ln 10}\frac{1}{\tilde{x}} - B\frac{1}{\ln 10}\frac{1}{1-\tilde{x}} = \frac{1}{\ln 10}\left(A\frac{1}{\tilde{x}} - B\frac{1}{1-\tilde{x}}\right) = \frac{1}{\ln 10}\frac{A-(A+B)\tilde{x}}{\tilde{x}(1-\tilde{x})}.$$

Таким образом, установив $\frac{df(\tilde{x})}{d\tilde{x}} = 0$, мы получаем наилучшее значение дискриминатора:

$$D^*(x) = \tilde{x}^* = \frac{A}{A+B} = \frac{p_r(x)}{p_r(x) + p_g(x)} \in [0,1]. \quad (5)$$

Когда генератор оптимально обучен, вероятности сгенерированных данных (p_g) и реальных данных (p_r) сходятся, и дискриминатор (D^*) должен давать значение ½. Это приводит к функции потерь:

$$L(G,D) = \int_x \big(p_r(x)\log\big(D^*(x)\big) + p_g(x)\log\big(1 - D^*(x)\big)\big)dx = \log\frac{1}{2}\int_x p_r(x)dx + \log\frac{1}{2}\int_x p_g(x)dx. \quad (6)$$

На основе 6 можно рассчитать дивергенцию Йенсена-Шеннона между двумя распределениями, используя формулу:

$$D_{JS}\big(p_r\|p_g\big) = \frac{1}{2}D_{KL}\Big(p_r\Big\|\frac{p_r+p_g}{2}\Big) + \frac{1}{2}D_{KL}\Big(p_g\Big\|\frac{p_r+p_g}{2}\Big) =$$

$$= \frac{1}{2}\left(\log 2 + \int_x p_r(x)\log\frac{p_r(x)}{p_r+p_g(x)}\right) + \frac{1}{2}\left(\log 2 + \int_x p_g(x)\log\frac{p_g(x)}{p_r+p_g(x)}dx\right) = \frac{1}{2}(\log 4 + L(G,D^*)) \quad (7)$$

и формулу

$$L(G,D^*) = 2D_{JS}\big(p_r\|p_g\big) - 2\log 2. \quad (8)$$

Процесс обучения генеративно-состязательной сети (*GAN*) включает использование функции потерь, которая определяет сходство между сгенерированным распределением данных (p_g) и распределением реальной выборки (p_r). Одной из часто используемых функций потерь является дивергенция Дженсена-Шеннона, которая вычисляет разницу между двумя распределениями, когда дискриминатор оптимален. Оптимальный генератор (G^*), который может идеально воспроизвести реальное распределение данных, приводит к минимальным потерям $L(G^*,D^*) = -2\log 2$ [14, 15].

Существует несколько разновидностей *GAN*, которые предназначены для различных приложений и контекстов. Например, при полунаблюдаемом обучении дискриминатор обновляется для присвоения реальных меток классам с *1* по *K-1* и поддельной метки классу *K*, при этом генератор пытается обмануть дискриминатор, чтобы присвоить меньшую метку [16].

В данном исследовании для улучшения обучения *DCGAN* был использован модифицированный метод одностороннего сглаживания меток. Метод сочетает в себе два метода оценки сходства изображений, а именно функцию потерь и индекс структурного сходства (*SSIM*) [17]. Функция потерь, такая как средняя квадратичная ошибка или средняя абсолютная ошибка, измеряет расхождение между эталонным и сгенерированным изображениями. Между тем, *SSIM* рассчитывает численное представление сходства между двумя изображениями, используя три параметра: яркость, контраст и структуру. Итоговая оценка варьируется от 0 до 1, где оценка 1 означает, что изображения полностью схожи.

В процессе проектирования генеративно-состязательной нейронной сети возникает компромисс между производительностью модели и ее размером. Чем глубже сеть и чем больше в ней фильтров, тем большее количество параметров она будет содержать [18]. Например, слой свертки с размерностью *[8, 8, 256]* содержит примерно 0,59 миллиона параметров, а слой с размерностью *[4, 4, 512]* – 2,3 миллиона параметров. Это приводит к увеличению времени обучения. Однако, регулируя глубину и количество фильтров, можно экспериментировать и найти конфигурацию, которая обеспечивает оптимальный баланс между производительностью и размером модели.

После проведения нескольких испытаний была определена следующая оптимальная конфигурация:

- размеры фильтров: [64, 128, 128, 128, 256, 256, 512];
- размер ядра: [7, 7, 7, 7, 7, 3, 3, 3];
- размер входного изображения: [256, 256, 3];
- размер пакета: 12.

3. Обучение нейронной сети

Для получения достаточного количества обучающих данных для генеративной состязательной нейронной сети необходимы пары изображений человека с маской и без нее. Однако сбор таких данных вручную может быть затруднен из-за необходимости сопоставления входных и выходных изображений одного и того же человека [19]. Для решения этой проблемы была разработана система, позволяющая генерировать базу данных лиц с масками, используя существующие базы данных изображений лиц.

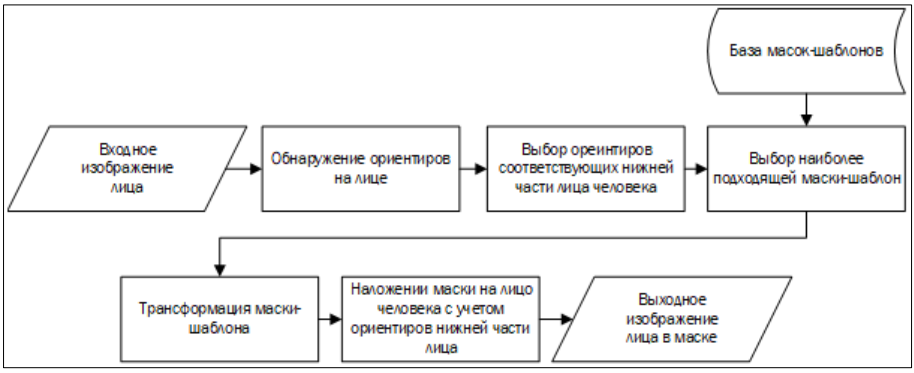

Рис. 2. Блок-схема алгоритма добавления маски на изображение лица человека
Fig. 2. Block diagram of the algorithm for adding a mask to the human face image

В качестве основной базы данных использовалась база данных *Labeled Faces in the Wild* Массачусетского университета, которая содержит более 5 000 изображений человеческих лиц. Для точного размещения маски на лицах использовался предварительно обученный детектор ориентиров на лицах *dlib*. Этот инструмент не только определяет ориентиры, но и вычисляет наклон лица, что крайне важно для правильного выравнивания изображения маски.

Наклон лица учитывается при выборе наиболее подходящего шаблона маски из набора имеющихся масок. Затем шаблон маски преобразуется на основе шести ключевых точек нижней части лица для достижения наилучшего прилегания. Процесс изображен на блок-схеме, показанный на рис. 2, а пример полученного изображения с маской и без маски показан на рис. 3.

Рис.3. Пример сгенерированных изображений лиц в маске
Fig.3. Example of generated images of faces in the mask

4. Практические результаты

Нейронная сеть обучалась в течение 50 эпох, используя 500 случайно выбранных изображений лиц с масками из собранной базы данных в каждой эпохе. Ход обучения отслеживался с помощью функции потерь и индекса структурного сходства (*SSIM*).

Выбор подходящей функции потерь имеет решающее значение для эффективности модели. Для решения этой задачи была использована комбинация двух функций потерь - среднеквадратичной ошибки (*MSE*) и средней абсолютной ошибки (*MAE*). Эти функции оценивают разницу в пикселях между изображением, полученным с помощью модели, и истинным изображением из набора данных до применения маски.

Чтобы модель могла создать реалистичное изображение лица с маской, необходимо, чтобы она понимала особенности, скрытые маской, такие как рот и нос, и, возможно, даже эмоции, отображаемые на видимых частях лица, таких как глаза. Комбинация функций потерь

позволяет модели научиться обобщать любое лицо, а не только те, которые есть в наборе данных для обучения, даже если она не может идеально соответствовать каждому пикселю.

SSIM, оценивает сходство между двумя изображениями. Он выдает числовой балл (от 0 до 1) на основе взвешенной комбинации яркости, контраста и структуры, а не просто подсчитывает количество совпадающих пикселей.

В табл. 1 представлена статистика производительности нейронной сети на этапе обучения для реконструкции лиц. В таблице показано значение функции потерь и точность сети во время обучения и тестирования.

Табл. 1. Статистика производительности нейронной сети
Table 1. Neural network performance statistics

№ эпохи	Loss (обучение)	SSIM (обучение)	Loss (тестовый)	SSIM (тестовый)
1	0,3956	0,7597	0,3686	0,7255
2	0,2877	0,8067	0,2792	0,7157
3	0,1968	0,8084	0,1957	0,7566
4	0,1570	0,8252	0,1760	0,7677
5	0,1542	0,8415	0,1794	0,8104
…	…	…	…	…
46	0,0625	0,9304	0,0727	0,9496
47	0,0628	0,9243	0,0764	0,9215
48	0,0605	0,9221	0,0738	0,8960
49	0,0600	0,9064	0,0721	0,9117
50	0,0606	0,9308	0,0717	0,9546

На рис. 4–5 представлены результаты обучения нейронной сети.

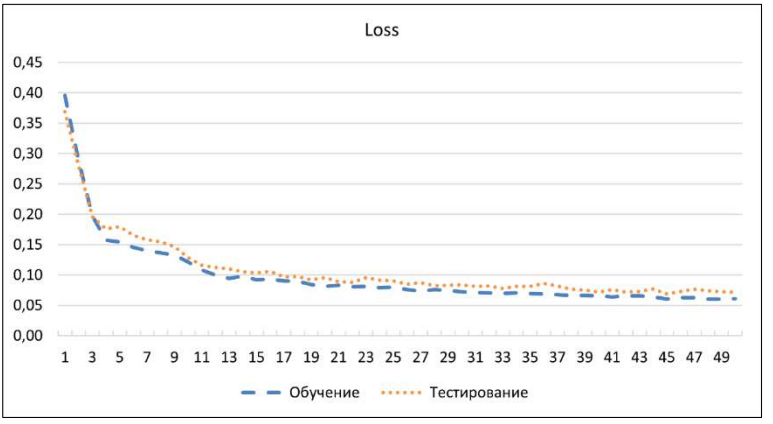

Рис. 4. График изменения функции потери во время обучения
Fig. 4. Graph of the change in the loss function during training

На рис. 6 представлены результаты, полученные в процессе обучения. На рисунке представлены три изображения: оригинальное изображение слева, изображение с маской в центре и восстановленные скрытые части лица справа.

Как показывают представленные данные, предложенная нейронная сеть демонстрирует замечательные результаты на тестовых наборах данных. Сеть демонстрирует способность к обобщению и, обладает достаточным уровнем распознавания эмоций, чтобы генерировать

улыбающиеся или грустные лица. Тем не менее, все еще есть возможности для дальнейшего совершенствования.

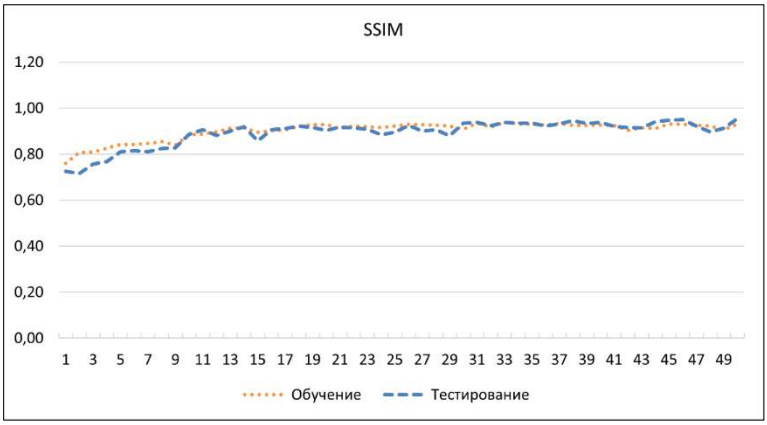

Рис. 5. График изменения функции схожести изображений во время обучения
Fig. 5. The graph of the change in the image similarity function during training

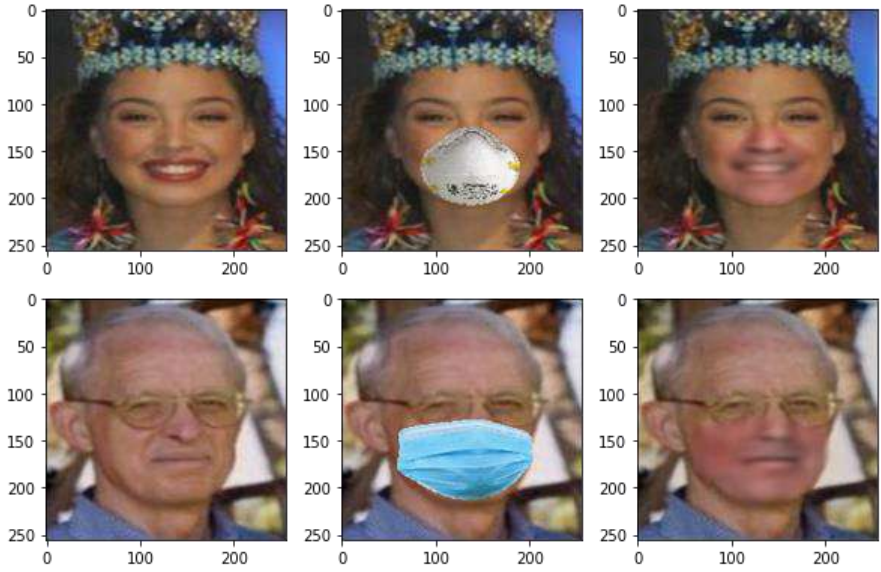

Рис. 6. Результаты восстановления изображения лица человека обученной нейронной сетью
Fig. 6. Results of reconstruction of a human face image by a trained neural network

5. Заключение

В заключение следует отметить, что результаты обучения *DCGAN* для восстановления скрытых участков лица на двух оцениваемых параметрах показало высокие результаты. Использование функции потерь, основанной на разнице пикселей, и метрики сходства изображений, предложенной Вангом, позволило получить значение точности 6,06% при обучении и 7,17% на тестовом наборе данных для функции потерь и 93,08% при обучении и 95,46% на тестовом наборе данных для метрики *SSIM*. Эти результаты показывают, что реконструированные изображения скрытых областей лица очень похожи по деталям и очертаниям на оригинальные изображения до применения масок, что делает их полезными для идентификации лиц в маска

Данная исследовательская работа подчеркивает важность разработки инновационных решений для решения проблем, возникающих при идентификации лиц со скрытыми областями, и потенциал использования глубоких сверточных генеративно-состязательных сетей для восстановления скрытых частей лиц и повышения точности распознавания. Результаты данного исследования имеет значительный вклад в области компьютерного зрения и его различных применений, подчеркивая необходимость постоянных инвестиций в исследования и разработки для продвижения передового опыта в этой области.

Список литературы / References

[1] Goldstein A.J., Harmon L.D., Lesk A.B. Identification of human faces. Proceedings of the IEEE, vol. 59, issue 5, 1971, pp. 748–760.

[2] Ngan M.L., Grother P.J., Hanaoka K.K. Ongoing Face Recognition Vendor Test (FRVT) Part 6B: Face recognition accuracy with face masks using post-COVID-19 algorithms. NIST Interagency/Internal Report (NISTIR) no. 8331, 2020, 83 p.

[3] Блейхут Р.Э.Быстрые алгоритмы цифровой обработки сигналов. М., Мир, 1989 г., 448 стр. / Blahut R.E. Fast Algorithms for Digital Signal Processing. Addison-Wesley, 1985, 455 p.

[4] Оппенгейм А., Шафер Р. Цифровая обработка сигналов. М., Техносфера, 2019 г., 1048 стр. / Oppenheim A.V., Schafer R.W. Discrete-Time Signal Processing. Pearson, 2009, 1144 p.

[5] Бершадская Е.Г., Маркин Е.И., Мартышкин А.И. Методы идентификации личности по изображению лица. XXI век итоги прошлого и проблемы настоящего плюс, том 9, вып. 1, стр. 49-53 / Bershadskaya E.G., Markin E.I., Martyshkin A.I. Methods for personal image identification. XXI Century: Resumes of the Past and Challenges of the Present plus, vol. 9, issue 1, pp. 49-53 (in Russian).

[6] Choi J., Han B. MCL-GAN: Generative Adversarial Networks with Multiple Specialized Discriminators. arXiv2107.07260 , 2021, 20 p.

[7] Adler J., Lunz S. Banach wasserstein GAN. In Proc. of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 6755-6764.

[8] Arjovsky M., Bottou L. Towards principled methods for training generative adversarial networks. arXiv1701.04862, 2017, 17 p.

[9] Маркин Е.И., Мартышкин А.И., Зупарова В.В. Анализ возможностей нейронных сетей для генерации фотореалистичных изображений. Современные информационные технологии, вып. 33, 2021, стр. 30-34 / Маркин Е.И., Мартышкин А.И., Zuparova V.V. Analysis of the capabilities of neural networks to generate photorealistic images. Modern Information Technology, vol. 33, 2021, pp. 30-34 (in Russian).

[10] Chi N.N.K. Active shape models their training and application. Bachelor' Thesis. International University HCMC, Vietnam, 2011.

[11] Gulrajani I., Ahmed F. et al. Improved training of wasserstein GANs. In Proc. of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 5769-5779.

[12] Salimans T., Goodfellow I. et al. Improved techniques for training GANs. In Proc. of the 30st International Conference on Neural Information Processing Systems, 2016, pp. 2234-2242.

[13] Huszár F. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? arXiv1511.05101, 2015, 9p.

[14] Zhang W., Liu Y. et al. RankSRGAN: Super Resolution Generative Adversarial Networks with Learning to Rank. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 10, 2022, pp. 7149-7166.

[15] Zhou E., Fan H. et al. Extensive facial landmark localization with coarse-to-fine convolutional network cascade. In Proc. of the IEEE International Conference on Computer Vision Workshops, 2013, pp. 386-391.

[16] Wei J., Liu M. et al. DuelGAN: A Duel Between Two Discriminators Stabilizes the GAN Training. Lecture Notes in Computer Science, vol. 13683, 2022, pp. 290-317.

[17] Wang Z., Bovik A.C. et al. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing, vol. 13, issue 4, 2004, pp. 600-612.

[18] Yamashita R., Nishio M. et al. Convolutional neural networks: an overview and application in radiology. Insights into Imaging, vol. 9, issue 4, 2018, pp. 611-629.

[19] Heo J.Three-dimensional generic elastic models for two-dimensional pose synthesis and face recognition Proquest, Umi Dissertation Publishing, 2011, 154 p.

Информация об авторах / Information about authors

Евгений Игоревич МАРКИН – кандидат технических наук, ассистент кафедры «Программирование». Сфера научных интересов: распознавание образов, обработка цифровых изображений, системы компьютерного зрения, машинное обучение и нейронные сети.

Evgeny Igorevich MARKIN – Candidate of Technical Sciences, Assistant of the Department "Programming". Research interests: pattern recognition, digital image processing, computer vision systems, machine learning and neural networks.

Валентина Владимировна ЗУПАРОВА – аспирант кафедры «Программирование». Сфера научных интересов: обработка данных, распознавание образов, распределенные вычислительные системы.

Valentina Vladimirovna ZUPAROVA – Postgraduate Student of the Department of Programming. Research interests: data processing, pattern recognition, distributed computing systems.

Алексей Иванович МАРТЫШКИН – кандидат технических наук, доцент, заведующий кафедрой «Программирование». Сфера научных интересов: обработка данных, моделирование вычислительных систем, исследование высокопроизводительных систем, распределенные вычислительные системы.

Alexey Ivanovich MARTYSHKIN – Candidate of Technical Sciences, Associate Professor, Head of the Department "Programming". Research interests: data processing, modeling of computing systems, research of high-performance systems, distributed computing systems.

DOI: 10.15514/ISPRAS-2022-34(6)-9

Модификация метода выделения контуров объекта в интеллектуальных системах

А.И. Мартышкин, ORCID: 0000-0002-3358-4394 <mai@penzgtu.ru>
Е.Г. Бершадская, ORCID: 0000-0003-3142-7958 <bereg.50@mail.ru>
Е.И. Маркин, ORCID: 0000-0003-4848-8757 <evgeniymarkin1@gmail.com>
В.В. Зупарова, ORCID: 0000-0002-7903-083X <zuparova_vv@mail.ru>

Пензенский государственный технологический университет,
440039, Россия, г. Пенза, проезд Байдукова/ул. Гагарина, д. 1а/11.

Аннотация. Данная исследовательская работа посвящена использованию компьютерного зрения в интеллектуальных системах для анализа контуров человека. С ростом технологий в различных отраслях промышленности возникает необходимость повышения эффективности систем "человек-компьютер". Предлагаемый метод использует видеокамеру и компьютерное программное обеспечение для обнаружения человека на изображении и его обработки с помощью библиотеки OpenCV и языка программирования C++. В статье рассматриваются существующие методы обнаружения человека, анализируется альтернативный метод, использующий компьютерное зрение, и разрабатывается новый метод обнаружения человека. Модификация включает в себя применение фильтра Кувахара для размытия изображения и алгоритма Собеля для выделения контуров. Области применения данной технологии включают обеспечение безопасности на транспортных узлах и в местах скопления людей, удаленный мониторинг здоровья, усиленный контроль на границах и охраняемых объектах, а также интерактивную рекламу и развлечения.

Ключевые слова: компьютерное зрение; нейронные сети; интеллектуальные системы; контур человека.

Для цитирования: Мартышкин А.И., Бершадская Е.Г., Маркин Е.И., Зупарова В.В. Модификация метода выделения контуров объекта в интеллектуальных системах. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 127-136. DOI: 10.15514/ISPRAS-2022-34(6)-9

Modification of the Method of Object Contours Extraction in Intelligent Systems

A.I. Martyshkin, ORCID: 0000-0002-3358-4394 <mai@penzgtu.ru>
E.G. Bershadskaya, ORCID: 0000-0003-3142-7958 <bereg.50@mail.ru>
E.I. Markin, ORCID: 0000-0003-4848-8757 <evgeniymarkin1@gmail.com>
V.V. Zuparova, ORCID: 0000-0002-7903-083X <zuparova_vv@mail.ru>

Penza State Technological University,
1a/11, Baidukova Passage/Gagarin st., Penza, 440039, Russia

Abstract. This research paper focuses on the use of computer vision in intelligent systems to analyze human contours. With the growth of technology in various industries, there is a need to improve the efficiency of human-computer systems. The proposed method uses a video camera and computer software to detect a person in the image and process it using the OpenCV library and C++ programming language. The paper reviews existing human detection methods, analyzes an alternative method that uses computer vision, and develops a new method for human detection. Modifications include the use of the Kuwahar filter for image blurring and the Sobel algorithm for outline extraction. Applications for this technology include security at transportation

hubs and crowded areas, remote health monitoring, enhanced control at borders and secure facilities, and interactive advertising and entertainment.

Keywords: computer vision; neural networks; intelligent systems; human contour.

For citation: Martyshkin A.I., Bershadskaya E.G., Markin E.I., Zuparova V.V. Modification of the Method of Object Contours Extraction in Intelligent Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 127-136 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-9

1. Введение

Широкое внедрение технологий в различных отраслях промышленности во всем мире приводит к росту проектирования и информационных систем. С быстрым развитием компьютерных технологий и растущим спросом на автоматизацию возникает необходимость повышения эффективности систем "человек-компьютер". Исследования показывают, что разработка интеллектуальных самоорганизующихся систем является важнейшим направлением. Цель искусственного интеллекта – дать машинам возможность анализировать информацию и принимать решения [1]. В данном исследовании мы сосредоточились на компьютерном зрении, области, входящей в состав интеллектуальных систем, для анализа контуров человека с помощью технологии распознавания изображений.

Приложения этой технологии включают в себя:

- безопасность в транспортных узлах и местах массового скопления людей,
- усиленный контроль на границах и охраняемых объектах,
- удаленный мониторинг здоровья в зонах повышенной опасности,
- интерактивная реклама и развлечения.

Предлагаемый метод использует видеокамеру и компьютер со специальным программным обеспечением для обнаружения человека на изображении. Он работает автономно и обрабатывает изображение с помощью библиотеки *OpenCV* и языка программирования *C++*. Целью данного исследования является обзор существующих методов обнаружения человека, анализ альтернативного метода с использованием компьютерного зрения и разработка нового метода обнаружения человека путем нахождения и анализа его контуров.

2. Обзор существующих методов обнаружения человека

Современные методы обнаружения человека в системах безопасности включают детекторы движения, которые используют пироэлектрические датчики для регистрации инфракрасных волн, излучаемых телами людей. Обнаружение осуществляется с помощью оптической системы, состоящей из линз Френеля и сферического расположения от 20 до 60 линз. Размер контролируемой зоны определяется количеством и расположением линз. Метод надежен, но не обладает визуальным восприятием и требует специальных устройств [2].

Другим методом обнаружения человека является измерение его движений, что может быть использовано в медицинских целях. Это можно сделать с помощью смартфонов и специальных программ, которые отслеживают фазы сна и физическую активность. Этот метод требует использования смартфона на поверхности кровати, что может не гарантировать комфорт и безопасность.

Фитнес-браслеты и смарт-часы также используются для отслеживания основных показателей здоровья человека, таких как частота сердечных сокращений, температура тела и фазы сна. Однако недостатком этого метода является требование носить устройство и возможность неточностей в измерениях. Альтернативой является система, основанная на обнаружении контуров человека с помощью компьютерного зрения, которая не требует носимых датчиков.

Традиционным методом мониторинга состояния здоровья человека является использование контактных устройств, таких как носимые датчики и проводные устройства, используемые в

лабораториях. Недостатком этого метода является необходимость прямого контакта с устройствами, что не всегда может быть приемлемо. Технология обнаружения контуров человека предлагает более удобный и доступный вариант мониторинга функции дыхания без необходимости использования носимых датчиков или проводов [3].

Наконец, обнаружение человека также используется в интерактивной рекламе, где реклама реагирует на присутствие человека и привлекает его внимание.

Обнаружение человека широко используется в различных областях, включая развлечения, умные дома и обработку изображений. Одним из популярных примеров является *Kinect* от *Microsoft*, бесконтактный игровой контроллер с сенсорным экраном для *Xbox 360* и персональных компьютеров под управлением *Windows*. Он взаимодействует с пользователем с помощью речи и движений. Другое распространенное применение – датчики движения, которые управляют различными устройствами, такими как лампочки, видеокамеры или кондиционеры, в "умных" домах.

Проблема распознавания контуров человека становится все более важной с ростом практических потребностей человека. Предлагаемое решение заключается в использовании автоматического извлечения и анализа контуров человека на различных изображениях. Анализ изображений относится к процессу извлечения информации из изображений с помощью автоматических систем, обычно путем нахождения внешнего контура объектов и регистрации координат контура. Однако определение контура на изображении может быть неоднозначным из-за таких вопросов, как масштаб, текстура и семантика. Поэтому часто предпочтительнее анализировать проблему поиска контуров на изображении, а не поиска контуров объектов [4].

Нахождение контуров на изображении имеет решающее значение для решения различных задач, таких как распознавание человека. Задача обработки изображений заключается в построении алгоритма для автоматического анализа изображений компьютером и извлечения информации.

Рис. 1. Алгоритм метода распознавания контуров человека
Fig. 1. Algorithm of the human contour recognition method

Обнаружение человека является областью исследований как в биологии, так и в компьютерном зрении. Эти области находятся во взаимовыгодных отношениях благодаря

междисциплинарному обмену. Предложенный метод, хотя и не является совершенным, имеет преимущество в том, что обеспечивает точные результаты при минимальных технических ресурсах и максимальном удобстве для человека. Точность предлагаемого метода можно оценить по его способности правильно распознавать контуры человека. К ограничениям предложенного метода относятся высокие требования к качеству видеокамеры.

3. Описание метода решения задачи

Метод детектирования контура человека с помощью компьютерного зрения предусматривает реализацию следующего алгоритма (рис. 1).

3.1 Алгоритмы для размытия изображения

В данном исследовании рассматривается влияние алгоритмов размытия изображения на эффективность обнаружения контуров человеком. Размытие изображения является важным этапом предварительной обработки изображения, поскольку оно способно значительно повысить точность обнаружения контуров человека [5].

Одним из широко используемых алгоритмов размытия изображения является фильтр Гаусса (рис. 2). Этот фильтр использует функцию Гаусса в качестве импульсной переходной функции, что приводит к отсутствию проскакивания и максимизации постоянной времени. Такое поведение объясняется минимально возможной групповой задержкой гауссова фильтра. Гауссовский фильтр широко используется для фильтрации двумерных сигналов и снижения уровня шума [6].

Рис. 2. Обработка изображения фильтром Гаусса
Fig. 2. Processing the image with a Gaussian filter

Рис. 3. Обработка изображения медианным фильтром
Fig. 3. Processing the image with a median filter

Другим популярным алгоритмом размытия изображений является медианный фильтр (рис. 3), который используется для снижения уровня шума в цифровых сигналах и изображениях. Этот фильтр сортирует значения в своем окне в порядке возрастания или убывания и выбирает в качестве выхода медианное значение, которое является значением в центре отсортированного списка. В случае четных чисел в окне, выходом является среднее значение двух центральных значений. Медианная фильтрация – эффективный подход для обработки сигналов, подверженных импульсному шуму [7].

3.2 Алгоритмы для выделения контуров объекта

После того как изображение размыто, важно выделить контуры человека, что упростит восприятие программы, поскольку четким остается только тот контур, который необходимо проанализировать. Контур определяется как линия, на которой наблюдаются изменения яркости или цвета. Важно отметить, что контуров объекта меньше, чем контуров на изображении, и поэтому требуется дополнительный анализ контуров перед определением нужных контуров объекта.

Двумя популярными алгоритмами выделения контуров являются детектор Канни (рис. 4) [8] и оператор Лапласа (рис. 5) [9, 10]. Детектор Канни, разработанный Д. Канни в 1986 году, представляет собой оператор, который определяет границы изображения. С другой стороны, оператор Лапласа используется в качестве детектора краев, и библиотека *OpenCV* предоставляет его дискретную реализацию - функцию Лапласа.

Рис. 4. Обработка изображения детектором Канни
Fig. 4. Image processing with the Canny detector

Рис. 5. Обработка изображения оператором Лапласа
Fig. 5. Image processing by the Laplace operator

Известно несколько методов распознавания контуров человека, включая метод Виолы-Джонса [11], метод сканирующего окна [12], признаки Хаара [13], модель машинного обучения [14], каскадную модель [15], гистограмму направленных градиентов [16] и другие.

В данном исследовании мы анализируем характеристики существующих алгоритмов размытия изображения, выделения контуров и распознавания контуров человека, чтобы определить наиболее подходящий и эффективный алгоритм для разработки нашего проекта по распознаванию контуров человека на изображении.

Модифицированный алгоритм распознавания контуров человека

Для повышения эффективности алгоритма распознавания контуров человека очень важно начать с применения алгоритма размытия изображения. В данном случае используется фильтр Кувахара (рис. 6) [17], поскольку он позволяет получить изображения с четкими границами изображенных объектов.

Рис. 6. Обработка изображения фильтром Кувахара
Fig. 6. Processing the image with the Kuwahara filter

Фильтр Кувахара выполняет нелинейную фильтрацию изображения, сохраняя при этом четкие края. В результате изображение выглядит как нарисованное. Маска фильтра Кувахара показана на рис. 7.

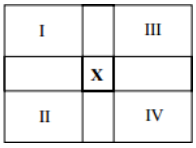

Рис. 7. Маска фильтра Кувахара
Fig. 7. Kuwahara Filter Mask

Алгоритм фильтра Кувахары работает следующим образом:

- маска размером $(2r + 1) \times (2r + 1)$ накладывается на все области пикселей;
- область делится на четыре сектора, имеющих размер $r \times r$ каждый;
- для каждого сектора производится расчет средней интенсивность и дисперсии;
- являющийся центральным пикселем X принимает среднее значение интенсивности сектора с наименьшей дисперсией.

После размытия изображения необходимо выделить контуры человека для более легкого восприятия программы, оставив в фокусе только контур, подлежащий анализу. Для этого используется алгоритм Собеля (рис. 8) [18], так как он надежен и дает визуально привлекательные результаты.

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, G_y = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad (1)$$

Рис. 8. Обработка изображения алгоритмом Собеля
Fig. 8. Image processing by the Sobel algorithm

Представленные коэффициенты оператора Собеля имеют форму матрицы, как показано в формуле (1). В центре матрицы находится коэффициент для данного пикселя, в правом нижнем углу – правый нижний коэффициент и так далее. Наконец, метод Виола-Джонса (рис. 9) непосредственно применяется к обработанному изображению. Выбор метода Виолы-Джонса для распознавания контуров человека обоснован оптимальным соотношением эффективности распознавания и скорости работы.

Рис. 9. Результат работы системы распознавания контуров человека
Fig. 9. The result of the human contour recognition system

4. Реализация модифицированного метода распознавания контура человека

Модифицированный метод распознавания контуров человека на изображениях реализован с использованием библиотеки *OpenCV* и языка программирования *C++*. Процесс начинается с захвата изображения с видеокамеры, затем выполняется размытие и выделение контуров для повышения чувствительности системы. Затем к обработанному изображению применяется алгоритм распознавания контуров человека. Если контуры человека обнаружены, на изображении появляется зеленый прямоугольник, указывающий на их местоположение.

Для оценки точности предложенного метода было проведено сравнение результатов обнаружения модифицированного алгоритма и метода Виолы-Джонса на 40 изображениях. Результаты, представленные в табл. 1, показывают, что модифицированный алгоритм имеет более низкий коэффициент ошибок типа II, что означает, что большая доля объектов была обнаружена, и более низкий коэффициент ошибок типа I, что означает меньшее количество ложных тревог. Хотя метод Виолы-Джонса имеет более короткое время обработки, он все еще отстает от модифицированного алгоритма по точности.

Табл. 1. Сравнительный анализ результатов детектирования контуров человека
Table 1. Comparative analysis of the results of detection of human contours

	Количество ложных срабатываний	Доля обнаруженных объектов	Время на обработку изображения (сек)
Метод Виолы-Джонса	14	79,5%	0,187
Модифицированный алгоритм	8	95,9%	0,154

5. Заключение

Модифицированный метод распознавания контуров человека с помощью компьютерного зрения является экономически эффективным, визуально воспринимаемым и адаптируемым путем усовершенствования программного обеспечения. Система состоит из компьютера, видеокамеры и набора алгоритмов для обработки изображений и распознавания контуров. Алгоритм, используемый для обработки, включает фильтр Кувахара для размытия, фильтр Собеля для выделения контуров и метод Виолы-Джонса для распознавания контуров. Результаты сравнительного анализа между предложенным методом и методом Виола-Джонса показывают, что модифицированный метод имеет меньшее количество ложных срабатываний и большую долю обнаруженных объектов.

Несмотря на свои преимущества, предложенный метод также имеет ограничения, такие как необходимость использования высококачественной видеокамеры и определенный процент ошибок. Однако преимущества использования технического зрения делают этот метод жизнеспособной альтернативой традиционным методам обнаружения человека, особенно в таких отраслях, как безопасность, медицина, реклама и развлечения. При дальнейшем совершенствовании модифицированный метод может быть интегрирован в различные приложения и достичь еще большего успеха.

Список литературы / References

[1] Gregor K., Besse F. Self-Organizing Intelligent Matter: A blueprint for an AI generating algorithm. arXiv2101.07627, 2021, 13 p.
[2] Yun J., Lee S. Human Movement Detection and Identification Using Pyroelectric Infrared Sensors. Sensors. vol. 14, issue 5, 2014, pp. 8057–8081
[3] Imano W., Kameyama K. et al. Non-Contact Respiratory Measurement Using a Depth Camera for Elderly People. Sensors, vol. 20, issue 23, 2020, pp. 1–12
[4] Бершадская Е.Г., Маркин Е.И., Мартышкин А.И. Методы идентификации личности по изображению лица. XXI век итоги прошлого и проблемы настоящего плюс, том 9, вып. 1, стр. 49-53 / Bershadskaya E.G., Markin E.I., Martyshkin A.I. Methods for personal image identification. XXI Century: Resumes of the Past and Challenges of the Present plus, vol. 9, issue 1, pp. 49-53 (in Russian).
[5] Mittal M., Verma A. et al. An Efficient Edge Detection Approach to Provide Better Edge Connectivity for Image Analysis. IEEE Access, vol. 7, pp. 33240-33255
[6] Jain A., Gupta R. Gaussian filter threshold modulation for filtering flat and texture area of an image. In Proc. of the International Conference on Advances in Computer Engineering and Applications, 2015, pp. 760-763
[7] Ramadhan A., Mahmood F., Elci A. Image denoising by median filter in wavelet domain. International Journal of Multimedia & Its Applications (IJMA), vol.9, issue 1, 2017, pp. 31-40.
[8] Xu Z., Baojie X., Guoxin W. Canny edge detection based on Open CV. In Proc. of the 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), 2017, pp. 53-56.
[9] Vliet L.J., Young I.T., Beckers G.L. A nonlinear laplace operator as edge detector in noisy images. Computer Vision, Graphics, and Image Processing, vol. 45, issue 2, 1989, pp. 167-195
[10] Bansal R., Raj G., Choudhury T. Blur image detection using Laplacian operator and Open-CV. In Proc. of the International Conference on System Modeling & Advancement in Research Trends (SMART), Moradabad, India, 2016, pp. 63-67

[11] Damanik R.R., Sitanggang D. et al. An application of viola jones method for face recognition for absence process efficiency. Journal of Physics: Conference Series, vol. 1007, 2018, article id. 012013, 8 p.
[12] Shakil S., Lee C., Keilholz S.D. Evaluation of sliding window correlation performance for characterizing dynamic functional connectivity and brain states. NeuroImage, vol. 133, 2016, pp. 111-128.
[13] Ma S. Bai L. A face detection algorithm based on Adaboost and new Haar-Like feature. In Proc. of the 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2016, pp. 651-654.
[14] Sun X., Wu P., Hoi C.H. Face detection using deep learning: An improved faster RCNN approach. Neurocomputing, vol. 299, 2018, pp. 42-50.
[15] Singh A., Herunde H., Furtado F. Modified Haar-cascade model for face detection issues. International Journal of Research in Industrial Engineering, vol. 9, issue 2, 2022, pp.143-171.
[16] Eng S. K., Ali H. et al. Facial expression recognition in JAFFE and KDEF Datasets using histogram of oriented gradients and support vector machine. IOP Conference Series: Materials Science and Engineering, vol. 705, 2019, article id. 012031, 7 p.
[17] Biswas S., Ghoshal D. A model of noise reduction using Gabor Kuwahara filter. In Proc. of the 4th International Conference on Advanced Computing and Communication Systems (ICACCS), 2017, pp. 1-5
[18] Lang Y., Zheng D. An Improved Sobel Edge Detection Operator. In Proc. of the 2016 6th International Conference on Mechatronics, Computer and Education Informationization (MCEI 2016), 2016, pp. 590-593.

Информация об авторах / Information about authors

Алексей Иванович МАРТЫШКИН – кандидат технических наук, доцент, заведующий кафедрой «Программирование». Сфера научных интересов: обработка данных, моделирование вычислительных систем, исследование высокопроизводительных систем, распределенные вычислительные системы.

Alexey Ivanovich MARTYSHKIN – Candidate of Technical Sciences, Associate Professor, Head of the Department "Programming". Research interests: data processing, modeling of computing systems, research of high-performance systems, distributed computing systems.

Елена Григорьевна БЕРШАДСКАЯ – кандидат технических наук, профессор, профессор кафедры «Программирование». Сфера научных интересов: системы массового обслуживания, большие данные, распределенные вычислительные системы.

Elena Grigorievna BERSHADSKAYA – Candidate of Technical Sciences, Professor, Professor of the Department of Programming, Penza State Technological University. Research interests: queuing systems, big data, distributed computing systems.

Евгений Игоревич МАРКИН – кандидат технических наук, ассистент кафедры «Программирование». Сфера научных интересов: распознавание образов, обработка цифровых изображений, системы компьютерного зрения, машинное обучение и нейронные сети.

Evgeny Igorevich MARKIN – Candidate of Technical Sciences, Assistant of the Department "Programming". Research interests: pattern recognition, digital image processing, computer vision systems, machine learning and neural networks.

Валентина Владимировна ЗУПАРОВА – аспирант кафедры «Программирование». Сфера научных интересов: обработка данных, распознавание образов, распределенные вычислительные системы.

Valentina Vladimirovna ZUPAROVA – Postgraduate Student of the Department of Programming. Research interests: data processing, pattern recognition, distributed computing systems.

DOI: 10.15514/ISPRAS-2022-34(6)-10

Автоматическая разметка данных для сегментации изображений документов с использованием глубоких нейронных сетей

А.А. Михайлов, ORCID: 0000-0003-4057-4511<mikhailov@icc.ru>

Институт динамики систем и теории управления имени В.М. Матросова СО РАН,
664033, Россия, Иркутск, ул. Лермонтова, 134
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В статье предложен новый метод автоматической аннотации данных для решения задачи сегментации изображений документов с помощью глубоких нейронных сетей обнаружения объектов. В качестве исходных данных для разметки рассматривается формат помеченных файлов PDF. Особенность данного формата заключается в том, что он включает в себя скрытые метки, которые описывают логическую и физическую структуру документа. Для их извлечения разработано инструментальное средство, которое имитирует работу стековой машины вывода на печать согласно спецификации формата PDF. Для каждой страницы документа генерируются изображение, и аннотация в формате PASCAL VOC. Классы и координаты ограничивающих рамок вычисляются в процессе интерпретации помеченного PDF файла на основе меток. Для апробации метода была сформирована коллекция размеченных PDF файлов из которой в автоматическом режиме получены изображения страниц документов и аннотации для трех классов сегментации. На основе этих данных обучена нейронная сеть архитектуры EfficientDet D2. Произведено тестирование модели на данных из того же домена, размеченных вручную, которое подтвердило эффективность применения автоматически сгенерированных данных для решения прикладных задач.

Ключевые слова: сегментация документов; сегментация изображений документов; глубокие нейронные сети; обнаружение объектов

Для цитирования: Михайлов А.А. Автоматическая разметка данных для сегментации изображений документов с использованием глубоких нейронных сетей. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 137-146. DOI: 10.15514/ISPRAS-2022-34(6)-10

Automatic data labeling for document image segmentation using deep neural networks

A.A. Mikhailov, ORCID: 0000-0003-4057-4511 <mikhailov@icc.ru>

Matrosov Institute for System Dynamics and Control Theory of the SB RAS
134, Lermontova st., Irkutsk, 664033, Russia.
Ivannikov Institute for System Programming of the RAS,
25 Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The article proposes a new method for automatic data annotation for solving the problem of document image segmentation using deep object detection neural networks. The format of marked PDF files is considered as the initial data for markup. The peculiarity of this format is that it includes hidden marks that describe the logical and physical structure of the document. To extract them, a tool has been developed that simulates the operation of a stack-based printing machine according to the PDF format specification. For each page of the document, an image and annotation are generated in PASCAL VOC format. The classes and

coordinates of the bounding boxes are calculated during the interpretation of the labeled PDF file based on the labels. To test the method, a collection of marked up PDF files was formed from which images of document pages and annotations for three segmentation classes (text, table, figure) were automatically obtained. Based on these data, a neural network of the EfficientDet D2 architecture was trained. The model was tested on manually labeled data from the same domain, which confirmed the effectiveness of using automatically generated data for solving applied problems.

Keywords: document layout analysis; PDF accessibility; ANN models; artificial intelligence

For citation: Mikhailov A.A. Automatic data labeling for document image segmentation using deep neural networks. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 137-146 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-10

1. Введение

Произвольные документы являются распространённым способом представления информации. Они повсеместно распространены в веб-пространстве. Большой объем и свойства структуры таких документов делают их ценным источником в приложениях науки о данных и бизнес аналитики. Однако, как правило, они не сопровождаются явной семантикой необходимой для машинной интерпретации своего содержания так, как задумано их автором. Накапливаемая в них информация часто является неструктурированной и не стандартизированной. Анализ этих данных нуждается в их предварительной трансформации к структурированному представлению с заданной формальной моделью.

В литературе по анализу и распознаванию документов такую задачу принято называть анализом компоновки документов (Document layout analysis). В последние годы наряду с классическими методами анализа компоновки документов активно развиваются подходы на основе глубоких нейронных сетей обнаружения и классификации объектов на изображениях. О чем свидетельствуют результаты одной из ведущих научных конференций по анализу документов – ICDAR (International Conference on Document Analysis and Recognition). Методы на основе глубоких нейронных сетей являются эффективными и демонстрируют высокие результаты, но они очень требовательны к размеру и качеству обучающей выборки. Разметка изображений для обучения очень медленный, трудоемкий и дорогостоящий процесс. Одним из методов решения данной проблемы является автоматическая разметка данных.

Таким образом, целью данной работы является разработка метода автоматической разметки данных для обучения глубоких нейронных сетей сегментации изображений документов.

В статье предложен оригинальный метод автоматической аннотации данных из помеченных PDF файлов. Собрана коллекция документов, из которых сгенерирована обучающая выборка емкостью 28 000 изображений страниц, охватывающая типичные структурные элементы для документа: текст, таблица, рисунок. Обучена нейронная сеть архитектуры EfficientDet D2 на полученном наборе данных. Тестирование полученной модели показало, что автоматически сформированных набор данных подходит для решения задачи сегментации изображений документов.

2. Современное состояние исследований

Изображения документов часто генерируются из физических документов путем оцифровки, с использованием сканеров или различных программ генерации (принтеров). Многие документы, такие как газеты, журналы и брошюры, содержат очень сложную компоновку из-за размещения рисунков, заголовков и подписей, сложного фона, художественного форматирования текста и т. д. Человек использует множество дополнительных подсказок, таких как контекст, условные обозначения, информацию о языке. Автоматический анализ произвольного документа со сложной версткой является чрезвычайно сложной задачей и выходит за рамки возможностей современных систем анализа компоновки документов. В научной литературе предложено большое количество методов анализа компоновки

документов. Согласно статье [1] они могут быть разделены на три группы: методы классификации на основе областей [2, 3]; методы классификации на основе анализа пикселей [4, 5]; анализ связных компонент [6-8].

С ростом эффективности и популярности свёрточных нейронных сетей область их применения постоянно расширяется. Начиная с 2014 года известны первые попытки использования искусственных нейронных сетей для решения задачи анализа компоновки документов [9-12]. Эти работы продемонстрировали свою эффективность по сравнению с классическими подходами, что подтверждается результатами соревнования 2017 года на конференции ICDAR [13]. С другой стороны, соревнования 2019 года показало, что на разнообразных данных, с большим количеством классов (10) комбинирование классических методов [14] наиболее эффективно по сравнению с глубокими нейронными сетями. Это обусловлено отсутствием достаточного количества разнообразных размеченных данных с большим количеством классов. В то время как для частных случаев нейронные сети работают намного эффективнее [15]. Следует отметить, что для решения задачи анализа компоновки документов в этих работах используются либо нейронные сети архитектуры (Fast/Faster) R-CNN, либо авторские разработки. Для обучения нейронных сетей обычно используются открытые наборы размеченных данных, в редких случаях авторы статей указывают, что они разметили собственную обучающую выборку. Вручную размеченные наборы данных редко достигают размера в 20 000 экземпляров и чаще всего не публикуются в открытом доступе.

В сентябре 2021 года состоялась одна из ведущих конференция по анализу документов ICDAR2021[1]. В рамках этой конференции прошли соревнования как по сегментации изображений документов [16], так и по отдельным задачам сегментации, таких как обнаружение формул в изображениях документов [17], обнаружение списка литературы в научных статьях [18]. Большинство участников в своих решениях применяли сложные архитектурные решения, использующих комбинации нейронных сетей и моделей машинного обучения. Результаты победителей данных соревнований варьируются от 82% до 99% по точности и полноте, что позволяет использовать их для решения некоторых практических задач. Но, использование большого количества нейронных сетей накладывает ограничение на ресурсы, что может стать барьером при их внедрении. Использование таких систем на практике затруднено в данный момент и тем, что они очень сильно зависят от предметной области, для которой проводится анализ документов.

Это подтверждают работы 2022 года [19, 20] по обнаружению и извлечению таблиц из документов. Обнаружение таблиц в данном случае является подзадачей сегментации изображений документов только на один класс - таблица. Авторы этих работ для обнаружения таблиц использовали доступные в открытом доступе модели и системы, которые позволяют находить область таблицы. Результаты тестирования на данных из новой предметной области показали, что результаты по качеству и полноте сильно уступают (от 20% до 50%) результатам тестирования из оригинальных статей.

В начале 2020 – конце 2019 года появилось сразу несколько датасетов большого размера: PubTabNet[2] – 568 000 примеров из PubMed, который содержит в себе только таблицы; SciTSR[3] – 15 000 примеров из LaTeX источников, размеченных вручную; TIES-2.0[4] – автоматически сгенерированный датасет на основе набора данных UNLV, который содержит 500 000 примеров. Все эти датасеты содержат всего один класс (таблица).

В статье [1] описан подход генерации синтетических данных для тонкой настройки моделей сегментации документов, размеров в 18000 обучающих примеров. Отдельно следует

[1] https://icdar2021.org.
[2] https://github.com/ibm-aur-nlp/PubTabNet.
[3] https://github.com/Academic-Hammer/SciTSR
[4] https://github.com/shahrukhqasim/TIES-2.0

отметить набор данных PubLayNet[5], который включает в себя около 360 000 размеченных медицинских документов. Авторы этого набора использовали коллекцию документов из базы PubMed[6]. Каждый документ из этой коллекции сопровождается XML аннотацией с разметкой на 5 классов. Для формирования размеченной коллекции использовались методы нечеткого сравнения строк (расстояние Левенштейна) с последующей коррекцией ошибок.

Такой подход обладает рядом недостатков. Во-первых, из-за нечеткого сравнения строк достаточно высока вероятность ошибок разметки. Во-вторых, данный набор содержит небольшое количество классов (5 классов). Такой набор не подходит в полной мере для создания на его основе компьютерной модели. Это подтверждается в статье, опубликованной по результатам соревнования RDCL2019 [14]. В ней отмечается, что методы, основанные на нейронных сетях, сильно подвержены влиянию сравнительно небольшой обучающей выборки. Подходы на основе глубоких нейронных сетей, которые использовали сторонние наборы и специальные техники для увеличения обучающей выборки показали хорошие результаты, но не смогли составить конкуренцию решениям на основе классических методов. Одной из причин этого является чувствительность нейронных сетей к объему и разнообразию размеченных данных. В статье отмечается, что подходы на основе нейронных сетей для отдельных классов значительно превосходят классические методы (например, задача обнаружения таблиц на изображениях документов).

3. Описание метода

В сети Интернет содержится большое количество оригинальных документов LaTex. Одним из самых известных ресурсов является arXiv[7] – бесплатный электронный архив, который содержит более 1,7 миллиона научных статей и препринтов по физике, математике, астрономии, информатике, биологии, электротехнике, статистике, финансовой математике и экономике. Практически все статьи и препринты распространяются по одной из разновидностей свободной лицензии CC BY, что позволяется публиковать полученные на их основе результаты в открытом доступе. Помимо этого, в сети интернет содержится большое количество свободно распространяемых документов в формате MS Word, Open Office.

Форматы файлов MS Word, Open Office, LaTex содержат в себе всю необходимую информацию о компоновке документа, такую как заголовки, подзаголовки, таблицы, изображения, диаграммы, формулы и т.д.

Для того, что разработать методы автоматической размети данных на основе этих форматов необходимо исследовать алгоритмы процессинга приведенных форматов и модифицировать их для генерации растровых изображений и сопутствующих файлов с координатами ограничивающих прямоугольников, и классов. Это очень трудоемкий процесс, который связан с рядом трудностей. Например, для форматов MS Word и Open Office сгенерировать изображение документа и правильно извлечь координаты можно только для заранее известного драйвера принтера. В общем случае это сделать затруднительно.

Начиная с 2006 года, спецификация формата PDF включает в себя специальные метки, которые позволяют с помощью скрытых тегов описывать логическую структуру документа (листинг 1).

Теги позволяют пометить в PDF документе такие объекты как таблицы, списки и заголовки, формулы, рисунки, строки, ячейки, сноски, ссылки, параграфы и т.д. Кроме того, PDF – это универсальный формат, который позволяет отображать документы в любой операционной системе, программ е или устройстве в том виде, в котором он был создан.

[5] https://github.com/ibm-aur-nlp/PubLayNet
[6] https://pubmed.ncbi.nlm.nih.gov
[7] https://arxiv.org

```
<ispAnotation index=30>
Ключевые слова:
список ключевых слов, разделенных точкой с
запятой.
(стиль ispAnotation)
</ispAnotation>
<ispAnotation index=31>
Для цитирования:
Иванов И.И., Петров П.П. Заголовок статьи. Труды
ИСП РАН, том 1, вып. 2, 2019
г., стр. 15-19. DOI: 10.15514/ISPRAS-2019-1(2)-1
</ispAnotation>
<ispAnotation index=32>
<ispAnotation2 Знак index=33>
Благодарности:
</ispAnotation2 Знак>
<ispAnotation2 Знак index=34>
В этом блоке перечисляются организации,
поддерживающие исследование, описанное в статье,
гранты и т.д.
</ispAnotation2 Знак>
</ispAnotation>
```

Листинг 1. Результат интерпретации помеченного PDF файла
Listing 1. Tagged PDF parsing result

Основная идея метода автоматической разметки данных (рис 1) заключается в использовании помеченных PDF файлов в качестве промежуточного этапа между слабоструктурированными форматами и аннотированными данными. Такой подход гарантирует полное соответствие координат тегов в PDF с их координатами в сгенерированном изображении страниц, из-за их полной идентичности.

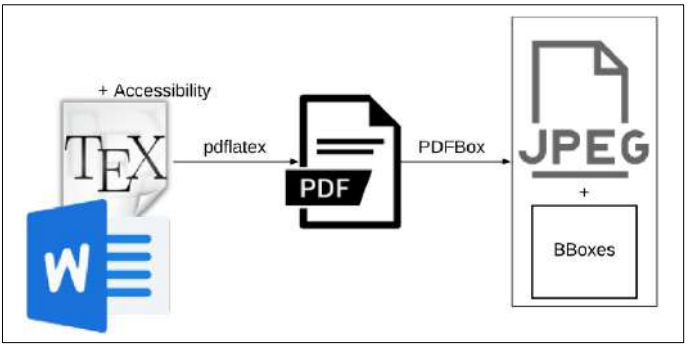

Рис. 1. Метод автоматической разметки данных для сегментации изображений документов с использованием нейронных сетей
Fig. 1. Automatic data labeling method for document image segmentation using neural networks

3.1 Формирование коллекции исходных данных

Коллекция исходных данных в количестве 1200 документов была сформирована из опубликованных в отрытом доступе технических заданий и нормативно-правовых актов[8].

[8] https://zakupki.gov.ru

3.2. Генерация помеченных PDF документов

Начиная с июля 2020 года, в основной дистрибутив LaTex был включен пакет Accessibility. Этот пакет предназначен для создания помеченных PDF документов с тегами из исходного кода LaTex. Для того, чтобы воспользоваться данной функций достаточно добавить в преамбулу исходного LaTex документа пакет Accessibility. Кроме того, современные версии Microsoft Office, Open Office, Libre Office и др. позволяют по исходному формату производить экспорт помеченных PDF (Tagged PDF) документов.

Для потоковой генерации помеченных PDF документов из файлов формата Microsoft Word использовался скрипт экспорта с помощью LibreOffice Community с включенной опцией «Tagged PDF (add document structure)». В результате было получено 1200 помеченных PDF файлов.

3.3. Описание классов сегментации

Название тегов в помеченных PDF документах, полученных из файлов Microsoft Word, задается стилевых файлом, который был использован при их создании. Из этого формируется одно из ограничений метода – необходимость сопоставить названия тегов с ограниченным набором заранее определенных классов.

В данной работе было принято решение использовать три класса:

- Text – текстовые блоки, содержащие однородный текст с единым форматированием (размером, жирностью, шрифтом, отступами между строк);
- Table – класс таблиц с границами, которые могут содержать объединенные ячейки по вертикали или по горизонтали; заголовок таблиц может отличаться от тела другим форматированием;
- Picture – класс, содержащий печати, подписи, изображения в документах.

Этот выбор обусловлен наличием тестового набора данных из того же домена с аналогичным набором классов для проведения сравнительной оценки разработанного метода.

3.4 Извлечение координат и классов

PDF - это универсальный межплатформенный формат, который позволяет отображать документы в любой операционной системе, программе или устройстве в том виде, в котором он был создан. Сам формат задается спецификацией, а его интерпретация производится с помощью стековой виртуальной машины. Фактически, содержимое PDF можно рассматривать как набор инструкций.

Все инструкции выполняются последовательно, каждая извлекает параметры со стека, производит операцию и помещает результат своей работы на стек (если это требуется) и изменяет контекст.

Для извлечения теговой информации разработано и опубликовано в открытом доступе инструментальное средство TaggedPDF[9], которое имитирует стековую машину вывода на печать согласно спецификации формата PDF. Это позволило правильным образом извлечь название класса и вычислить координаты начала и конца тега. Для разработки инструментального средства извлечения теговой информации из помеченных PDF документов использовалась библиотека Apache PDFBox. Данная библиотека предназначена для генерации и извлечения текстовой и графической информации из PDF документов на достаточном уровне для реализации поставленной задачи.

[9] https://github.com/sunveil/tagged_pdf/tree/develop

3.5. Генерация обучающей выборки

С помощью разработанного инструмента в автоматическом режиме из 1200 документов было получено 28000 изображений и аннотаций к ним в формате PASCAL VOC.

3.6. Обучение модели

Для тестирования применимости автоматически сгенерированного обучающего набора данных была выбрана нейронная сеть архитектуры EfficientDet D2 [22]. Данная модель имеет одну из лучших оценок при высокой производительности на наборе данных COCO, что стало определяющим фактором при выборе архитектуры модели.

Обучение модели производилось с помощью фреймворка Object Detection API TensorFlow 2 на двух видеокартах NVIDIA GeForce RTX 2060 Super.

Сгенерированный набор данных был разбит на обучающий (24000) и валидационный (4000). Предобученная модель EfficientDet D2 на наборе данных COCO обучалась 20 эпох (120000 шагов). Каждую эпоху производилась валидация модели и сохранялось ее состояние. В результате для тестирования была выбрана модель, полученная после 18 эпохи.

3.7. Тестирование модели

Для тестирования модели использовался набор реальных данных мощностью 277 изображений опубликованный в работе [21]. Для сравнения в табл. 1 приведены результаты тестирования (с применением постобрабоки (**Post**)) четырех моделей, обученных на разных наборах данных из [21]: **PLN** – набор данных научных статей PupLayNet, 125 тысяч изображений, **GEN** – набор сгенерированных данных, 18 тысяч изображений, **NPA-small** – набор реальных данных, 100 изображений, **NPA-big** – набор реальных данных, 500 изображений. Для оценки качества использовались метрика PASCAL VOC – (AP) = 0.5.

Табл. 1. Результаты сравнительной оценки
Table 1. Comparation results

Метод	Текст	Таблица	Изображение	Итого
PLN+GEN+NPA-big	0.810	0.937	0.696	0.820
PLN+GEN+NPA-big + Post	**0.840**	**0.968**	0.755	**0.855**
PLN+GEN+NPA-small	0.502	0.824	0.336	0.559
PLN+GEN+NPA-small + Post	0.652	0.888	0.448	0.663
PLN+NPA-small	0.489	0.846	0.346	0.565
PLN	0.045	0.065	0.004	0.039
TaggedPDF	*0.032*	*0.876*	***0.763***	*0.557*

Результаты тестирования, приведенные в Табл. 1. показывают, что модель, обученная только на автоматически аннотированных данных, показывает лучший результат на классе *изображение*. Сопоставимый результат с моделью (PLN+GEN+NPA-small + Post), дообученной на трех разных наборах данных с использованием постобработки достигается для класса *таблица*.

Неудовлетворительная оценка получена для класса *текст*. Это объясняется тем, что Microsoft Word предоставляет большой набор инструментов для редактирования текста. Он может быть отредактирован различными способами при полной визуальной идентичности. При этом семантически единый текстовый блок для эксперта во внутреннем представлении документа будет разбит на множество частей с разными тегами. Это порождает несоответствие экспертной разметки с автоматической, из-за чего данный класс распознается плохо.

С другой стороны, для редактирования таблиц и вставки рисунков в документ представлен ограниченный набор инструментов, что порождает более точное соответствие внутреннего представления с визуальным. Из-за этого автоматическая разметка в основном соответствует экспертной.

4. Заключение

Предложенный метод автоматической разметки данных для решения задачи сегментации изображений документов можно считать эффективным только для классов с высокой согласованностью экспертов. К таким классам относятся таблица и изображение, а из не рассмотренных в данной работе – формула, колонтитул, номер страницы. В других случаях из-за неоднозначности внутреннего представления форматирования документа с его визуальным восприятием данные размечаются некорректно.

Основываясь на полученных результатах, можно сделать вывод, что предложенный метод имеет ряд ограничений, связанных с разнообразием инструментов форматирования Microsoft Word. Для устранения указанных ограничений возможно использование в качестве исходных данных файлы в формате LaTex. Он имеет более строгий набор инструментов форматирования документов, что должно позитивно повлиять на устранение неоднозначности между внутренним представлением документа и его визуальным представлением. В качестве альтернативы можно использовать синтетически сгенерированные документы в качестве исходных данных для аннотирования обучающей выборки.

Список литературы / References

[1] Lee E., Park J. et al. Deep-learning and graph-based approach to table structure recognition. Multimedia Tools and Applications, vol. 81, issue 4, 2022, pp. 5827-5848.

[2] Le V.P., Nayef N. Text and non-text segmentation based on connected component features. In Proc. of the 13th International Conference on Document Analysis and Recognition (ICDAR), 2015, pp. 1096-1100.

[3] Wong K.Y., Casey R.G., Wahl F.M. Document analysis system. IBM Journal of Research and Development, vol. 26, issue 6, 1982, pp. 647-656.

[4] Okun O., Doermann D., Pietikainen M. Page segmentation and zone classification: The state of the art. Technical Report LAMP-TR-036, CAR-TR-927, CS-TR-4079. University of Maryland, 1999, 38 p.

[5] Moll M.A., Baird H.S., An C. Truthing for pixel-accurate segmentation. In Proc. of the Eighth IAPR International Workshop on Document Analysis Systems, 2008, pp. 379-385.

[6] Moll M.A., Baird H.S. Segmentation-based retrieval of document images from diverse collections. In Proc. of the IS&T/SPIE 20th Annual Symposium on Electronic Imaging, 2008, 8 p.

[7] Fletcher L.A., Kasturi R. A robust algorithm for text string separation from mixed text/graphics images. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, issue 6, 1988, pp. 910-918, 1988.

[8] Tombre K., Tabbone S., et al. Text/graphics separation revisited. Lecture Notes in Computer Science, vol. 2423, 2002, pp. 200–211.

[9] Bukhari S.S., Al Azawi M.I.A. et al. Document image segmentation using discriminative learning over connected components. In Proc. of the 9th IAPR International Workshop on Document Analysis Systems, 2010, pp. 183-190.

[10] Kang L., Kumar J. et al. Convolutional neural networks for document image classification. In Proc. of the 22nd International Conference on Pattern Recognition, 2014, pp. 3168-3172.

[11] Harley A.W., Ufkes A., Derpanis K.G. Evaluation of deep convolutional nets for document image

classification and retrieval. In Proc. of the 2015 13th International Conference on Document Analysis and Recognition (ICDAR), 2015, pp. 991-995.
[12] Oliveira D.A.B., Viana M.P. Fast CNN-based document layout analysis. In Proc. of the IEEE International Conference on Computer Vision, 2017, pp. 1173-1180.
[13] Vincent N., Ogier J.M. Shall deep learning be the mandatory future of document analysis problems? Pattern Recognition, vol. 86, 2019, pp. 281-289.
[14] Clausner C., Antonacopoulos A., Pletschacher S. ICDAR2017 Competition on Recognition of Documents with Complex Layouts – RDCL2017. In Proc. of the 14th International Conference on Document Analysis and Recognition (ICDAR), 2017, pp. 1404-1410.
[15] Clausner C., Antonacopoulos A., Pletschacher S. ICDAR2019 Competition on Recognition of Documents with Complex Layouts - RDCL2019. In Proc. of the 15th International Conference on Document Analysis and Recognition (ICDAR), 2019, pp. 1521-1526.
[16] Gao L., Huang Y. et al. ICDAR 2019 Competition on Table Detection and Recognition (cTDaR). In Proc. of the 15th International Conference on Document Analysis and Recognition (ICDAR), 2021, pp. 1510-1515.
[17] Lopes C.A.M. Junior, das Neves R.B. Junior et al. ICDAR 2021 Competition on Components Segmentation Task of Document Photos. Lecture Notes in Computer Science, vol. 12824, 2021, pp. 678-692.
[18] Anitei D., Sánchez J.A. et al. ICDAR 2021 Competition on Mathematical Formula Detection. International Conference on Document Analysis and Recognition. Lecture Notes in Computer Science, vol. 12824, 2021, pp. 783-795.
[19] Yepes A. J., Zhong P., Burdick D.. ICDAR 2021 Competition on Scientific Literature Parsing. Lecture Notes in Computer Science, vol. 12824, 2021, pp. 605-617.
[20] Adams T., Namysl M. et al, Benchmarking table recognition performance on biomedical literature on neurological disorders. Bioinformatics, vol. 38, issue 6, 2022, pp. 1624-1630,
[21] Беляева О.В., Перминов А.И., Козлов И.С. Использование синтетических данных для тонкой настройки моделей сегментации документов. Труды ИСП РАН, том 32, вып. 4, 2020 г., стр. 189–202 / Belyaeva O.V., Perminov A.I., Kozlov I.S. Synthetic data usage for document segmentation models fine-tuning. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 4, 2020. pp. 189–202 (in Russian). DOI: 10.15514/ISPRAS–2020–32(4)–14.
[22] Tan M., Pang R., Le Q.V. EfficientDet: Scalable and efficient object detection. In Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. pp. 10778-10787..

Информация об авторах / Information about authors

Андрей Анатольевич МИХАЙЛОВ – кандидат технических наук, старший научный сотрудник лаборатории комплексных информационных систем ИДСТУ СО РАН, научный сотрудник ИСП РАН. Сфера научных интересов: искусственный интеллект, большие данные, анализ документов.

Andrey Anatolievitch MIKHAYLOV – Candidate of Technical Sciences, Senior Researcher in the Laborotory of Complex Information Systems at IDSTU SB RAS, Researcher at ISP RAS. Research interests: artificial intelligence, big data, document analisys.

DOI: 10.15514/ISPRAS-2022-34(6)-11

Modelling Interrelationship between Diseases with Communicating Stream X-Machines

D. Jayatilake, ORCID: 0000-0002-3377-7129 <dilshan.jayatilake@uwe.ac.uk>
K. Phung, ORCID: 0000-0002-9341-2033 <khoa.phung@uwe.ac.uk>
E. Ogunshile, ORCID: 0000-0002-6276-188X <emmanuel.ogunshile@uwe.ac.uk>
M. Aydin, ORCID: 0000-0002-4890-5648 <mehmet.aydin@uwe.ac.uk>

University of the West of England
Coldharbour ln, Bristol, BS16 1QY, UK

Abstract. The world is moving towards alternative medicine and behavioural alteration for treating, managing, and preventing chronical diseases. In the last few decades, diagrammatical models have been extensively used to describe and understand the behaviour of biological organisms (biological agents) due to their simplicity and comprehensiveness. However, these models can only offer a static picture of the corresponding biological systems with limited scalability. As a result, there is an increasing demand to integrate formalism into more dynamic forms that can be more scalable and can capture complex time-dependent processes. In this paper, we introduce a generic disease model called Communicating Stream X-Machine Disease Model (CSXMDM), which has been developed based on X-Machine and Communicating X-Machine theories. We conducted an experiment on modelling an actual disease using a case study of Type II Diabetes. The results of the experiment demonstrate that the proposed CSXMDM is capable of modelling chronic diseases.

Keywords. Communicating Stream X-Machine; Stream X-Machine; Type II Diabetes; modelling; formal method; cardiovascular disease

For citation: Jayatilake D., Phung K., Ogunshile E., Aydin M. Modelling Interrelationship between Diseases with Communicating Stream X-Machines. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 147-164. DOI: 10.15514/ISPRAS-2022-34(6)-11

Моделирование взаимосвязи между заболеваниями с помощью взаимодействующих потоковых X-машин

Д. Джаятилаке, ORCID: 0000-0002-3377-7129 <dilshan.jayatilake@uwe.ac.uk>
К. Фунг, ORCID: 0000-0002-9341-2033 <khoa.phung@uwe.ac.uk>
Э. Огуншиле, ORCID: 0000-0002-6276-188X <emmanuel.ogunshile@uwe.ac.uk>
М. Айдин, ORCID: 0000-0002-4890-5648 <mehmet.aydin@uwe.ac.uk>

Университет Западной Англии,
Великобритания, BS16 1QY, Бристоль, Колдхарбор-лейн

Аннотация. Мир движется к альтернативной медицине и изменению методов лечения, контроля и профилактики хронических заболеваний. В последние несколько десятилетий диаграммные модели широко использовались для описания и понимания поведения биологических организмов (биологических агентов) благодаря их простоте и полноте. Однако эти модели могут предложить только статическую картину соответствующих биологических систем с ограниченной масштабируемостью. В результате растет спрос на интеграцию формализма в более динамичные формы, которые могут быть более масштабируемыми и могут охватывать сложные процессы, зависящие от времени. В этой статье мы представляем общую модель на основе теорий X-машин и взаимодействующих X-машин. Мы провели эксперимент по моделированию реального заболевания на примере диабета II типа. Результаты

эксперимента демонстрируют, что предложенный метод способен моделировать хронические заболевания.

Ключевые слова: взаимодействующие потоковые Х-машины; потоковая Х-машина; диабет второго типа; моделирование; формальный метод; сердечно-сосудистые заболевания

Для цитирования: Джаятилаке Д., Фунг К., Огуншиле Э., Айдын М. Моделирование взаимосвязи между заболеваниями с помощью взаимодействующих потоковых Х-машин. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 147-164. DOI: 10.15514/ISPRAS-2022-34(6)-11

1. Introduction

Despite the development in medical domain still the chronical disease (e.g., osteoporosis (OP), gout, rheumatoid arthritis (RA), type 2 diabetes (T2D), Alzheimer's disease (AD)) prevention, mitigation and managing patients who are suffering from chronicle diseases has limited to a single dimension treatment pattern. Still there is a very limited understanding about the etiology and the mechanism of the diseases [1, 2]. It is important to understand the mechanism (behaviour of the disease). Therefore, precise modelling of the diseases will improve treating patients with chronicle diseases and in prevention and management. On the other hand, having a better understanding about the mechanisms of the disease will help in directing the launches of proper clinical trials, target for effective lifestyle interventions and pharmacological innovations.

Human body consists of closely connected subsistence (example: cardiovascular system, digestive system etc.;). Therefore, the human body can be identified as a complex system where continuant subsistence work together in order to keep a human alive [3, 4]. As in any other complex system, there are some undesired states where the human body can reach. in our research we consider diseases as undesired states (similar to the error states in complex systems) [3]. Due to the close coupling among those systems and the complex relationships has made it challenging to precisely model the human body. Because of the complex interrelationships among the subsystems of the body, we believe that diseases mimic the same.

Shaikh F. Hossain et al. [5], has discussed the inter relation between diseases based on the biomarkers. Based on 432 biomarkers mapping in to 18 diseases, authors have been able to demonstrate the chemical and the alliance between the disease. Furthermore, literature provide evidence of the relationship between multiple chronic diseases [6], and effects on the human behaviour. These further discuss the importance of having clear and precise understanding for treating patients with such conditions.

With the evolving research and development in the fields of metabolomics, proteomics, and nutrigenomics are shifting the perspective of nutrition and diet management to be considered as medicine and concept of nutrition therapeutics [7] are actively used in experimental patient management [8]. Due to the demand of utmost accuracy and precision in this domain, to successfully implement in practice has been a challenge.

Literature provides evidence of employing formal methods in modelling critical systems where they have demanded very high level of accuracy [9-11]. In our previous research formal methods has been successfully used modelling diseases [3].

Furthermore, we discuss the significance of the accuracy modelling diseases and the importance of having the ability to communicate between diseases to demonstrate the impact of one disease has towards the other disease(s). During this research we have decided to use well known formal method namely, Communicating Stream X-Machines (CSXM), an extension of Stream X-Machine (SXM) to model diseases and the relationship among them.

This paper is structured as follows. Section 2 discusses the related work. Section 3 and Section 4 outline the proposed disease model with the employment of Stream X-Machine and Communicating Stream X-Machine, respectively. Section 5 demonstrates the experiment and its results. Section 6 consists of the discussions about the experimental results and the proposed disease model. Section 7 concludes the paper and outlines future research directions.

2. Related Work

2.1. Finite State Machine, X-Machine, and Stream X-Machine

X-Machine (XM) [12] is an extension of the finite state machine (FSM), where the FSM is empowered with a data structure and a processing function(s). As explained, XM is very similar to an FSM with the difference of having a data sent X (type of the machine), where finite set of processing functions Φ operates on the X ($\varphi: X \rightarrow X \mid \varphi \in \Phi$) and inbuilt memory (M) where thy transitions can alter the memory. In the FSM diagram each transition (arch) is ladled with a processing function of φ ($\varphi \mid \varphi \in \Phi$). Because of the generalisation of the type X, X-Machine has proven that it is capability of modelling any type of system accurately and also in a very generic manner where extension and/or modification is possible [11-14, 16].

Another class of X-Machine has been introduced to specify and model software namely Stream X-Machines (SXM), which has the ability to define and validate data types and the functionalities of a software systems [17]. The definition of SXM is as follows.

$$\mathbf{Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)}$$

where:

- Σ = input alphabet;
- Γ = output alphabet;
- Q = finite set of states;
- M = memory (possibly infinite);
- Φ = the type of the machine X, where a set of partial functions $\varphi(\varphi|\varphi \in \Phi)$ which transforms input and a memory to an output and a possibly to a different memory state: $\varphi: \Sigma \times M \rightarrow \Gamma \times M$;
- F = Next state partial function which drives the machine state: $F: Q \times \Phi \rightarrow Q$;
- q_0, m_0 = initial state and the initial memory of the machine respectively.

For the above illustrated SXM, the associated FSM is **Az = (Φ, Q, F, q_0)**. This Az is also known as associated finite state automaton (FA). Fig 1 illustrates the state-transition diagram of a three-state Stream X-Machine.

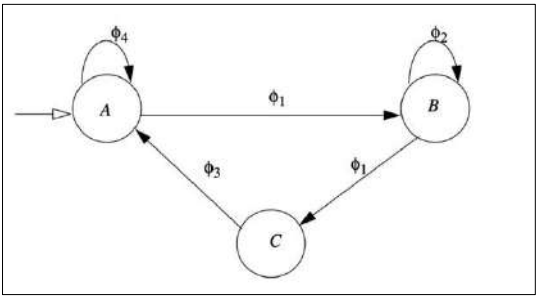

Fig. 1. A three-state Stream X-Machine

2.2. Communicating Stream X-Machine (CSXM)

There are number of different approaches can be found for CSXM in the literature [18-20]. In this research, we discuss the standard approach and an alternative approach which leads to the development of SXM – Communicating Component (SXMCM).

2.2.1 Communicating Stream X-Machine standard approach

Communicating Stream X-Machine which consists of n number of constituent components can be identified [18, 21] as a tuple of: $((XM_i)_{i=1..n}, CM, C_0)$ where:

- XM_i is the i^{th} X-Machine of the system;

- CM is a n×n matrix, namely the Communication Matrix;
- C_0 is the initial communication matrix.

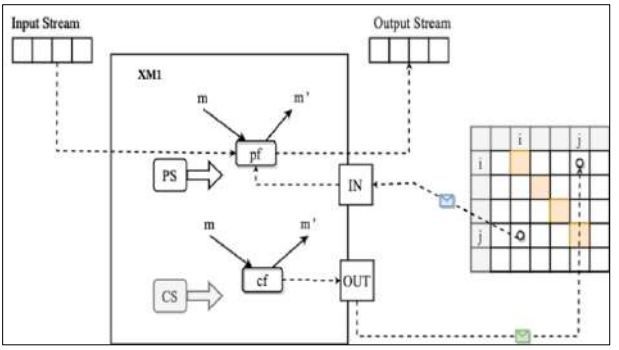

Fig 2. CSXM illustration with Communication Matrix

In this approach, there is a difference from the usual standalone SXM definition because the XM_i has defined IN and OUT ports defined for communication (**Ошибка! Источник ссылки не найден.**) [21]. As illustrated in **Ошибка! Источник ссылки не найден.**, both IN and OUT ports are connected to a communication matrix (CM). CM facilitates the communication between the SXMs. CM cells holds me messages from the SXMs. For example, cell (i, j) has the message that was sent from SXM_i to SXM_j. The type of the message can be any type which is defined in the memory. λ stands for empty or no message. Communication is initiated only from a communicating state, which accepts empty symbol ε as an input and produces ε as the output without effecting the memory. Communicating function only either reads an element from the associated CM and writes in the IN port and assign the value of λ to the cell in which the function read from or writes an element from the OUT port to CM if the cell has no message (cell has the value of λ) [18, 21]: **cf(ε, m, in, out, c) = (ε, m, in', out', c')** where m ∈ M; in, in' ∈ IN; out, out' ∈ OUT; and c, c' ∈ CM.

If the communication function is not applicable, it waits. The processing function only affects the ports (IN & OUT) but do not affect the communication matrix) [18, 21]: **pf(σ, m, in, out) = (γ, m', in', out')** where: σ ∈ Σ; m, m' ∈ M; in, in' ∈ IN; out, out' ∈ OUT; and γ ∈ Γ.

2.2.2 Communicating Stream X-Machine alternative approach

Standard CSXM suffers from inability to conceive as an independent component. Due to this limitation it has to be started from the beginning in order to add a new component to the arrangement. Furthermore, this limits the ability of the components in this arrangement to act as a standalone SXM or be a part in other arrangements.

Petros Kefalas et al. [21] proposed an X-Machine type (MT) without the initial state or memory as an alternative approach to overcome the previously identified issues. MT is defined as follows:

$$\mathbf{MT = (\Sigma, \Gamma, Q, M, \Phi, F).}$$

An X-Machine is constructed through application of operators:

$$\mathbf{OP\ inst:\ MT_i \times (q_{0i}, m_{0i}) \rightarrow M_i, \forall\ q_{0i} \in Q, m_{0i} \in M.}$$

That creates the instance of MT: $\mathbf{M = MT\ OP\ inst\ (q_0, m_0).}$

Communicating X-Machine Component is defined as:

$$\mathbf{XMC_i = (\Sigma_i, \Gamma_i, Q_i, M_i, \Phi_i, F_i)\ OP\ inst\ (q_{0i}, m_{0i})\ OP\ comm\ (IS_i, OS_i, \Phi_i, S_i, \Phi_{OSi})}$$

where:

- IS_i is a tuple with *n* input streams, which contain the source of the message where it has been generated (CSXM which send the message) (in i is the standard input source of $CSXM_i$): $IS_i = (is_1, is_2, ...is_i, ..., is_n)$, and $is_j = \varepsilon$ (if no communication is required) or $is_j \subseteq \Sigma_j$;
- OS_i is a tuple defined in correspondence with *n* output streams, which consists of the destination

of the *n* message that used in generating the message. destination of $CSXM_i$): $OS_i = (os_1, os_2, ..., os_i, ..., os_n)$, and $os_j = \varepsilon$ (if no communication is required) or $os_j \subseteq \Sigma_j$;

- ΦIS_i is an association of function $\varphi_i \in \Phi_i$ and the input stream IS_i, ΦIS_i: $\varphi_i \leftrightarrow IS_i$;
- ΦOS_i is an association of function $\varphi_i \in \Phi_i$ and the output stream OSi, ΦOS_i: $\varphi_i \leftrightarrow OS_i$.

Applying the operator $\mathbf{OP_{comm}: XM_i \times (IS_i, OS_i, \Phi_{ISi}, \Phi_{OSi}) \rightarrow CXMC_i}$ has a result a Communicating X-machine component $CXMC_i$ as a tuple:

$$\mathbf{SXMC_i = (\Sigma_i, \Gamma_i, Q_i, M_i, \Phi_{Ci}, F_i, q_0, m_0, IS_i, OS_i)}$$

where:

- $\mathbf{\Phi_{Ci}}$ is a set of partial functions that read from standard input or any other input stream and write to standard output or any other output stream. Such set will consist of 4 different sets of function: $\mathbf{\Phi_{Ci} = SISO_i \cup SIOS_i \cup ISSO_i \cup ISOS_i}$, where:
  - $SISO_i$ is a set of functions φ which reads and writes to standard input (is_i) and standard output (os_i) streams respectively. $SISO_i = \{(is_i, m) \rightarrow (os_i, m) \mid \varphi i = (\sigma, m) \rightarrow (\gamma, m) \in \Phi_i \wedge \varphi_i \notin dom(IS_i) \wedge \varphi_i \notin dom(OS_i)\}$;
  - $SIOS_i$ – a set of functions φ which read and writes to standard input (is_i) and j^{th} output (os_j) streams respectively. $SIOS_i = \{(is_i, m) \rightarrow (os_j, m) \mid \varphi_i = (\sigma, m) \rightarrow (\gamma, m) \in \Phi_i \wedge \varphi_i \notin dom(IS_i) \wedge (\varphi_i \rightarrow os_j) \in OS_i\}$;
  - $ISSO_i$ is a set of functions φ which reads and writes to j^{th} input (is_j) and standard output (os_i) streams. $ISSO_i = \{(is_j, m) \rightarrow (os_i, m) \mid \varphi_i = (\sigma, m) \rightarrow (\gamma, m) \in \Phi_i \wedge (\varphi_i \rightarrow is_j) \in IS_i \wedge \varphi_i \notin dom(OS_i)\}$;
  - $ISOS_i$ is a set of functions φ which read and writes to j^{th} input (is_j) and k^{th} output (os_k) streams. $ISOS_i = \{(is_j, m) \rightarrow (os_k, m) \mid \varphi_i = (\sigma, m) \rightarrow (\gamma, m) \in \Phi_i \wedge (\varphi_i \rightarrow is_j) \in IS_i \wedge (\varphi_i \rightarrow os_k) \in OS_i\}$.

Communicating X-Machine is defined as a tuple of *n* XMC as:

$$\mathbf{CXM = (XMC1, XMC2, ..., XMCn)} \text{ with}$$

- $\Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_n = (os_{11} \cup os_{12} \cup ... \cup os_{1n}) \cup ... \cup (os_{n1} \cup os_{n2} \cup ... \cup os_{nn})$;
- $\Gamma_1 \cup \Gamma_2 \cup ... \cup \Gamma_n = (is_{11} \cup is_{12} \cup ... \cup is_{1n}) \cup ... \cup (is_{n1} \cup is_{n2} \cup ... \cup is_{nn})$.

In this approach, the CM has been replaced by several input streams associated with each Stream X-Machine component. Through replacement of the communication state and the functions with functions that belongs to type Φ and providing the ability to read and write to different input and output streams has improve the usability of the models (fig. 3) [21].

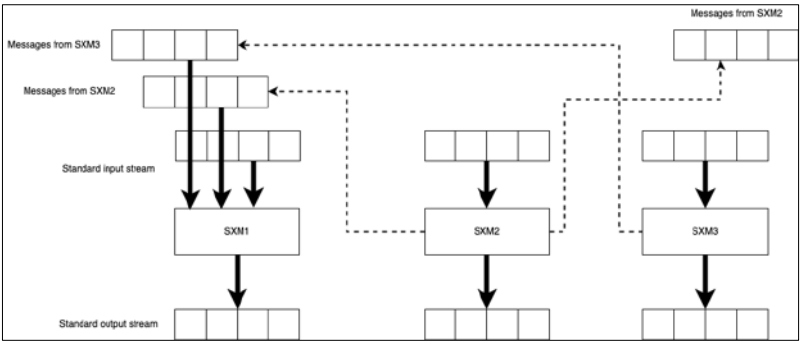

Fig. 3. Three Communicating Stream X-Machines

2.3 Disease modelling with formal methods

Biological systems such as human body consists of tightly connected components (organs) that change their actions and behaviours over the time and their interactions with exposure to external factors (e.g., nutrients, viruses, bacteria). In addressing such challenges, literature reveals that, there are few formal approaches has been occupied namely, Boolean Networks (BN) and its extensions

(i.e. Qualitative Networks (QN), Gene Regulatory Networks (GRN) [22-24], Petri Nets (PR), Cellular Automata (CA), population P systems (PPS), etc.

Table 1. Review of formal methods

Method	**Strengths**	**Weaknesses**	**Example**
BN/QN [15, 16, 18]	Correctly defined binary states.	Due to lack of state space in BN (Active and Inactive), harder to model complex state models such as Biological Systems. Even though, QN discrete variable values, and the dependencies of those values are expressed with algebraic functions instead of Boolean functions, complex relations mapping is not possible.	BN/QC has been employed in Schematic view of signal transduction in the pancreatic cancer model [28].
PR [29, 30]	Has the capability of comprehensive modelling and analysing facility for distributed and concurrent systems.	The ability of providing the concurrency the complexity of the model increases significantly. PR suffers from the inability to test in unbounded places.	PR has been used to model enzymatic reactions in Metabolic pathways [31].
CA [25]	Capable of modelling interactive components. CA is a powerful way of defining agent-based system due to the simplicity.	CA presents challenges in modelling non-trivial systems due to lack of data representation. CA suffers from state and input symbol explosion	CA has been occupied in modelling virus behaviour (e.g. Human Immunodeficiency Virus (HIV)) [32].
PPS [26]	PPS provides a robust mechanism to introduce new nodes, remove nodes, and change the behaviour of the defined nodes, which helps to map the disease behaviour.	PPS lacks the ability to represent internal states and individual behaviour of the nodes.	PPS has been employed in modelling species behaviour, especially when it eloper is introduced to the system (e.g. behaviour on school of fish upon a predator) [33].

As evaluated in Table 1, current formal specification models have not been able to comprehensively model biological systems. Even we consider the human body as a biological system, each disease has its own behaviour and attributes which are interconnected yet different. Considering the BN and QN would have the capability of modelling the binary state of biological systems yet suffers from lack of state space [15, 16, 18]. When considering diseases, there are multiple states for most of the diseases. For instance, diabetes has few inner states such as Pre-metabolic syndrome, metabolic syndrome pre-diabetic and Type II Diabetes. Furthermore, this will not facilitate the interconnectivity with expected efficiency and the accuracy. PR would mostly address the issues that has been identified with BN and QN. PR provides a comprehensive modelling and analysing facilities for distributed agents (systems), which would be ideal for modelling diseases; hence, the anonymity and the connectivity (in terms of having relationships between diseases) at the same time

[26, 27]. However, the concurrency introduces significant and unmanaged complexity to the system. Diseases act in concurrently in nature. Therefore, this would bring unexpected and unrealistic complexity to the models, which hinders the demanded accuracy. CA addresses the above discussed complexity and has been proven to be used in agent-based systems [25]. However, CA suffers from data representation, whereas disease models should encapsulate and persist their own data for monitoring disease progression. One other aspect of disease modelling is that diseases are expected to be infected and cured (excluding incurable diseases). The models should be easily introduced and removed from the system. PPR provides the capability of adding, deleting, and changing nodes [26]. However, PPR fails to introduce the initial state and modelling individual behaviour. Therefore, it can be inferred that the reviewed formal methods in Table 1 have not provided a comprehensive mechanism to model diseases and their behaviours completely with the expected accuracy.

There is an evidence of research that they have used formal methods in order to model biological systems, in specific they have employed Communicating X-Machines in order to model the bio-systems where it has used the ability of representing the interrelationship and the communication aspects of the biological systems [26], while preserving the ability of data persistence with flexibility. To the best of our knowledge this is the first attempt that formal verification methods has been occupied in order to model a chronic disease and model the inter-relationship among diseases.

2.4 Strengths and weaknesses of existing models

Literature provides evidence of occupying formal methods to model biological systems. However, most of these approaches have not been able to provide the much needed 100% accuracy level in modelling such systems. Hence the medical domain is one of the most critical domains which the cost of a mistake could be a life [15, 27]. In this paper, we have considered few approaches and their strengths and weaknesses (table 1).

As illustrated above, mathematical models present significant challenges in modelling diseases due to the highly complex interrelationship(s) among the large state space and disease models themselves with other disease models. Due to this highly complex and heavily interconnected model behaviour, most of the mathematical models are being challenged, and/or the models become unmanageably complex [15].

Due to the criticality of the domain in nature, to the best of the authors' knowledge, it is expected that the formal modelling approaches would bring more efficiency and accuracy for disease modelling in general, especially X-Machine models would be more realistic due to their capability in modelling critical systems [34], where interrelationships among the models can be more accurately handled. Therefore, in this research we have selected X-Machines in general to model diseases, particularly Stream X-Machine is used to model Type II Diabetes to demonstrate the capability of modelling diseases.

2.5 State of the art in biological system modelling and gap analysis

Literature provides many evidence of employing mathematical models to precisely understand the disease behavior [15, 16, 18, 25, 26, 28, 29, 30, 35-38]. These have been always focused on one disease and also disease as a whole and its impact in the perspective of spread, implication the medical sector and/or holistic implication on the society. In our previous publication we demonstrated how formal speciation can be employed in modelling diseases. To the best of our knowledge this the first attempt that, employing formal methods in general especially classes of X-Machine in modelling the behaviour of diseases with representing its impact on the other diseases.

3. Stream X-Machine Disease Model (SXMDM)

We have introduced the SXMDM with the capability of adopting to any diseases, which gives the capability of mimicking the disease behaviour. SXMDM proposed in our previous research has the capability of:

- Describe diseases and their stages (Demonstration/research purposes for the medical professionals).
- Simulate the progression of the diseases (Positive and negative).
- Determine the stages of the diseases at the time of the patient is being diagnosed.
- Determine the stages with the change of the symptoms.
- Understand the progression of the diseases.
- Monitoring the progression of the patient with past information (Symptoms and data).

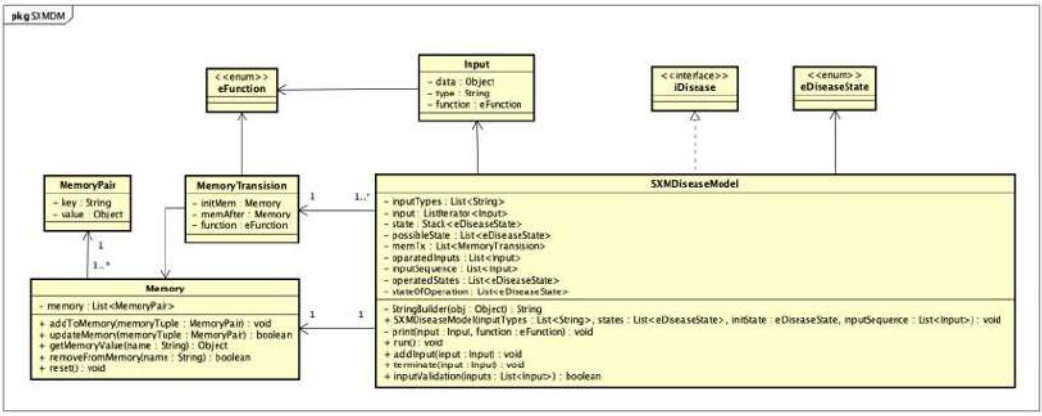

Fig 4. Class diagram of SXMDM

The class diagram of the proposed SXM disease model is illustrated in fig 4.

- XMachineDisease: this class contains the generic X-Machine model which is implemented iDiseaseType interface.
- iDiseaseType: this interface provides the necessary functionalities to describe the Type II Diabetic medical condition.
- MemoryPair: this class provides the unit model which holds the memory units.
- eFunction: this special class of constant data (enum) class provides all the processing functions that involves in the defining disease type SXM.
- MemoryTransition: this class contains the memory transition based on the processing function (eFunction).
- Input: this class provides the input unit which holds in the input sequence of the SXM.
- eDiseaseState: this special class of constant data (enum) provides all the states of the disease.

4. Communicating Stream X-Machine Disease Model (CSXMDM)

CSXMDM is an extension of SXMDM proposed SXDM in our previous research [3]. During our last research we have discussed that, similar to the organs (subsystems) of the body work together for the purpose of wellbeing of the human, disease has the similar characteristic of having inter-relationship among them.

New extended CSXMDM has address the about shortcoming of the SXMDM and given the capability of communicating with other disease models. The CSXMDM has been designed as a pluggable component to SXMDM. In order to facilitate preciously defined SXMDM has been slightly modified. CSXMDM is focusing on modelling the diseases and their inter-relationship(s). The CSXMDM we proposed takes a state-based communication approach extended from modular approach. Instead of considering the CSXM as a whole, in the proposed CSXMDM has communicates among the other models with preserving the communication-initiated state, during and after communication.

4.1 Changes in SXMDM

We have changed the SXMDM discussed in Section 3 to facilitate the communication. Hence, we are more interested in the state level communication, sates have given the ability to define its readiness to communicate at the state level (fig. 5).

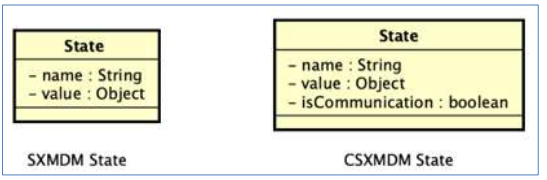

Fig. 5. Implementation of State in SXMDM & CSXMDM

4.2 SXM – Communicating Component (SXMCM)

In order to support Communications between the SXMDMs, we are proposing SXMCM. The implementation of the SXMCM is illustrated below (fig. 6).

- **CommunicatorBase**: This class provides the basic infrastructure to establish communication between the SXMDMs. This class acts as the common mediator of the communication.
- **Communicator**: This abstract class provides interface for establishing the communication in the SXMDMs (provide that the SXMDM is using the new version of State discussed previously).
- **Message**: This provides the unit model for messages.
- **MessageUnit**: This class facilitates the message buffer for the Communicator.

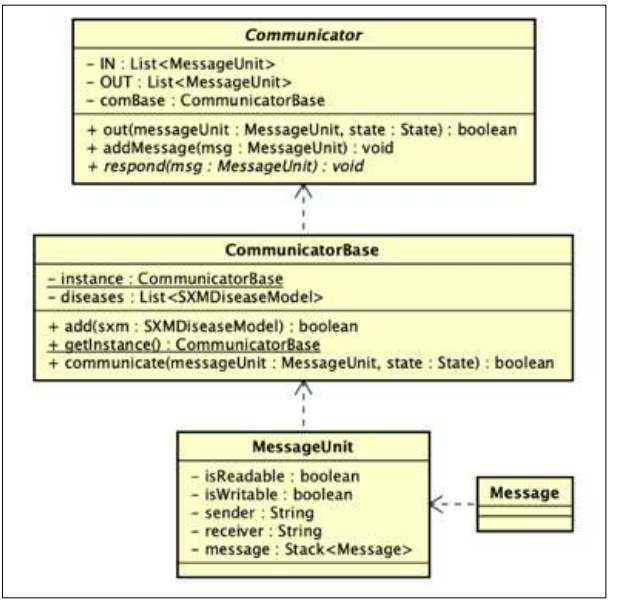

Fig. 6. Implementation of SXMCM

4.3 Discussion of implementation

SXMCM consists of two main packages. *com.communication.communicationCmp* consists of basic implementation of the communication plugin. This consists of the *CommunicationBase* and *Communicator*. *com.communication.util* provides the utilities for *Communicator* and *CommunicationBase*. This consists of the *Message* and *MessageUnit*.

5. Case Study

We have employed the same case study from our last research with the addition of cardiovascular disease (CVD) model (fig. 7).

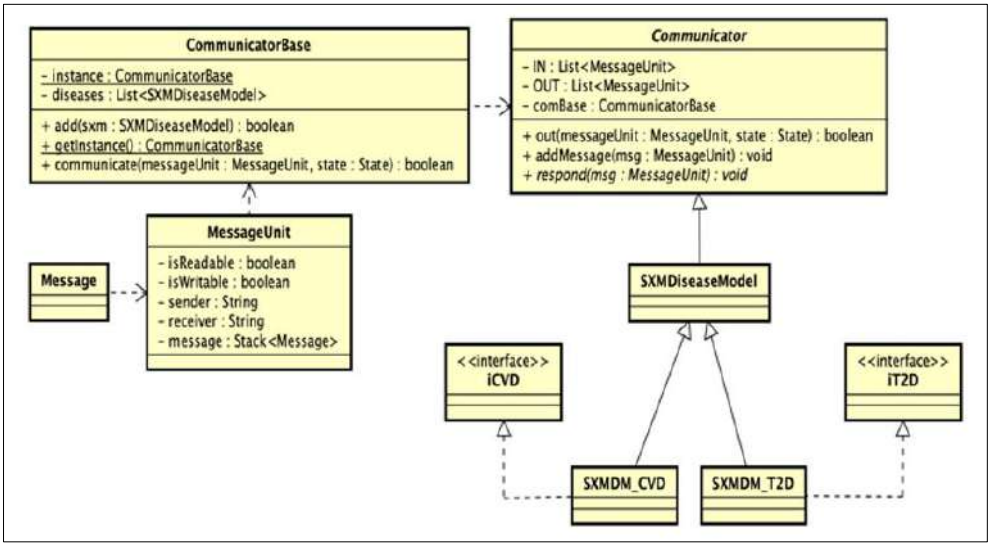

Fig. 7. Implementation of the Type II Diabetes disease model with CVD rick model

5.1 Type II Diabetes Model

To demonstrate the validity of the Stream X-Machine Disease Model (SXMDM), we consider the implementation of Type II Diabetes Model (TTDM). To test the correctness of the TTDM, we have to adopt the TTDM to SXDM. In our proposed TTDM, we have assumed that the symptom collection is done elsewhere, and the data (Symptom's information) is fed into the model.

5.2 Type II Diabetes and Symptoms Evaluation

There are clearly defined four stages of diabetes namely, pre metabolic syndrome, metabolic syndrome, pre diabetic and type II diabetic. Depending on the time of the diagnosis a patient can be in any of the above mentioned four stages. With the time, a patient can progress his/her diabetic stage positively or negatively.

The initial state of the disease (diabetic) is defined by the number of symptoms that the patient is presenting at the time of the diagnosis [13]. The symptom evaluation criteria to define the stage of the diabetic is as follows.

If a patient is presenting with:

- A waist circumference off:
  - 94 centimetres and above for European men or 90 centimetres an above for South Asian men.
  - 80 centimetres or more for South Asian woman.
- High levels of triglyceride in the blood.
- Low levels of HDL.
- Constant levels of high blood pressure 140 / 90mmHg
- Low levels of insulin response (inability to control blood sugar levels)
- A tendency to develop irritation and swelling off the body tissues in general inflammation.

Table 2. Evaluation criteria

Number of Symptoms	Stage of Diabetes
0	Normal
1-2	Pre-Metabolic Syndrome
2-3	Metabolic Syndrome
4-5	Pre-Diabetes
5 or more	Type II Diabetes

The Type II Diabetic Stage Identification can be described as follows: The TTDM is in the Start State, waiting to process patient's symptoms data. When the patient information is fed in, to start the evaluation process. Then the machine will validate the inputs (Symptom information) and process the data for Type II Diabetic stage evaluation. Assuming that the input data is valid, machine starts the evaluation process. Once the evaluation is completed the machine will determine the Diabatic Stage of the patient. Then machine will wait until recheck occurs to re-evaluate the patient. When the re-evaluation is called, the machine will follow the same process as before (fig. 8).

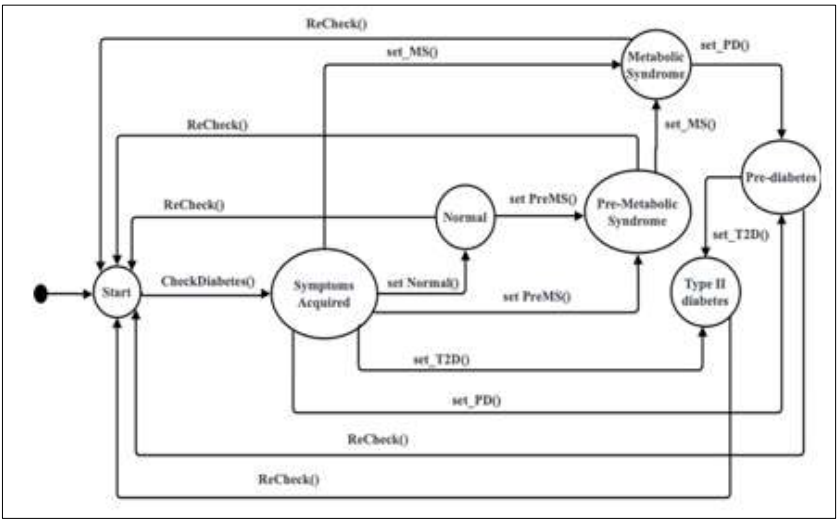

Fig. 8. Type II Diabetes state-transition diagram

We consider first stage of the disease as Start and then leads to the state where the symptoms are acquired and validated (Symptoms acquired). Depending on the symptoms that the particular patient presented at the time of the diagnosis, TTDM defines the stage of the disease. Then the TTDM will wait at the particular level till, recall method is called to submit new (updated symptoms), and repeats the same process.

With the above description, SXMDM can be modelled as a SXM with 6 states. The states are as follows:

- Normal: TTDM waits for the recheck data after acquiring the diabetes state.
- Pre-Metabolic Syndrome: TTDM waits for the recheck data after acquiring the diabetes state.
- Metabolic Syndrome: TTDM waits for the recheck data after acquiring the diabetes state.
- Prediabetes
- Type II Diabetes: TTDM waits for the recheck data after acquiring the diabetes state.
- Symptoms Acquired: TTDM waits for identifying the diabetes stage.
- Start (initial state): TTDM waits for the patient data.

The memory of the TTDM consists of one element with accordance to our implementation.

- Diabetes: a data object of the disease which contains all the necessary to evaluate a patient's diabetic stage.

In the TTDM, we have identified that there are seven functions but drives the model and one internal function.

- set_PreMS(Diabetes diabetes);
- set_MS(Diabetes diabetes);
- set_PD(Diabetes diabetes);
- set_Normal(Diabetes diabetes);
- set_T2D (Diabetes diabetes);
- checkDiabetes(Diabetes diabetes);
- getCurrentState();
- reCheck(Diabetes diabetes).

5.3 Cardiovascular Disease Risk Model (CVDRM)

CVD has a clear relationship with Type II Diabetes (T2D) [40]. The current model of the CVDRM is designed only with the intention of demonstrating the CSXMDM. The model we have developed for the CVD is capable of modelling the level of risk with relevant to T2D. Currant CDVR consists of four risk states and two operational states including start state namely *Start, CheckCVD, Normal, LowRisk, ModerateRisk, HighRisk*. The model consists of six processing functions that drives through the above states namely, *SET_NORMAL (CVD cvd), CHECKCVD (CVD cvd), SET_LOW (CVD cvd), SET_MODERATE (CVD cvd), SET_HIGH (CVD cvd), RECHECK (CVD cvd).*

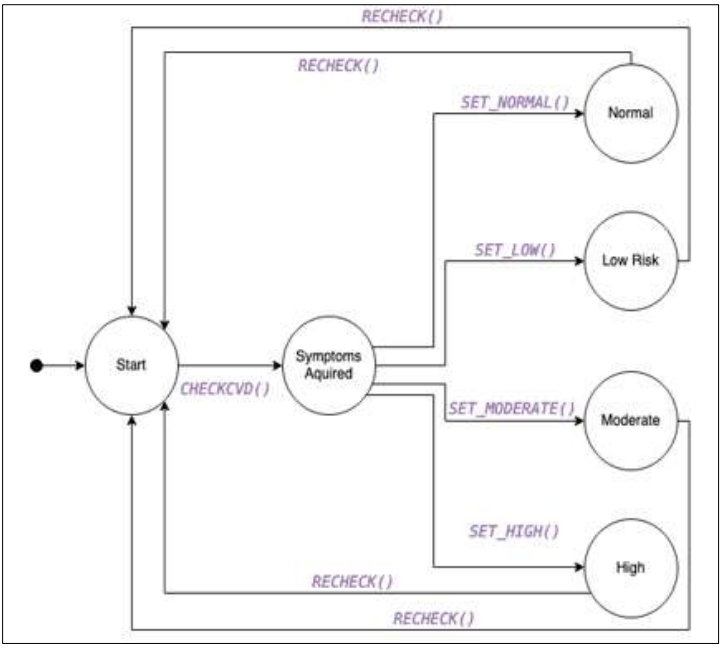

Fig. 9. CVD state-transition diagram

For the demonstration purposes, similar to the above discussed TTDM, CVDRM will evaluates the level risk according to the below mentioned criteria. In the context of this research, we consider if a patient is diagnosed with high levels of HDL, high levels of Triglycerides, higher degree of waist circumference and had been diagnosed with any stage of diabetes will be considered as a symptom. In the context of this research, we assume that if a patient is presenting no symptoms, has no risk

(implied by *Normal*), with one symptom has a low risk, with two symptoms has a moderate risk, with three risks will identified as high risk (fig. 9).

5.4 Demonstration of the TTDM

For demonstration purposes let us assume that a patient is presenting with:

- Level of HDL: Low;
- Level of Triglycerides: Low;
- Degree of waist circumference: Low;
- Stage of diabetes: Non (Normal) / Not diagnosed yet..

While diagnosing towards the patients' CVD check-up. CVDRM will be in its *Start* state. When the symptoms are given, CVDRM invokes *CHECKCVD(CVD cvd)* processing function which will update the memory with the newly received CVD object and proceeds to evaluate the number of symptoms with against the defined evaluation criteria. Based on the evaluation the input sequence will be amended with the relevant processing function to define the next state. In the given scenario the CVDRM will settles in ***Normal*** state hence there are no positive symptoms.

Thereafter, we will assume that the same patient is being diagnosed with high levels of triglyceride in the blood, low levels of HDL, consistent high blood pressure, a waist circumference in normal range and no complaints about body inflammation and high level of insulin response while being checked for diabetes.

By this time, the TTDM initially will be in the *Start* state. When the symptoms are given, TTDM invokes *checkDiabetes(Diabetes diabetes)* processing function, which updates the memory of Diabetes with the data object. Then the TTDM evaluates the Diabetes data object and analyse the number of symptoms present, against the evaluation criteria. Then amends the input sequence with the relevant processing function. In this scenario, the patient is presenting with four positive symptoms and two negative symptoms.

Therefore, the TTDM will evaluates the symptoms and decides the next state as ***Metabolic Syndrome***. In this scenario any stage of Type II diabetes as a communication state, which means in this context dragonising with Metabolic Syndrome improves the risk of CVD.

Therefore, TTDM will be initiating the communication. TTDM will generate the message and add it to its OUT communication-buffer. This will trigger the communication component. The communication component evaluates the message and checks for the intended receives of the message (in this instance it is the CVDRM). Then the risk factors are evaluated. Based on the received message through the communication buffer, the CVDRM will alter its input stream accordingly. In this scenario this will add one risk factor where the risk state will move from ***Normal*** to ***Low Risk***.

Furthermore, the CVDRM has given a second dataset through *RECHECK(CVD cvd)* with all positive symptoms. This will drive the model to ***High-Risk*** state. High Risk being a communicating state, this will be communicated to TTDM. Subsequently, TTDM has taken necessary steps to adjust its own state accordingly (fig. 10).

6. Final Discussion

In the scope of this research, we have further demonstrated the capability of modelling diseases with SXMDM and introduced CSXM to overcome the previously identified limitation of not being able to represent the relationship between diseases [3]. Even though we have overcome one of the limitations that we identified through our previous research, still this is lacking the user interface to be recognised as a tool for generic users and still lacking the capability of representing the inner states (substages of diseases). However, SXMCM, with combining the SXMDM (TTDM & CVDRM) has demonstrated the capability of modelling the diseases and its inter relationship which

would enable medical professionals and the researchers to model diseases with high level of accuracy.

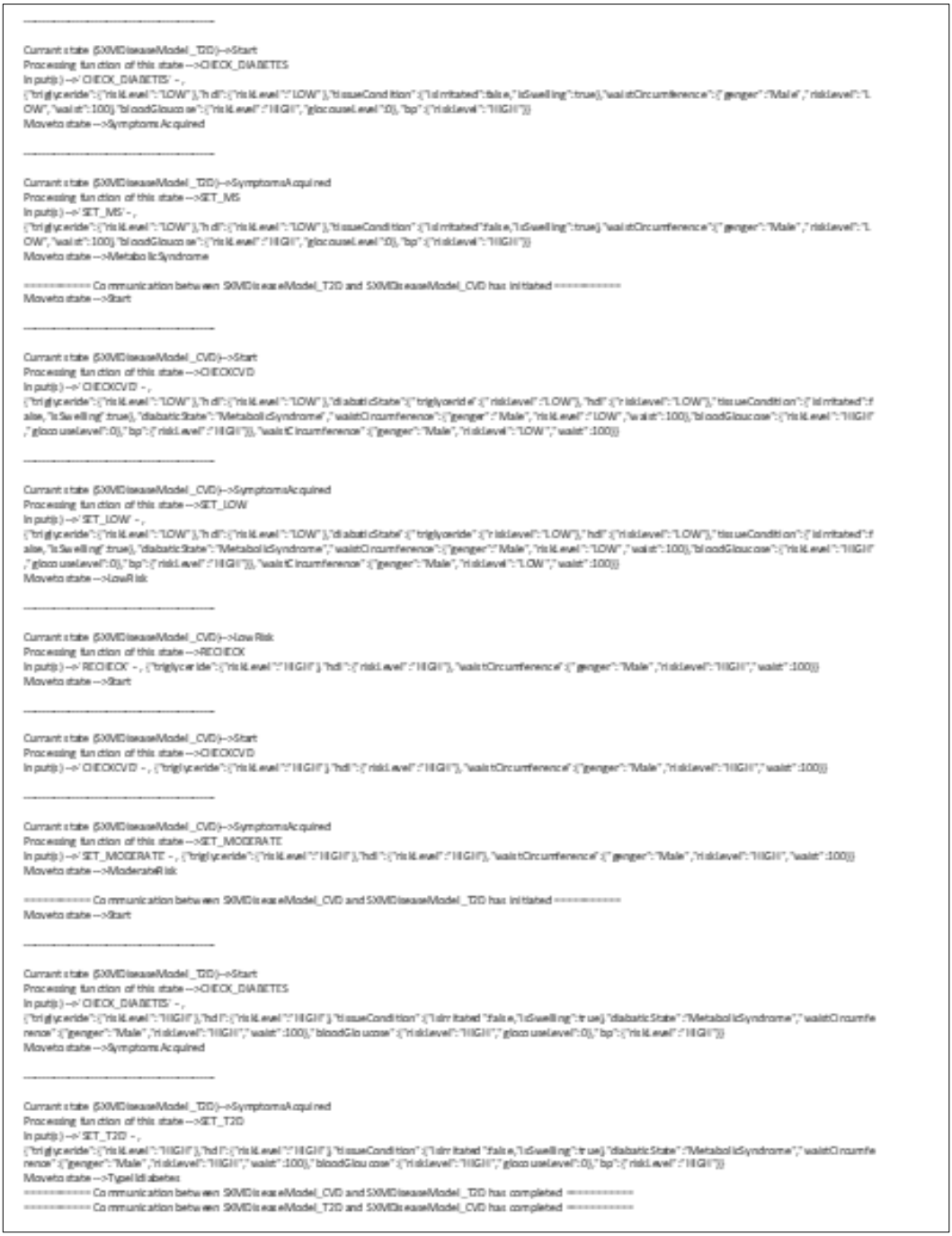

Fig 10. Communication between CVDRM & TTDM

In this research, using formal methods in general and especially classes of X-Machines we have identified some potential limitations. One of the most prominent would the limitation of having inner

states as mentioned above. Hence, diseases might have internal states or many stages, managing and designing could be complicated.

This has been intended to be used by professionals from various domains. The limitation from the previous model remains that, this is heavily programming dependent (all the models need to be programmed using JAVA). This poses a serious limitation in both time and expertise.

7. Conclusion & Future Research

With the limitations identified in the Section 6, we the research team is intended to investigate the possibility of introducing the inner states in the SXMDM and remain the capability using SXMCM. With the successful implementation of the SXMCM, we have been able to demonstrate the ability of state base communication while preserving the identity of the particular state that communication has been initiated. Furthermore, this has given the ability of customising the response to the communication from the point of receiving the message. As discussed in the Section 2, the world is moving towards alternative medicine and lifestyle interventions for treating, managing, and preventing chronic diseases. The ultimate intention of this research is to produce a comprehensive tool which has the capability of modelling diseases and their inter-relationships and has the capability of mimicking the reaction upon exposure to nutrients. This will give the medical professionals the ability of determining the course of action without risking the lives of patients and apply alternative medicine in patient care with more confidence.

In conclusion, this paper has presented a novel approach of modelling diseases with their inter-relationship(s) without concerning the inner states (substages) of the diseases. Even though the current combination of SXMDM and SXMCM, has presented a novel approach of disease modelling with the discussed reservations in Section 6. As the main researcher in this project, we firmly believe that this is a significant milestone of our research. We are committed to conduct further research on this and improve the disease models further.

References

[1] S. Licher, A. Heshmatollah et al. Lifetime risk and multimorbidity of non-communicable diseases and disease-free life expectancy in the general population: A population-based cohort study. PLOS Medicine, vol. 16, issue 2, 2019, article id e1002741.

[2] J.F. Ferguson, H. Allayee et al. Nutrigenomics, the Microbiome, and Gene-Environment Interactions: New Directions in Cardiovascular Disease Research, Prevention, and Treatment: A Scientific Statement from the American Heart Association. Circulation: Genomic and Precision Medicine, vol. 9, issue 3, 2016, pp. 291-313.

[3] S. Jayatilake, E. Ogunshile et al. Modelling Diseases with Stream X-Machine. In Proc. of the 9th International Conference in Software Engineering Research and Innovation (CONISOFT), 2021, pp. 61-68.

[4] B.M. Sörensen et al. Prediabetes and Type 2 Diabetes are Associated with Generalized Microvascular Dysfunction: The Maastricht Study, Circulation, vol. 134, issue 18, 2016, pp. 1339–1352.

[5] S.F. Hossain, M. Huang et al. Inter Disease Relations Based on Human Biomarkers by Network Analysis, In Proc. of the IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE), 2019, pp. 103-108.

[6] J. Seo, B. Choi et al. The relationship between multiple chronic diseases and depressive symptoms among middle-aged and elderly populations: results of a 2009 korean community health survey of 156,747 participants. BMC Public Health, vol. 17, no. 1, 2017, article id 8442.

[7] C.D. Filippo, M. Di Paola et al. Diet, Environments, and Gut Microbiota. A Preliminary Investigation in Children Living in Rural and Urban Burkina Faso and Italy. Frontiers in Microbiology, vol. 8, 2017, article id. 1979

[8] E.A. Finkelstein, J.G. Trogdon et al. Annual Medical Spending Attributable to Obesity: Payer-And Service-Specific Estimates. Health Affairs, vol. 28, 2009, pp. w822-–w831.

[9] J. Bowen and V. Stavridou. Safety-critical systems, formal methods and standards. Software Engineering Journal, vol. 8, issue 4, 1993, pp. 182-183.

[10] J.S. Ostroff. Formal methods for the specification and design of real-time safety critical systems. Journal of Systems and Software, vol. 18, issue 1, 1992, pp. 33-60.
[11] S. Gerhart, D. Craigen, and T. Ralston. Experience with formal methods in critical systems. IEEE Software, vol. 11, issue 1, 1994, pp. 21-28.
[12] S. Eilenberg, Automata, languages, and machines. Academic Press, 1974, 450 p.
[13] F. Ipate and M. Holcombe. A method for refining and testing generalised machine specifications. International Journal of Computer Mathematics, vol. 68, issue 3-4, 1998, pp. 197-219.
[14] F. Ipate, M. Holcombe. Specification and Testing Using Generalised Machines: A Presentation and a Case Study. Software Testing, Verification and Reliability, vol. 8, issue 2, 1998, pp. 61-81.
[15] M.A. Boemo, L. Cardelli, and C.A. Nieduszynski. The Beacon Calculus: A formal method for the flexible and concise modelling of biological systems. PLOS Computational Biology, vol. 16, issue 3, 2020, article id. e1007651.
[16] I. Stamatopoulou, I. Sakellariou et al. Formal modelling for in-silico experiments with social insect colonies. In Proc. of the 11th Panhellenic Conference in Informatics (PCI'07), vol. B, 2007, pp. 79-89.
[17] S. Coakley, R. Smallwood, and M. Holcombe. Using x-machines as a formal basis for describing agents in agent-based modelling. In Proc. of the Spring Simulation Multiconference 2006 (SpringSim'06), 2006, pp. 33-40.
[18] T. Balanescu, A. Cowling et al. Communicating Stream X-Machines Systems are no more than X-Machines. Journal of Universal Computer Science, vol. 5, issue 9, 1999, pp. 494-507.
[19] H. Georgescu and C. Vertan. A New Approach to Communicating X-Machine Systems. Journal of Universal Computer Science, vol. 6, issue 5, 2000, pp. 490-502.
[20] J. Barnard. COMX: a design methodology using communicating X-machines. Information and Software Technology, vol. 40, issue 5-6, 1998, pp. 271-280.
[21] P. Kefalas, G. Eleftherakis, and E. Kehris. Communicating X-Machines: From Theory to Practice. Lecture Notes in Computer Science, vol. 2563, 2003, pp. 316-335.
[22] M.A. Schaub, T.A. Henzinger, and J. Fisher. Qualitative networks: a symbolic approach to analyze biological signaling networks. BMC Systems Biology, vol. 1, 2007, article no. 4.
[23] A. Naldi, D. Thieffry, and C. Chaouiya. Decision Diagrams for the Representation and Analysis of Logical Models of Genetic Networks. Lecture Notes in Computer Science, vol. 4695, 2007, pp. 233-247.
[24] N. Miskov-Zivanov, P. Wei, and C. S. C. Loh. THiMED: Time in Hierarchical Model Extraction and Design. Lecture Notes in Computer Science, vol. 8859, 2014, pp. 260-263.
[25] S. Wolfram. A New Kind of Science. Available at: https://www.wolframscience.com/, accessed May 25, 2021.
[26] I. Stamatopoulou, M. Gheorghe, and P. Kefalas. Modelling Dynamic Organization of Biology-Inspired Multi-Agent Systems with Communicating X-Machines and Population P Systems. Lecture Notes in Computer Science, vol. 3365, 2005, pp. 389-403.
[27] A. Rashid, O. Hasan et al. Formal reasoning about systems biology using theorem proving. PLOS ONE, vol. 12, issue 7, 2017, article id. e0180179.
[28] Q. Wang. Formal Methods for Biological Systems: Languages, Algorithms, and Applications. Ph.D. Thesis. Carnegie Mellon University, 2016, 149 p.
[29] Q. Wang and E. M. Clarke. Formal modeling of biological systems. In Proc. of the 2016 IEEE International High Level Design Validation and Test Workshop (HLDVT), 2016, pp. 178-184.
[30] J. Mortimer, ed. The FMS Report: Ingersoll Engineers. Springer, 1984, 180 p.
[31] C. Chaouiyaю Petri net modelling of biological networksю Briefings in Bioinformatics, vol. 8, issue. 4, 2007, pp. 210-219,
[32] T. Koster, P. J. Giabbanelli, and A. Uhrmacher. Performance and Soundness of Simulation: A Case Study Based on a Cellular Automaton for In-Body Spread of HIV. In Proc. of the 2020 Winter Simulation Conference (WSC), 2020, pp. 2281-2292.
[33] G. Pardini. Formal modelling and simulation of biological systems with spatiality. Ph.D. Thesis. Università di Pisa, 2011, 145 p.
[34] D. Dranidis, K. Bratanis, and F. Ipate. JSXM: A Tool for Automated Test Generation. Lecture Notes in Computer Science, vol. 7504, 2012, pp. 352-366.
[35] W. Hao, H.M. Komar et al. Mathematical model of chronic pancreatitis. Proceedings of the National Academy of Sciences *(*PNAS), vol. 114, issue 19, 2017, pp. 5011-5016.
[36] M.B. Alazzam, A.A. Hamad, and A.S. AlGhamdi. Dynamic Mathematical Models' System and Synchronization. Mathematical Problems in Engineering, 2021, article id. 6842071.

[37] A. De Gaetano, T. Hardy et al. Mathematical models of diabetes progression. American Journal of Physiology-Endocrinology and Metabolism, vol. 295, issue 6, pp. E1462–E1479.
[38] D. Wodarz and M.A. Nowak. Mathematical models of HIV pathogenesis and treatment. BioEssays, vol. 24, issue 12, 2002, pp. 1178-1187.
[39] M. Buysschaert, J.-L. Medina et al. Definitions (and Current Controversies) of Diabetes and Prediabetes. Current Diabetes Reviews, vol. 12, issue 1, 2015, pp. 8-13.
[40] S. M. Haffner. Pre-diabetes, insulin resistance, inflammation and CVD risk. Diabetes Research and Clinical Practice, vol. 61, 2003, pp. S9-S18.

Information about authors / Информация об авторах

Senerath Mudalige Don Dilshan Dhanishtha JAYATILAKE, Master of Science. Research interests: Software Development, Java Programming, Object-Oriented Programming, Java Language, Relational Databases.

Сенерат Мудалиге Дон Дилшан Дхаништха ДЖАЯТИЛАКЕ, магистр наук. Область научных интересов: разработка программного обеспечения, программирование на Java, объектно-ориентированное программирование, язык Java, реляционные базы данных.

Khoa PHUNG, PhD Student. Research interests: Java, program testing, software fault tolerance, decision trees, diseases, learning (artificial intelligence), medical computing, medical diagnostic computing, mobile computing, multilayer perceptrons, pattern classification, program debugging, quality assurance, software metrics, software quality, support vector machines.

Хоа ФУНГ, аспирант. Область научных интересов: Java, тестирование программ, отказоустойчивость программного обеспечения, деревья решений, болезни, обучение (искусственный интеллект), медицинские вычисления, медицинские диагностические вычисления, мобильные вычисления, многослойные персептроны, классификация шаблонов, отладка программ, обеспечение качества, метрики программного обеспечения, качество программного обеспечения, машины опорных векторов.

Emmanuel OGUNSHILE, Ph.D., Senior Lecturer in Computer Science and Chair Athena SWAN process. Research interests: Automated Specification, Verification and Testing of Software and Hardware, Model Based System Engineering, Formal Methods, Object-Oriented Languages, Type Theory, Cloud Computing, Big Data, Machine Learning, Robotics, Artificial Intelligence, Data Science, Internet of Things, Cyber-Security and Natural Language Processing.

Эммануэль ОГУНШИЛЕ, кандидат наук, старший преподаватель компьютерных наук. Научные интересы: автоматизированная спецификация, проверка и тестирование программного и аппаратного обеспечения, разработка систем на основе моделей, формальные методы, объектно-ориентированные языки, теория типов, облачные вычисления, большие данные, машинное обучение, робототехника, искусственный интеллект, наука о данных, Интернет вещей, кибербезопасность и обработка естественного языка

Mehmet AYDIN, Ph.D., Senior Lecturer in Computer Science. Research interests: machine learning, particularly reinforcement learning, wired/wireless network planning and optimization, combinatorial optimization, parallel and distributed metaheuristics, evolutionary computation and intelligent agents and multi agent systems.

Мехмет АЙДИН, кандидат наук, старший преподаватель компьютерных наук. Область научных интересов: машинное обучение, особенно обучение с подкреплением, планирование и оптимизация проводных/беспроводных сетей, комбинаторная оптимизация, параллельная и распределенная метаэвристика, эволюционные вычисления, интеллектуальные агенты и мультиагентные системы.

DOI: 10.15514/ISPRAS-2022-34(6)-12

Опция «Phonology» платформы LingvoDoc как способ верификации (на материале сосьвинского диалекта мансийского языка)

Н.А. Кошелюк, ORCID: 0000-0002-5833-7971 <NKoshelyuk@yandex.ru>

Институт системного программирования им. В.П. Иванникова РАН
109004, Россия, г. Москва, ул. А. Солженицына,д. 25

Аннотация. В статье приводятся результаты исследования полевого материала сосьвинского мансийского диалекта села Ломбовож. Его экспериментально-фонологический анализ был выполнен с помощью современной системы обработки данных LingvoDoc. Сопоставление полученной в результате этого анализа системы гласных звуков с другими фонетическими системами сосьвинского диалекта, предложенными лингвистами XX в., позволило прийти к заключению о необходимости дальнейшего уточнения фонетической системы этого мансийского диалекта. Так, в процессе исследования были выявлены неточности в интерпретации полевого материала села Ломбовож, а также обнаружен ряд уникальных фонем (o, ɛ, e, i, ə, u), обладающих нехарактерными показателями для общепринятых параметров F1 и F2 международного фонетического алфавита, базирующегося в основном на данных анализа европейских языков.

Ключевые слова: мансийский язык; сосьвинский диалект; полевые данные; фонология; LingvoDoc

Для цитирования: Кошелюк Н.А. Опция «Phonology» платформы LingvoDoc как способ верификации (на материале сосьвинского диалекта мансийского языка). Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 165-172. DOI: 10.15514/ISPRAS-2022-34(6)-12

Благодарности. Исследование выполнено при поддержке гранта РНФ № 20-18-00403 ‘Цифровое описание диалектов уральских языков на основании анализа больших данных’ (рук. Ю.В. Норманская).

The «Phonology» option of the LingvoDoc platform as a verification method (based on the data of the Sosva dialect of the Mansi language)

N.A. Koshelyuk, ORCID: 0000-0002-5833-7971 <NKoshelyuk@yandex.ru>

Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The article presents the results of a study f the field material of the Sosva Mansi dialect of the village of Lombovozh. Its phonological analysis was performed using a modern data processing system Lingvodic. Comparison of the system of vowel sounds obtained as a result of this analysis with other phonetic systems of the Sosva dialect proposed by linguists of the XX century, allowed us to come to the conclusion that it is necessary to further refine the phonetic system of this Mansi dialect. Thus, in the course of the study, inaccuracies in the interpretation of the field material of the village of Lombovozh were revealed, and a number of unique phonemes (o, ɛ, e, i, ə, u) were found that have uncharacteristic indicators for the generally accepted parameters F1 and F2 of the international phonetic alphabet, based mainly on the data of the analysis of European languages.

Keywords: Mansi language; Sosva dialect; field data; phonology; LingvoDoc

For citation: Koshelyuk N.A. The «Phonology» option of the LingvoDoc platform as a verification method (based on the data of the Sosva dialect of the Mansi language). Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 165-172 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-12

Acknowledgments. Supported by Russian Science Foundation, project no. 20-18-00403 'Digital Description of Uralic Languages on the Basis of Big Data' (Yu.V. Normanskaya)

1. Введение

Разработанные на сегодняшний день системы вокализма мансийского языка признаются корректными многими лингвистами, однако они, так или иначе, противоречат друг другу. Условно их можно объединить в две основные группы по именам исследователей, чьи мнения совпадают [1, 2] (табл. 1-2):

1) «Штейнитц-Хонти»

Табл. 1. Система вокализма первого слога северномансийского диалекта по [3, 4]
Table 1. The system of vocalism of the first syllable of the Northern Mansi dialect according to [3, 4].

Ряд	Негубные	Губные
Верхний	*ē i*	*u ū*
Нижний	*a ā*	*o ō*

2) «Кальман-Ромбандеева-Керестеш»

Табл. 2. Система гласных фонем первого слога северных мансийских диалектов по [5, 6, 2]
Table 2. The system of vowel phonemes of the first syllable of the northern Mansi dialects according to [5, 6, 2]

Ряд Подъем	Передний		Средний		Задний	
	негубные	губные	негубные	губные	негубные	губные
Верхний	*i ī*					*u, ū*
Средний	*ē (e)*					*o ō*
Нижний			*a a*			

Система вокализма для сосьвинского говора Б. Кальмана, опубликованная в его работе «Das nordwogulische Phonemsystem» [5] строится на полевом материале 1957-1958 гг. Анализируя собранные данные, исследователь пришел к выводу, что транскрипция мансийского языка, предложенная до него в [3], не совсем корректна: в ряде слов ученым был обнаружен аллофон *ī* (в словах 'жеребенок', 'лоб', 'дерево', 'книга', 'филин', 'он танцует' и т.д.), регулярно появляющийся после и перед согласным *j* или палатализованными согласными *ń, l′, t′* [5]. Гласные фонемы, выявленные для северной диалектной группы (верхнелозьвинского, сосьвинского и сыгвинского говоров) Л. Хонти, являются результатом многолетних исследований венгерского исследователя и выявляются на основе данных экспедиции А. Каннисто и собственных наработок лингвиста. В материалах Л. Хонти, как и у В. Штейнитца, фонема *ī* отсутствует (см. [7]).

Известны и другие системы вокализма для северных мансийских диалектов. Например, отличающийся от представленных выше взгляд на мансийский вокализм представлен в издании 1957 года «Мансийский язык: учебное пособие для педагогических училищ» [8]. В книге предпринята первая попытка дать комплексную характеристику мансийского языка.

Лингвисты выделяют шесть гласных фонем в сосьвинском диалекте мансийского языка: *а, о, у, и, ы, э* (табл. 3). При этом, гласные фонемы *а, о, у* могут быть и нейтральными, и долгими; гласный э всегда долгим, кроме случаев, когда он стоит в безударной позиции второго слога [8].

По сравнению с системой вокализма по Кальману-Ромбандеевой-Керестешу, в материалах [8] не представлены долгие фонемы *и* (*ī*) и э (*ē*), и в отличие от данных [2], [3], [4], [5], [6], отдельно выделяется *ы* (*ɨ*).

Табл. 3. *Система гласных фонем первого слога сосьвинского диалекта [8*
Table 3. The system of vowel phonemes of the first syllable of the Sosva dialect [8].

Ряд Подъем	Передний		Средний		Задний	
	негубные	губные	негубные	губные	негубные	губные
Верхний	*и*		*ы*			*у (ӯ)*[1]
Средний	*э*					*о (ō)*
Нижний						*а (ā)*

В [9] предложена точка зрения на мансийскую фонологию, полностью совпадающая с В. Штейнитцем и Л. Хонти: исследователь сходится во мнении с предшественниками и последователями в отношении гласных *и,о, а* и их долгих вариантов, но исключает наличие фонем *ī* и *е*, и *ɨ* (табл. 4).

Табл. 4. Система вокализма первого слога для сосьвинского диалекта [9]
Table 4. The system of vocalism of the first syllable for the Sosva dialect [9]

Ряд Подъем	Передний		Средний		Задний	
	негубные	губные	негубные	губные	негубные	губные
Верхний	*і*		*ы*			*и (ū)*
Средний	*ē*					*о (ō)*
Нижний					*а, а*	*а (ā)*

Уточнение системы вокализма сыгвинского и сосьвинского северных диалектов мансийского языка представлено Е. И. Ромбандеевой в изданиях [10, 11], где собраны результаты экспериментально-фонетического и фонологического исследования ученого. Работа над фонемным составом сыгвинского диалекта была проведена под руководством М. И. Матусевич в 1959 и 1967 годах в фонетической лаборатории Ленинградского государственного университета.

Для обозначения фонем сыгвинского диалекта Е.И. Ромбандеевой были использованы знаки, принятые в общей финно-угорской традиции (табл. 5). В этом диалекте, по мнению исследователя, насчитывается 15 гласных и 19 согласных фонем [10].

Табл. 5. Общая система вокализма сыгвинского диалекта, выполненная на основе материалов 1970-х г. [10]
Table 5. The general system of vocalism of the Sygvin dialect, made on the basis of data from the 1970s [10].

Место артикуляции Положение языка (подъем)	Передний		Передне-средний		Задне-средний		Задний	
	ил.-лаб.	лаб.	ил.-лаб.	лаб.	ил.-лаб.	лаб.	ил.-лаб.	лаб.
Верхний1	*i*							*u̯ и*
Верхний2	*ī*							*ū*
Средний1	*e*		*ə*[2]		*i̯*			
Средний2	*ē (ɛ)*		*ɛ̯*[3]		*ī̯*			*ō*
Низкий			*а*					
Самый низкий					*ā*			

Фонетические данные по сосьвинскому диалекту мансийского языка приводятся в монографии Е. И. Ромбандеевой «Современный мансийский язык: лексика, фонетика, графика, орфография, морфология, словообразование» [11]. Полевой материал, положенный в основу исследования, был собран лингвистом в разные годы в период с 1958 по 1969 гг. и охватил практически все известные места расселения манси. Информантами выступили в

[1] В круглых скобках отмечены фонемы, характерные для позиции первого слога, но редко встречающиеся.

[2] Не встречается в первом слоге.

[3] Не встречается в первом слоге.

основном люди старшего поколения в возрасте 50-90 лет. В результате экспедиций Е. И. Ромбандеевой был собран языковой материал на 10 тысяч страниц рукописных текстов. Обработка данных также была проведена на базе лаборатории экспериментальной фонетики ЛГУ под началом проф. М. И. Матусевич и В. М. Наделяева [11]. В результате проведенной лабораторной работы по изучению фонемного состава мансийского языка было установлено, что в сосьвинском диалекте мансийского языка выделяется 12 гласных фонем (табл. 6). Как пишет Е. И. Ромбандеева, «наиболее существенной особенностью гласных мансийского языка является фонематическая оппозиция долгих и кратких (квантитативный признак)» [11].

Как видно из таблицы, в сравнении с данными Б. Кальмана и Л. Керестеша в [11] представлено больше фонем: *i̯, ī, e, ē, ə~ī̆, ù~ĭ*; относительно системы вокализма В. Штейнитца, Л. Хонти и Л. Мёрфи добавляются фонемы *ī, e, i̯, ə~ī̆, ù~ĭ*, а в сравнении с [8] – *ī, ē, ə~ī̆*.

Табл. 6. Система вокализма сосьвинского диалекта [11]
Table 6. The system of vocalism of the Sosva dialect [11].

Ряд Подъем	Передний		Средний		Задний	
	Неогубленные				Огубленные	
	краткие	долгие	краткие	долгие	краткие	долгие
Верхний	*i*	*ī*			*u*	*ū*
Средний	*e*	*ē*	*ù~ĭ*[4]	*ə~ī̆*[5]	*o*	*ō*
Нижний			*a*	*ā*		

Исходя из приведенного материала, мы видим, насколько неоднозначна система вокализма по первому слогу не только одной диалектной группы мансийского языка, но и его поддиалектов: Б. Кальман и Л. Керестеш насчитывают в сосьвинском диалекте 10 гласных фонем, В. Штейнитц, Л. Хонти (верхнелозьвинский, сосьвинский, сыгвинский) – 8, А. Н. Баландин и М. П. Вахрушева (сосьвинский) – 9, Л. Мёрфи для сосьвинского – 8, Е. И. Ромбандеева – 13 (сыгвинский) и 10 фонем (сосьвинский).

Табл. 7. Система гласных звуков северного сосьвинского диалекта с. Ломбовож (1975 г.)
Table 7. The system of vowel sounds of the northern Sosva dialect of S. Lombovozh (1975)

	Передние	Ненапряженные передние	Средние	Ненапряженные задние	Задние
Верхние	*ii:*		*ɨɨ: ʉ ʉ:*		*uu:*
Ненапряженные верхние		*ɪ ɪ:*	*ɪ̈ ɪ̈:*	*ʊ*	
Средне-верхние	*e e:*		*ɵ*		*ɤ o o:*
Средние			*ə*		
Средне-нижние	*ɛɛ: oe*		*ɜɞ*		*ɔ*
Ненапряженные нижние	*æ*		*ɐ*		
Нижние	*a a: ä*				*a: ɒ*

Привлечение нового полевого материала, обработанного с помощью современных фонетических программ, позволяет провести верификацию этих данных. В качестве такого источника был выбран словарь сосьвинского диалекта села Ломбовож (табл. 7). В настоящее время словарь размещен на лингвистической платформе LingvoDoc[6]. Он состоит из 234 лексем с размещенной звуковой разметкой, предварительно выполненной в программе Praat. Известно, что материал этого словаря был собран новосибирскими учеными в 1975 г. в

[4] Не встречается в первом слоге.

[5] Не встречается в первом слоге.

[6] См. http://lingvodoc.ispras.ru/dictionary/688/9870/perspective/688/9871/view.

Лаборатории экспериментально-фонетических исследований в 1975 г. [12], однако его введение в научный оборот и частичный анализ (исследование долгот) был осуществлен Ю. А. Тамбовцевым в 2009 г. [13], а полноценная экспериментальная обработка впервые проведена сотрудницей лаборатории лингвистической антропологии Томского государственного университета Е. В. Ковалевой лишь в 2017 году. Информантом выступил П. Е. Шешкин из д. Ломбовож, Ломбовожского сельсовета.

По данным фонетического анализа, выполненного в системе LingvoDoc, состав гласных сосьвинского диалекта с. Ломбовож составляет 33 аллофона (табл. 7), объединенных алгоритмами *Phonology*[7] в 9 фонем: *a, i, o, ʊ, i, ɐ, ɛ, e, ə*[8] (наличие не менее 8 примеров с наибольшей долготой предполагаемого гласного засчитывалось как существующий аллофон). В сопоставлении с представленными выше традиционными системами гласных, в полевых материалах северного мансийского диалекта количество фонем совпадает с данными Штейнитца, Хонти и Мёрфи, но имеют различия по качеству: ʊ, ɐ, ɛранее не зафиксированы ни в одной из них.

В статье представлена попытка соотнести эти фонетические варианты гласных с традиционной транскрипцией, чтобы проверить, действительно ли можно говорить о наличии в сосьвинском мансийском диалекте подобного набора фонем. Так, ниже мы провели сравнительный анализ показателей фонем полевого материала в сопоставлении со средними значениями параметров F1 и F2 для гласных по данным МФА (IPA).

2. Исследование

В основе алгоритма Phonology, с помощью которого был проведен наш фонологический анализ данных с. Ломбовож, лежат результаты, изначально полученные Е. В. Ковалевой с помощью фонетической программы Praat, в которой для каждого звука определяются физические характеристики звука: форманты, длительность, интенсивность. После чего вся разметка была подгружена в словарь этого диалекта на платформе LingvoDoc и запущен алгоритм анализа.

Известно, что для каждого гласного звука в выбранном словаре Phonology собирает его физические характеристики во всех произнесениях, обрабатывая спектрограммы, предварительно размеченные в фонетической программе Praat на отдельные звуки. На выходе исследователь получает таблицу Excel со списком всех физических характеристик каждого звука всего множества лексем в исследуемом диалекте. При создании этой таблицы используются алгоритмы анализа интенсивности звука и формантных характеристик Praat.

В результате работы программы все данные таблицы Excel с указаниями физических параметров каждого звука преобразуются в 3D-график, построенный на основании показателей трех формант (F1-F2-F3) для каждого гласного звука (рис. 1). О правильности выделения той или иной фонемы мы можем заключить, обратившись к 3D-модели, сформированной программой LingvoDoc на основе результатов фонологического анализа.

Говорить о правомерности выделения фонемы схожих дифференциальных признаков (ряд, подъем, лабиализованность) и звучания можно на основании того, насколько сильно пересечение их облаков. В данном случае о возможной ошибке может говорить близость расположения облаков формант *i* и *ʊ*. Но в пользу правильности выделения этих звуков для сосьвинского диалекта с. Ломбовож говорит тот факт, что гласный ʊ является ненапряженным огублённым заднего ряда верхнего подъема, а *i*-неогублённым гласным среднего ряда верхнего подъема.

[7] Подробнее с результатами анализа можно ознакомиться, скачав фонологический анализ словаря на платформе LingvoDoc (Tools -> Phonemic Analysis).

[8] Не встречается в первом слоге.

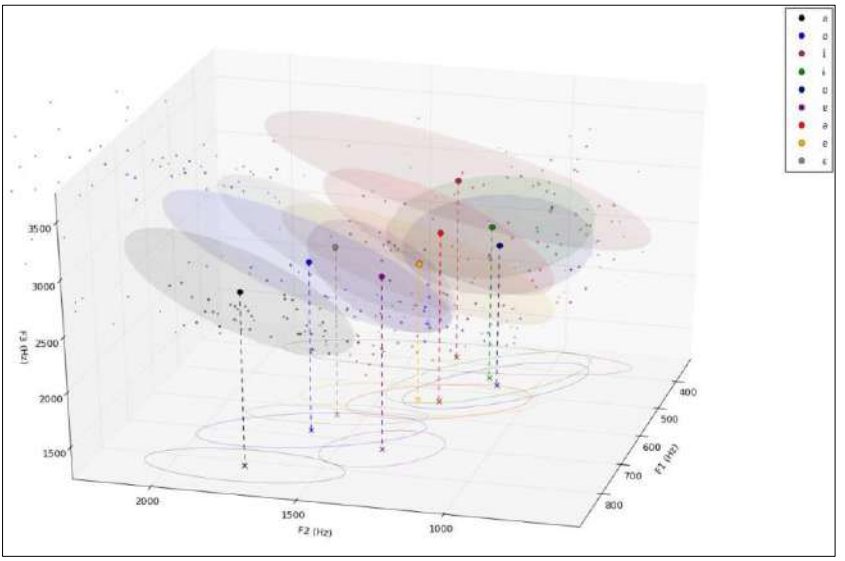

Рис.1. 3D-модель гласных формант сосьвинского диалекта с. Ломбовож
Fig. 1. 3D model of vowel formants of the Sosva dialect of the village of Lombovozh

Таким образом, согласно формантной 3D-модели сосьвинского диалекта с. Ломбовож, мы можем говорить о довольно корректной обработке фонетического материала. Однако для того, чтобы подобное заключение можно было считать правомерным, мы провели сопоставление выделенных с помощью программы Praat параметров гласных фонем полевых данных со средними значениями гласных международного фонетического алфавита.

В табл. 8 отмечены все полученные в результате обработки полевого материала в программе Praat гласные фонемы с указанием их минимального и максимального диапазона. Сопоставление этих данных с данными МФА выявило некоторые неточности в проведенном ранее Е. В. Ковалевой фонологическом анализе.

Табл. 8. Форманты гласных первого слога сосьвинского диалекта с. Ломбовож
Table 8. Vowel formants of the first syllable of the Sosva dialect of the village of Lombovozh

Аллофон		Диапазон
	Минимальный показатель F1 (Гц)	Максимальный показатель F2 (Гц)
a	514,088-941,115	1641,192-2290,902
ɨ	341,263-619,996	1138,123-1502,069
o	692,61-786,262	1594,894-1893,258
ʊ	501,565-659,786	1147,43-1593,267
i	385,291-427,651	1092,098-1500,312
ɐ	735,104-835,24	1379,79-1565,969
ə	575,691-644,74	1266,588-1547,925
ɛ	608,491-679,402	1488,502-1856,715
e	580,356-638,697	1635,976-2151,768

В соответствии с МФА [14], средние показатели *а* равны F1 850 и F2 1610 Гц, что соответствует значениям F1-F2 мансийских полевых данных для этого аллофона по F1, но, согласно F2, мансийская *а* имеет более высокую частоту – 1950 Гц; для *у* по МФА характерны следующие значения: F1 235 Гц, F2 2100 Гц – в исследуемом материале он обозначается как *ɨ* и имеет показатели F1 341,263-619,996 Гц и F2 1138,123-1502,069 Гц. Значительное расхождение указывает на неверно определенный аллофон. Предполагаем, что по заданным параметрам, по МФА ему ближе всего неогубленный гласный заднего ряда верхнего подъема *ш* F1 300 Гц, F2 1390 Гц.

Аллофон *o,* по МФА, имеет средние значения F1 360 Гц и F2 640 Гц, но его показатели, согласно полевым данным с. Ломбовож, превосходят указанные пределы примерно в 2 раза – F1 692,61-786,262 Гц и F2 1594,894-1893,258 Гц, что свидетельствует о необходимости уточнить, насколько правильно было проведено выделение этого варианта фонемы. При соотнесении среднего-арифметического числа указанных параметров с показателями других огубленных гласных системы МФА, сходств между ними выявлено не было.

Диапазон ненапряженного огубленого гласного заднего ряда верхнего подъема *ʊ* составляет F1 501,565-659,786 Гц и F2 1147,43-1593,267 Гц. Предполагаем, что выделение такого аллофона также нуждается в корректировке, так как его значения ближе всего соотносятся с усредненными параметрами другого гласного международного фонетического алфавита – *ɤ* (F1 460 Гц, F2 1310 Гц). Аллофон *ɐ*, согласно данным экспериментально-фонетической программы Praat, имеет показатели F1 735,104-835,24 и F2 1379,79-1565,969 Гц, что отличается от средних показателей этого гласного по МФА. Вероятно, обозначение *ɐ* также следует несколько скорректировать: по параметрам он в большей степени соотносится с неогубленым гласным переднего ряда нижнего подъема *a*.

Показатели аллофона *ɛ* полевых данных и его средние значения F1 610 Гц, F2 1900 Гц для соответствующего гласного по МФА можно назвать сопоставимыми по F1, но расходящимся в отношении параметра F2, где полевым данным соответствуют: F1 608,491-679,402 и F2 1488,502-1856,715 Гц. Согласно данным МФА, усредненные значения аллофона *e*равны F1 390 Гц и F2 2300 Гц, при этом показатели для этого гласного, полученные в результате обработки полевого материала, имеют следующие значения: F1 580,356-638,697 Гц, F2 1635,976-2151,768 Гц. Отметим, что эти параметры максимально близки значениям F1-F2 для *ɛ* по МФА, а также приближены к этому же аллофону, но выделенному Е. В. Ковалевой в результате анализа полевых данных сосьвинского мансийского диалекта с. Ломбовож. Однако однозначно говорить об ошибочном обозначении аллофона *e* и необходимости его переименования мы не можем, ввиду довольно значительного расхождения их показателей по верхней границе F2 на 295,053 Гц.

Параметры *ə* по МФА составляют примерно F1 500 и F2 1510 Гц, что соответствует средним значениям F1 и F2 для этого аллофона в рассматриваемых полевых данных.

3. Заключение

В результате сравнительного анализа показателей формант полевых данных с. Ломбовож со средними значениями соответствующих гласных МФА, мы допускаем вероятность несколько некорректного обозначения *i, ʊ, ɐ*, а также можем предположить наличие нескольких уникальных для мансийского языка аллофонов, чьи формантные показатели отличаются от общеизвестных показателей МФА, полученных в результате анализа, в основном, европейских языков, и указывающих на ранее типологически неописанную фонетическую уникальность этого языка – *a, o, ɛ, e, i.*

Таким образом, проведенное комплексное исследование полевого материала с. Ломбовож с помощью экспериментально-фонетического и фонологического анализа, выполненного на базе платформы «LingvoDoc», позволяет заключить, насколько точно исследователями XX в. и современными лингвистами была описана фонетическая система северного мансийского диалекта.

Список сокращений

МФА – международный фонетический алфавит; F1 – самый низкий показатель частоты форманта; F2 – самый высокий показатель частоты форманта; Гц – герц (единица частоты колебаний).

Список литературы / References

[1] Vogul R.T. Languages of the World / Materials 158. Lincom Europa, 2001, 89 p.
[2] Keresztes L. Mansi. In Daniel Abondolo, ed. The Uralic languages, Routledge, 1998, pp. 387-427.
[3] Steinitz W. Geschichte des wogulischen Vokalismus. Berlin, Akademie-Verlag, 1955. 366 p. (in German).
[4] Honti L. Geschichte des obugrischen vokalismus der ersten Silbe. Budapest, Akadémiai Kiadó, 1982, 227 p. (in German).
[5] Kálmán, B. Das nordwogulische Phonemsystem. In Hajdú P., Honti L., eds. Studien zur phonologischen Beschreibung uralischer Sprachen. Budapest, Akadémiai Kiadó, 1982, 227 p. (in German).
[6] Ромбандеева Е.И. Мансийский (вогульский) язык. М., Наука, 1983 г., 208 стр. / Rombandeeva E.I. Mansi (Vogul) Language. Moscow, Nauka, 1973, 208 p. (in Russian).
[7] Honti L. Die wogulische Sprache. Sinor D., ed. The Uralic Languages. Description, History and Foreign Influences. Brill, 1988, pp. 147-171 (in German).
[8] Баландин А.Н., Вахрушева М.П. Мансийский язык. Учебное пособие для педагогических училищ. Л., Учпедгиз, 1957 г., 275 стр. / Balandin A.N., Vakhrusheva M.P. Mansi Language: A Textbook for Pre-Service Teachers. Leningrad, Uchpedgiz, 1957, 275 p. (in Russian).
[9] Murphy L. W. Sosva Vogul Grammar. Ph.D. Tesis. Indiana University, 1968, 187 p.
[10] Ромбандеева Е.И. Сыгвинский диалект мансийского (вогульского) языка. М., Гамбург, 1995 г., 204 стр. / Rombandeeva E. I. The Sygva dialect of the Mansi (Vogul) Language. Moscow, Hamburg, 1995, 204 p.
[11] Ромбандеева Е.И. Современный мансийский язык: лексика, фонетика, графика, орфография, морфология, словообразование. 2-е изд. Тюмень, Формат, 2017 г., 318 стр. / Rombandeeva E.I. Modern Mansi: Vocabulary, Phonetics, Graphics, Orthography, Morphology, Word Formation. 2nd ed. Tyumen, Format, 2017, 318 p. (in Russian).
[12] Норманская Ю.В., Кошелюк Н.А. Архивный пелымско-русский словарь, составленный русским священником о. Константином Словцовым, как источник, позволяющий оценить точность записей в мансийских словарях А. Каннисто и Б. Мункачи. Урало-алтайские исследования, вып. 3 (26), 2017 г., стр. 151-161 / Normanskaya Yu.V., Koshelyuk N.A. Archival Pelym-Russian dictionary compiled by the Russian priest, Father Konstantin Slovtsov, as a source for estimating the accuracy of recordings in the Mansi dictionaries of A. Kannisto and B. Munkácsi. Ural-Altaic Studies, issue 3 (26), 2017, pp. 151-161 (in Russian).
[13] Тамбовцев Ю.А. Вариативность долгих и кратких гласных в мансийском языке. Финно-угорский мир, вып. 1, 2009, стр. 22-27 / Tambovtsev Yu.A. The variability of long and short vowels in the Mansi language. Finno-Ugric World, issue 1, 2009, pp. 22-27 (in Russian).
[14] Catford J.C. A Practical Introduction to Phonetics, Second Edition. Oxford University Press, 2001, 229 p.

Информация об авторах / Information about authors

Наталья Андреевна КОШЕЛЮК – кандидат филологических наук, младший научный сотрудник лаборатории «Лингвистические платформы». Сфера научных интересов: уральские языки, диалектология, лингвистическая типология, документация и сохранение исчезающих языков, междисциплинарные исследования.

Natalia Andreyevna KOSHELYUK – Candidate of Philological Sciences, Junior researcher at the Linguistic Platforms Laboratory. Research interests: Uralic languages, dialectology, linguistic typology, documentation and preservation of endangered languages, cross-disciplinary research.

DOI: 10.15514/ISPRAS-2022-34(6)-13

Research Perspectives on the Tatar language based on the LingvoDoc platform

[1,2] *F.Sh. Nurieva, ORCID: 0000-0001-9957-9734 <fanuzanurieva@yandex.ru>*
[1] *G.R. Galiullina, ORCID: 0000-0001-6923-2190 <caliullina@list.ru>*
[1] *A.F. Yusupov, ORCID: 0000-0003-0363-5303 <faikovich@mail.ru>*

[1] *Kazan Federal University,*
18 Kremlevskaya str., Kazan, 420008, Russian Federation
[2] *Ivannikov Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia

Abstract. The article discusses research perspectives on the Tatar language based on the LingvoDoc platform. Digitalization of language learning in modern linguistics allows us to move to a new level of describing the language structure. Large corpora containing millions of word forms have been created in all European languages since the 90s of the last century. Currently, this has been done not only in the Russian language, but also in many national languages of Russia such as Tatar, Bashkir, Udmurt, Mari, Moksha, Komi, etc. One of the recognized platforms in modern national linguistics is the development of the LingvoDoc virtual laboratory, created ISP RAS. This platform gives an opportunity to create, store and analyze multilayer dictionaries, language materials and dialects. The main functionality of Lingvodoc is used by more than 250 linguists who process their materials online, more than 1000 dictionaries and 300 text corpora in the national languages of the Russian Federation have already been collected. We consider the possibilities of this platform to study the Tatar language. We believe that electronic corpora allow us to solve a variety of theoretical and practical problems of the language. At present, when the Tatar literary and everyday spoken language is actively used in all fields, it is very important to make a complete description of its features, which will help create more accurate grammars and dictionaries. The relevance of the study is due to the need to use a gloss corpus of texts in the Tatar language. As modern studies in linguistics show, nowadays it is impossible to describe the state of the language without such corpora and analyze its grammatical structure, which corresponds to the world standards of modern science. The LingvoDoc platform makes it possible to process a significant amount of material in a short time and create corpora with glossing and removed homonymy based on samples of the Tatar literary, business, colloquial and dialect languages.

Keywords: Tatar language; LingvoDoc; corpus of the Tatar language; grammar; colloquial speech

For citation: Nurieva F.Sh., Galiullina G.R., Yusupov A.F. Research Perspectives on the Tatar language based on the LingvoDoc platform**.** Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 173-178. DOI: 10.15514/ISPRAS-2022-34(6)-13

Перспективы исследований татарского языка на платформе LingvoDoc

[1,2] *Ф.Ш. Нуриева, ORCID: 0000-0001-9957-9734 <fanuzanurieva@yandex.ru>*
[1] *Г.Р. Галиуллина, ORCID: 0000-0001-6923-2190 <caliullina@list.ru>*
[1] *А.Ф. Юсупов, ORCID: 0000-0003-0363-5303 <faikovich@mail.ru>*

[1] *Казанский (Приволжский) федеральный университет*
420008, Россия, г. Казань, ул. Кремлевская, д.18
[2] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В статье рассматриваются перспективы исследования татарского языка на платформе LingvoDoc. Цифровизация изучения языка в современной лингвистике позволяет перейти на новый уровень описания структуры языка. С 90-х годов прошлого века во всех европейских языках созданы большие корпуса, содержащие миллионы словоформ. В настоящее время это сделано не только в русском языке, но и во многих национальных языках России, таких как татарский, башкирский, удмуртский, марийский, мокшанский, коми и др. Одной из признанных площадок в современном отечественном языкознании является разработанная в ИСП РАН виртуальная лаборатория. Эта платформа дает возможность создавать, хранить и анализировать многослойные словари, языковые материалы и диалекты. Основным функционалом LingvoDoc пользуются более 250 лингвистов, обрабатывающих свои материалы онлайн, уже собрано более 1000 словарей и 300 корпусов текстов на национальных языках РФ. Мы рассматриваем возможности этой платформы для изучения татарского языка. Мы считаем, что электронные корпуса позволяют решать самые разные теоретические и практические проблемы языка. В настоящее время, когда татарский литературно-бытовой разговорный язык активно используется во всех сферах, очень важно сделать полное описание его особенностей, что поможет создать более точные грамматики и словари. Актуальность исследования обусловлена необходимостью использования глоссового корпуса текстов на татарском языке. Как показывают современные исследования в области языкознания, в настоящее время невозможно описать состояние языка без таких корпусов и проанализировать его грамматический строй, соответствующий мировым стандартам современной науки. Платформа LingvoDoc позволяет в сжатые сроки обрабатывать значительный объем материала и создавать корпуса с глоссированием и снятием омонимии на основе образцов татарского литературного, делового, разговорного и диалектного языков.

Ключевые слова: татарский язык; LingvoDoc; корпус татарского языка; грамматика; разговорная речь

Для цитирования: Нуриева Ф.Ш., Галиуллина Г.Р., Юсупов А.Ф. Перспективы исследований татарского языка на платформе LingvoDoc. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 173-178. DOI: 10.15514/ISPRAS-2022-34(6)-13

1. Introduction

Modern linguistic science pays more attention to the language research using digital technologies. Such technologies make it possible to model the “functioning of a language in certain conditions” [1]. As is known, computational linguistics has various developing areas, such as automatic analysis and synthesis of texts, creating and maintaining electronic dictionaries, creating linguistic databases and electronic corpora of languages, etc., they all aimed at a comprehensive analysis of linguistic phenomena [2; 3], as well as the preservation of linguistic facts.

Language corpora are considered to be one of the forms of preservation and ordering of linguistic facts. Electronic corpora along with linguistic information make it possible to solve a variety of theoretical and practical problems of the language. This area is becoming one of leading in modern linguistic research. Our research considers the achievements of national science in this area, but we see that there is the lack of the most complete corpus of the Tatar language with its dialects and colloquial speech, so our study based on LingvoDoc will fill the existing gap in here [4].

Russian linguistics currently has corpora for the Russian language (cf. https://ruscorpora.ru/new/), as well as for many national languages of Russia:

- Tatar (cf. http://www.tugantel.tatar/, https://www.corpus.tatar/),
- Bashkir (cf. https://bashcorpus.ru/),
- Udmurt (cf. http://udmcorpus.udman.ru/, http://udmurt.web-corpora. net/),
- Mari (cf. http://corp.marnii.ru/, http://meadow-mari.web-corpora.net/),
- Erzya (http://erzya.web-corpora.net/) Moksha (http://moksha.web-corpora.net/),
- Komi (cf. http://komi-zyrian.web-corpora.net/)

and corpus work is being actively conducted in other national languages.

Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS) developed a platform for digital data processing in linguistics called LingvoDoc. At present, the LingvoDoc platform (lingvodoc.ispras.ru) allows users to upload a document in Word format and process it using a parser. The available text corpora such as http://www.tugantel.tatar/, https://www.corpus.tatar/ are a linguistic resource of the modern literary Tatar language, but they don't provide complete morphological tagging. Creating corpora of fiction, official texts, dialects and colloquial speech with glossing and removed homonymy using LingvoDoc will allow us to evaluate existing dictionaries and collections of texts in terms of their reliability. Creating electronic corpora, along with linguistic information, is becoming one of the main methods of modern linguistic research.

2. State of art in Tatar language research and perspectives of LingvoDoc use

At present, when the Tatar literary and everyday spoken language is actively used in all fields, it is very important to make a complete description of its features, which will help create more accurate grammars and dictionaries. The relevance of the study is due to the need to use a gloss corpus of texts in the Tatar language. As modern studies in linguistics show, nowadays it is impossible to describe the state of the language without such corpora and analyze its grammatical structure, which corresponds to the world standards of modern science. The LingvoDoc platform makes it possible to process a significant amount of material in a short time and create a corpora with glossing and removed homonymy based on samples of the Tatar literary, business, colloquial and dialect languages.

Our study analyzes the achievements of national science in this field. Research based on LingvoDoc will fill the existing gap that appeared due to the lack of the most complete corpus of the Tatar language with its dialects and colloquial speech.

2.1 Grammar-oriented research

Tatar language grammar and its dialects are fully studied well enough. In recent years, research has been carried out on topic-comment articulation, the semantic structure of the sentence, the syntax and text style as the main unit of speech. The results of grammatical studies are shown in various types of descriptions. Traditionally, they are distinguished as scientific, descriptive and normative.

Scientific grammar includes historical grammar, studying the structure of the language in development or at its individual stages in the past. This field was led by L. Zalay, V.Kh. Khakov, F.S. Faseev, I.A. Abdullin, D.G. Tumasheva, F.M. Khisamova, F.A. Ganiev and others.

Comparative (contrastive) grammar is also a part of scientific grammar. Tatar language grammar describes the similarities and differences in Tatar and other languages, rather than in Russian (grammar books by K.Z. Zinnatullina, E.M. Akhunzyanova, L.K. Bayramova, etc.). The academic book "Tatar Grammar " was published in 3 volumes in 1992-1993 (State Prize of the Republic of Tatarstan, 1994). It contains a description of the phonetic and grammatical structure of the modern Tatar literary language. The Ibragimov Institute of Language, Literature and Art published "Academic lexicology" in 3 parts (2015–2018), "Academic Grammar" in 3 parts (2015–2017) and others, but there are quite a lot of unresolved issues.

2.2 Tatar language morphology

Problems in Tatar language morphology, in particular, the criteria and principles for identifying parts of speech, the interaction of units of different classes, transitional phenomena in the parts of speech system can be solved using the LingvoDoc toolkit. Modern technical means make it possible to present texts in various formats, and the current level of linguistic knowledge enables us to classify and index linguistic data, which allows us to present the collected material in a complex way, i.e. in the form of multimedia marked-up corpora and dictionary databases. The language features identified in this way demonstrate the system and structural, communicative and functional characteristics of the language.

An urgent task for Tatar morphology is the study of parts of speech as a continuum consisting of core and peripheral elements, which combine different word orders based, first of all, on their functional significance. The research will provide an opportunity to identify the process of diffusion and mutual transition between classes of words of certain speech parts. In the Tatar language, the problem of identifying a separate group of words is still controversial, for example, words like altyn 'gold' – altyn baldak 'golden ring', tash 'stone' – tash kuper 'stone bridge', agach 'tree' – agach өy 'wooden house', etc. Thus, the study using the LingvoDoc toolkit will allow us to solve not only the most confusing problems of the modern theory of part-of-speech attribution of words in the Tatar language, but also it will give an opportunity to present our view on solving such problems.

2.3 Colloquial speech

Created electronic corpora of Tatar fiction, colloquial speech and official business texts with full gloss and removed homonymy, as well as audio dictionaries of spontaneous colloquial speech with transcription in IPA, audio recordings of contexts will make it possible to determine the conceptual and semantic potential and functional features of language units, both in individual subsystems of the language, and in the language as a whole. So, for example, data on the functioning and frequency of individual classes of words and lexemes that function in different subsystems of the language allow solving some controversial issues in the theory of parts of speech, such as mutual transition 'noun-adjective', 'adjective-adverb', modal words, and onomatopoeic words in the Tatar language. These results will clarify theoretical issues in the field of grammar and word formation of the Tatar language.

Research models of the vocabulary system of the language are designed to model and reflect the real relationship between the functional-semantic types of words in all the variety of connections between them. Research on the LingvoDoc platform opens up opportunities to combine all elements: nominative, connective, etc. because these units have a functional value.

One of the promising tasks is to present a separate corpus of texts of Tatar spontaneous colloquial speech, characteristic of different social and age groups living in rural and urban areas.

As is known, the national language manifests itself in literary forms, in territorial and social dialects, and in vernacular, they all interact and meet various communicative needs of society. Modern Tatar linguistics practically doesn't have any comprehensive studies of the spoken language based on these territorial and social dialects and colloquial forms. At the present stage, the influence of dialects and forms of the spoken language in the literary language and its norms has sharply increased in the Tatar language. As part of the project, we intend to identify the main forms and features of the spoken language (phonetic, lexical, semantic, derivational, and grammatical) and study their implementation in the communicative process of individual social groups.

"The need to study Tatar colloquial speech with all its specific subsystems, structural and content diversity is due to several factors. Let's look at some of them. In modern society, there is a tendency to spread Tatar colloquial speech at home, in the media and electronic communication. There is a noticeable tendency for elements of an uncodified language to penetrate into the literary language. In the context of the integration of languages and dialects, there is a need for a systematic and comprehensive study of the Tatar spoken language in order to reveal the specifics of modern Tatar

speech, determine the boundaries of the territorial and social dialects of the Tatar language, identify the mechanisms of mutual influence and diffusion of the speech characteristics of the ethnicities living in the Republic of Tatarstan" [5: 5]. Spoken language will be presented in transcription and audio or video dubbing, which will give the oral corpus an accurate reproduction format. The inclusion of oral speech in the form of corpus texts will open up opportunities for developing theoretical problems of the current state of the Tatar language and determining the prospects for development in the context of globalism. All results will be available on the LingvoDoc platform.

It is known, the norm of the codified language acts as a regulator of the interaction between literary forms of the language and dialects. However, the norm itself is subject to changes over time and depends on the nature of the processes that take place in society. Society development is not only reflected in the language, but also has a significant impact on the further evaluation of its forms and elements. Unfortunately, traditional studies of the grammatical and lexical structure are behind the changes in modern colloquial speech. Because the volume and specificity of the material presented in this kind of research may not be full-scale.

3. Conclusion

Electronic corpora make it possible to fully reflect the language processes in different subsystems of the language. Thus, we will be able to predict in time the further development of the structure and possible changes in the norms of the codified Tatar language.

References / Список литературы

[1] Baranov A.N. Introduction to Applied Linguistics: Learning Guide. 3rd ed. Moscow, LKI, 2007, 360 p. (in Russian) / Баранов А.Н. Введение в прикладную лингвистику: учебное пособие. 3-е изд. М., ЛКИ, 2007 г., 360 стр. /

[2] Zubov A.V., Zubova I.I. Information technology in linguistics: Learning Guide. Moscow, Academy, 2004, 208 p. (in Russian) / Зубов А.В., Зубова И.И. Информационные технологии в лингвистике: учебное пособие. М., Академия, 2004 г., 208 стр.

[3] Bolshakov I A., Gelbukh A. Computational Linguistics. Models, Resources, Applications. Mexico, IPN-UNAM-FCE, 2004, 186 pp.

[4] Normanskaja J.V. Frst turkic cyrillic books on the LingvoDoc platform. Native languages and Cultures in the modern changing world, issue 1, 2022, pp. 43-57 / Норманская Ю.В. Первые тюркские кириллические книги на платформе ЛингвоДок. Родные языки и культуры в современном изменяющемся мир, вып. 1, 2022 г., стр. 43-57.

[5] Galiullina G.R., Kadirova E.Kh., Khadieva G.K. Modern Tatar colloquial speech: identification features and social differentiation. Kazan, Kazan University Publishing House, 2022, 222 p. (in Russian) / Галиуллина Г.Р., Кадирова Э.Х., Хадиева Г.К. Современная татарская разговорная речь: идентификационные признаки и социальная дифференциация. Казань, изд-во Казанског университета, 2020 г,, 222 стр.

Information about authors / Информация об авторах

Fanuza Shakurovna NURIEVA, Doctor of Philology, Professor, Chief Researcher of ISP RAS, Professor of Kazan Federal University. Research interests: Tatar language, turkology, dialectology, comparative and historical linguistics

Фануза Шакуровна НУРИЕВА, доктор филологических наук, профессор, главный научный сотрудник ИСП РАН, профессор КФУ. Научные интересы: татарский язык, тюркология, диалектология, сравнительно-историческое языкознание

Gulshat Raisovna GALIULLINA, Doctor of Philology, Professor, Head of the Department. Research interests: Tatar language, languages of the peoples of the Russian Federation, lexicology, semasiology, onomastics, linguoculturology, sociolinguistics, ethnic culture

Гульшат Раисовна ГАЛИУЛЛИНА, доктор филологических наук, профессор, заведующий кафедрой. Область научных интересов: татарский язык, языки народов РФ, лексикология, семасиология, ономастика, лингвокультурология, социолингвистика, этническая культура.

Airat Faikovich YUSUPOV, Doctor of Philology, Associate Professor. Research interests: Philology, literary criticism, textual criticism, semiotics, Islamic studies.

Айрат Фаикович ЮСУПОВ, доктор филологических наук, доцент. Сфера научных интересов: филология, литературоведение, текстология, семиотика, исламоведение.

DOI: 10.15514/ISPRAS-2022-34(6)-14

Оценка языковой способности нейронных моделей на материале предикативного согласования в русском языке

К.А. Студеникина, ORCID: 0000-0002-4098-7167 <xeanst@gmail.com>
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Исследование нацелено на то, чтобы изучить способность нейронных сетей моделировать грамматику, которая проявляется в функции автоматической оценки грамматичности языковых выражений. В данной работе моделируются правила предикативного согласования по числу в русском языке. Для обучения языковых моделей был создан датасет, включающий искусственно сгенерированные грамматичные и неграмматичные предложения. Мы используем трансферное обучение предобученных нейронных сетей. Результаты показывают, что все рассмотренные модели демонстрируют высокие результаты при дообучении на задачу оценки грамматичности. Точность классификации снижается для предложений с неодушевленными существительными, поскольку для них, в отличие от одушевленных существительных, наблюдается совпадение форм именительного и винительного падежа. Сложность синтаксической структуры оказывается значимой для русскоязычных моделей и модели для славянских языков, но не влияет на распределение ошибок для мультиязычных моделей.

Ключевые слова: языковая способность; языковые модели; трансферное обучение; искусственный интеллект; обработка естественного языка; оценка грамматичности

Для цитирования: Студеникина К.А. Оценка языковой способности нейронных моделей на материале предикативного согласования в русском языке. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 179-184. DOI: 10.15514/ISPRAS-2022-34(6)-14

Благодарности. Исследование выполнено при финансовой поддержке Некоммерческого Фонда развития науки и образования «Интеллект».

Evaluation of neural models' linguistic competence: evidence from Russian predicate agreement

K.A. Studenikina, ORCID: 0000-0002-4098-7167 <xeanst@gmail.com>
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. This study investigates the linguistic competence of modern language models. Artificial neural networks demonstrate high quality in many natural language processing tasks. However, their implicit grammar knowledge remains unstudied. The ability to judge a sentence as grammatical or ungrammatical is regarded as key property of human's linguistic competence. We suppose that language models' grammar knowledge also occurs in their ability to judge the grammaticality of a sentence. In order to test neural networks' linguistic competence, we probe their acquisition of number predicate agreement in Russian. A dataset consisted of artificially generated grammatical and ungrammatical sentences was created to train the language models. Automatic sentence generation allows us to test the acquisition of particular language phenomenon, to detach from vocabulary and pragmatic differences. We use transfer learning of pre-trained neural networks. The results show that all the considered models demonstrate high accuracy and Matthew's correlation coefficient values

which can be attributed to successful acquisition of predicate agreement rules. The classification quality is reduced for sentences with inanimate nouns which show nominative-accusative case syncretism. The complexity of the syntactic structure turns out to be significant for Russian models and a model for Slavic languages, but it does not affect the errors distribution of multilingual models.

Keywords: linguistic competence; language models; transfer learning; artificial intelligence; natural language processing; grammaticality judgments.

For citation: Studenikina K.A. Evaluation of neural models' linguistic competence: evidence from Russian predicate agreement. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 179-184 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-14

Acknowledgements. This work was supported by Non-commercial Foundation for support of Science and Education «INTELLECT».

1. Введение

Современные нейронные сети демонстрируют уровень обработки естественного языка, сравнимый с человеческим. Появляется все больше новых языковых моделей различной архитектуры, в связи с чем возникает необходимость их сравнения. В науке о данных уделяется большое внимание созданию так называемых бенчмарков – ресурсов для обучения, оценки и анализа языковых моделей. Подобный бенчмарк существует и для русского языка под названием Russian SuperGLUE [1]. Он включает в себя преимущественно такие задачи, которые оценивают понимание смысла текста: это, например, разрешение семантической неоднозначности [2], логический вывод по тексту [3], поиск ответа на вопрос в тексте [4].

Тем не менее, остается открытым вопрос о том, обладают ли нейронные сети имплицитным знанием грамматики. Умение оценивать грамматическую правильность предложения на родном языке является ключевым свойством языковой способности человека [5]. Суждения о правильности языкового выражения получили название «оценок грамматичности/ приемлемости». В лингвистике суждения о грамматичности, порожденные людьми, выступают как эмпирическая база для моделирования языковой способности человека. Следовательно, возможно использовать автоматические суждения о грамматичности для оценки языковой способности нейронных моделей.

Предшествующие исследования в данном направлении выполнены преимущественно на материале английского языка, для русского языка работ значительно меньше. Первое подобное исследование решает задачу автоматической оценки приемлемости в русском языке на основе корпуса предложений из лингвистических статей [6], что вызывает две проблемы. Во-первых, данные не содержат информацию, какой тип ошибки наблюдается в предложении, что делает невозможной детальную оценку языковой способности модели. Во-вторых, используется бинарная классификация, однако примеры иллюстрируют сложные феномены, которые не могут быть однозначно оценены как грамматичные или неграмматичные.

Задача нашего исследования состоит в том, чтобы изучить способность нейронных сетей к моделированию грамматики на материале конкретного феномена: функции автоматической оценки грамматичности языковых выражений на материале русского языка. Для исследования будут использованы искусственно сгенерированные данные, не содержащие пограничных случаев, что делает возможной бинарную классификацию по грамматичности. Проведенные исследования и код с результатами обучения представлены по ссылке: https://github.com/Xeanst/Grammaticality-judgements-with-neural-language-models.

2. Автоматическая генерация данных

Цель исследования состоит в том, чтобы оценить, насколько языковые модели обладают знаниями грамматики. Поскольку нейросеть должна разграничить граммматичные и неграмматичные предложения, обучающие и тестовые данные содержат не только

корректные примеры, но и примеры с ошибками. В качестве датасета могут быть использованы искусственно сгенерированные предложения [7, 8]. Данный метод позволяет самостоятельно определять, ошибка на какое языковое явление будет содержаться в предложении. Однако сами примеры будут несколько неестественными. В нашем исследовании мы осуществили искусственную генерацию предложений. Поскольку нам необходимо понять, насколько хорошо модель «понимает» устройство грамматики языка, некоторая неестественность предложений поможет абстрагироваться от лексических и прагматических аспектов и оценить именно знание грамматики.

Было сгенерировано 700 примеров. Мы исследовали, насколько хорошо модель способна усвоить механизм согласования по числу подлежащего и сказуемого: в грамматичных примерах число подлежащего и сказуемого совпадает (1), в неграмматичных – отличается (2). Для каждого предложения сгенерировано несколько вариантов, содержащих одни и те же лексические единицы и отличающиеся ровно двумя параметрами, числом подлежащего и числом сказуемого. Это позволяет определить, насколько хорошо языковая модель усваивает целевое грамматическое явление – предикативное согласование.

(1) a. Студент_ рассуждал_.
    b. Студенты рассуждали.
(2) a. Студент_ рассуждали.
    b. Студенты рассуждал_.

Мы рассматривали два лингвистических аспекта, которые могут осложнить автоматическое моделирование правил предикативного согласования. В первую очередь, это параметр одушевленности: одна половина примеров содержала одушевленные существительные, для которых формы именительного и винительного падежа различаются (3), другая – неодушевленные, где падежные формы совпадают (4). В первом случае модель должна была показать высокое качество классификации, однозначно определить подлежащее и дополнение предложения и правильно предсказать, в каких предложениях предикативное согласование корректно, а в каких – ошибочно. Во втором случае мы ожидали более низкое качество предсказания, поскольку по падежной форме дополнения нельзя определить, в каком падеже оно стоит – в именительном или винительном.

(3) a. Студент_ знал_ охранников.
    b. Студент_ знали охранников.
    c. Охранников знал_ студент_.
    d. Охранников знали студент_.
(4) a. Студент_ знал_ рассказы.
    b. Студент_ знали рассказы.
    c. Рассказы знал_ студент_.
    d. Рассказы знали студент_.

Помимо деления по одушевленности, данные содержали примеры различной степени сложности. Если в примере есть несколько существительных или несколько глаголов, то модели будет сложнее понять, грамматично ли предложение. Если же в предложении имеется только одно существительное и только один глагол, модель должна легко понимать, правильно ли указано число глагола. Всего данные содержали 7 типов синтаксических структур (по 10 предложений на каждый тип): простое предложение с подлежащим без дополнения (1-2), простое предложение с подлежащим и дополнением в прямом порядке (3a-b, 4a-b), простое предложение с подлежащим и дополнением в обратном порядке (3c-d, 4c-d), сложноподчиненное предложение (5), простое предложение с зависимым существительным при подлежащем (6), сложное предложение с субъектным зависимым предложением при подлежащем (7), сложное предложение с объектным зависимым предложением при подлежащем (8).

(5) a. Писатели говорили, что студенты рассуждали.

b. Писатели говорили, что студенты рассуждал_.

(6) a. Студент_ рядом с писателями рассуждал_.

b. Студент_ рядом с писателями рассуждали.

(7) a. Студент_, который знал_ охранников, рассуждал_.

b. Студент_, который знал_ охранников, рассуждали.

(8) a. Студент_, которого знали охранники, рассуждал_.

b. Студент_, которого знали охранники, рассуждали.

Таким образом, благодаря искусственной генерации примеров, удалось получить объемный сбалансированный датасет.

3. Обучение и тестирование моделей

В обучающей и тестовой выборке грамматичные предложения имели метку «1», предложения с ошибкой – метку «0». Следовательно, задача оценки грамматичности сводится к задаче бинарной классификации. В исследовании мы использовали трансферное обучение. Заранее обученные языковые модели с помощью полученного датасета дообучались на задачу классификации по грамматичности. Данные были разделены в следующем соотношении: для обучения – 80%, для валидации – 10%, для тестирования – 10%. При обучении количество эпох было равно 5, использовался графический процессор. Мы использовали как модели для русского языка: ruBERT [9] и Russian RoBERTa [10], так и мультиязычные модели: multilingual BERT (104 языка) [11], Slavic BERT (4 языка) [12], XLM RoBERTa (15 языков) [13].

В табл. 1 представлены общие результаты тестирования моделей в задаче классификации по грамматичности. Для оценки эффективности работы классификационного алгоритма используются такие метрики, как точность (accuracy) и коэффициент корреляции Мэтьюса (Matthews Correlation Coefficient, MCC). Можно заметить, что модели показывают довольно высокое качество. Наиболее высокие результаты демонстрируют модели на основе архитектуры BERT: модель для русского языка ruBERT, мультиязычная модель multilingual BERT и модель для славянских языков SlavicBERT. Более низкие результаты достигаются на основе архитектуры Roberta: модель для русского языка Russian RoBERTa и мультиязычная модель XLM Roberta.

Табл. 1. Общие результаты тестирования моделей
Table 1. General results for models testing

№	Модель	Точность	Коэффициент корреляции Мэтьюса
1	ruBERT multilingual BERT	98.6	0.97
2	Slavic BERT	97.1	0.94
3	Russian RoBERTa	77.1	0.54
4	XLM Roberta	57.1	0.33

Проведем лингвистический анализ результатов и подробнее рассмотрим точность классификации в зависимости от одушевленности существительных и синтаксической сложности. В табл. 2 представлены значения точности классификации для предложений с одушевленными и неодушевленными существительными. Результаты показывают, что все языковые модели оказались восприимчивы к совпадению падежных форм и продемонстрировали больше ошибок при определении корректной согласовательной стратегии для неодушевленных существительных, чем для одушевленных. Модели ruBERT, multilingual BERT и Slavic BERT ошибочно предсказали оценку только для предложений с неодушевленными существительными. Модели Russian RoBERTa и XLM Roberta также допустили большее количество ошибок на предложениях с неодушевленными существительными.

Табл. 2. Точность классификации при тестировании моделей на предложениях с одушевленными и неодушевленными существительными
Table 2. Classification accuracy of models testing on sentences with animate and inanimate nouns

Существительное	ruBERT	multilingual BERT	Slavic BERT	Russian RoBERTa	XLM Roberta
Одушевленное	100.0	100.0	100.0	80.0	62.9
Неодушевленное	97.1	97.1	94.3	74.3	51.4

В табл. 3 продемонстрирована точность классификации для предложений различной синтаксической сложности. Для некоторых моделей, действительно наблюдается больше ошибок для предложений со сложной синтаксической структурой: для сложноподчиненного предложения (тип 4, модель ruBERT), для простых предложений с обратным порядком подлежащего и дополнения (тип 3, модель Slavic BERT и модель Russian RoBERTa), для сложных предложений с объектным зависимым при подлежащем (тип 7, также модель Russian RoBERTa). В то же время, другие модели допускают ошибки вне зависимости от синтаксической сложности структуры: они демонстрируют наибольшее количество ошибок для простых предложений с подлежащим (тип 1, модели multilingual BERT и XLM Roberta).

Табл. 3. Точность классификации при тестировании моделей на предложениях с различной синтаксической структурой
Table 3. Classification accuracy of models testing on sentences with various syntactic structure

№	Тип предложения	ruBERT	multilingual BERT	Slavic BERT	Russian RoBERTa	XLM Roberta
1	Простое предложение с подлежащим без дополнения	100.0	90.0	100.0	90.0	30.0
2	Простое предложение с подлежащим и дополнением в прямом порядке	100.0	100.0	100.0	80.0	60.0
3	Простое предложение с подлежащим и дополнением в обратном порядке	100.0	100.0	80.0	60.0	50.0
4	Сложноподчиненное предложение	90.0	100.0	100.0	80.0	40.0
5	Простое предложение с зависимым существительным при подлежащем	100.0	100.0	100.0	70.0	70.0
6	Сложное предложение с субъектным зависимым предложением при подлежащем	100.0	100.0	100.0	100.0	80.0
7	Сложное предложение с объектным зависимым предложением при подлежащем	100.0	100.0	100.0	60.0	80.0

Наши ожидания относительно лингвистических аспектов, которые могут осложнять классификацию, подтвердились лишь частично. Для большинства моделей возникают трудности при оценке предложений с неодушевленными существительными, поскольку для них, в отличие от одушевленных существительных, наблюдается совпадение форм именительного и винительного падежа. Однако предположение относительно сложности структур подтвердилось не до конца. Синтаксическая структура оказывается значимой для

русскоязычных моделей и модели для славянских языков, но не влияет на распределение ошибок для мультиязычных моделей.

4. Заключение

В данной работе была проведена оценка языковой способности нейронных моделей на материале русского языка. Исследование показало, что при обучении на большом количестве неразмеченных данных модели некоторым образом выучивают грамматику. Полученный список моделей, ранжированный по точности классификации, имеет высокую технологическую ценность и может быть использован для выбора модели при решении различных задач обработки языка. Если модель хорошо справляется с автоматической оценкой грамматичности, она отлично покажет себя в таких задачах, как морфологический и синтаксический анализ.

Список литературы / References

[1] Shavrina T., Fenogenova A. et al. RussianSuperGLUE: A Russian language understanding evaluation benchmark. In Proc. of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 4717-4726.
[2] Panchenko A., Lopukhina A. et al. RUSSE'2018: a shared task on word sense induction for the Russian language. In Proc. of the International Conference "Dialogue 2018", 2018, pp. 547-564.
[3] De Marneffe M. C., Simons M., Tonhauser J. The CommitmentBank: Investigating projection in naturally occurring discourse. Proceedings of Sinn und Bedeutung, vol. 23, issue 2, 2019, pp. 107-124.
[4] Glushkova T., Machnev A. et al. DaNetQA: A Yes/No Question Answering Dataset for the Russian Language. Lecture Notes in Computer Science, vol. 12602, 2020, pp. 57–68.
[5] Chomsky N. Aspects of the theory of syntax. MIT Press, 1969, 251 p.
[6] Mikhailov V., Shamardina T. et al. RuCoLA: Russian Corpus of Linguistic Acceptability. In Proc. of the 2022 Conference on Empirical Methods in Natural Language Processing, 2022, pp. 5207-5227.
[7] Marvin R., Linzen T. Targeted syntactic evaluation of language models. In Proc. of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 1192-1202.
[8] Warstadt A., Parrish A. et al. BLiMP: The benchmark of linguistic minimal pairs for English. Transactions of the Association for Computational Linguistics, vol. 8, 2020, pp. 377-392.
[9] Kuratov Y., Arkhipov M. Adaptation of deep bidirectional multilingual transformers for Russian language. In Proc. of the International Conference "Dialogue 2019", 2019, pp. 333-339.
[10] Blinov P., Avetisian M. Transformer models for drug adverse effects detection from tweets. In Proc. of the Fifth Social Media Mining for Health Applications Workshop & Shared Task, 2020, pp. 110-112.
[11] Devlin J., Chang M.W. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language, 2018, pp. 4171-4186.
[12] Arkhipov M., Trofimova M. et al. Tuning multilingual transformers for language-specific named entity recognition. In Proc. of the 7th Workshop on Balto-Slavic Natural Language Processing, 2019, pp. 89-93.
[13] Lample G., Conneau A. Cross-lingual language model pretraining. In Proc. of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), 2019, 11 p.

Информация об авторах / Information about authors

Ксения Андреевна СТУДЕНИКИНА является программистом лаборатории автоматизированных лексикографических систем НИВЦ МГУ и аспирантом кафедры теоретической и прикладной лингвистики филологического факультета МГУ. Её научные интересы включают автоматическую обработку естественного языка и исследование синтаксиса русского языка.

Kseniia Andreevna STUDENIKINA is a programmer in the Laboratory for Computational Lexicography of Research Computing Center of MSU and PhD student at the Department of Theoretical and Applied Linguistics of Philological Faculty of MSU. Her research interests include natural language processing and studies of Russian syntax.

DOI: 10.15514/ISPRAS-2022-34(6)-15

Effect of relative longitudinal spacing on the dynamic behavior of two interacting ships in head waves

R. Ali, ORCID: 0000-0003-0591-6221 <ramimamdouhali@gmail.com>

State Marine Technical University (SMTU),
Lotsmanskaya st.3, Saint-Petersburg, 190121, Russia

Abstract. The influence of relative longitudinal position on the frequency characteristics of two interacting ships floating stationary in close proximity in head waves and shallow water is investigated in this paper. A CFD approach has been adopted to simulate the dynamic behavior of the interacting ships. The numerical simulation for "Aleksey Kosygin" and "Novgorod" ships floating on head waves at a small relative transverse distance $\boldsymbol{\eta = 1.3}$ was carried out using two scaled models. The effect of longitudinal separation on the frequency characteristics of both ships was studied. Heave, roll, and pitch RAOs for various cases were analyzed, and recommendations for relative longitudinal positions were made on the basis of the present analysis.

Keywords: Lightering operation; Hydrodynamic Interaction; ship motions; RANS; CFD; OpenFOAM

For citation: Ali R. Effect of relative longitudinal spacing on the dynamic behavior of two interacting ships in head waves. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 185-196. DOI: 10.15514/ISPRAS-2022-34(6)-15

Влияние относительного продольного расстояния на динамическое поведение двух взаимодействующих судов при встречном волнении

Рами Али, ORCID: 0000-0003-0591-6221 <ramimamdouhali@gmail.com>

Санкт-Петербургский Государственный морской технический университет,
190121, Россия, Санкт-Петербург, ул. Лоцманская, 3

Аннотация. В работе исследуется влияние взаимного продольного положения на частотные характеристики двух взаимодействующих судов, плавающих неподвижно в непосредственной близости на встречном волнении и мелководье. Подход CFD был принят для моделирования динамического поведения взаимодействующих судов. Численное моделирование судов «Алексей Косыгин» и «Новгород», плавающих на встречных волнах при малом относительном поперечном расстоянии η=1,3, проводилось с использованием двух масштабных моделей. Исследовано влияние продольного расстояния на частотные характеристики обоих судов. Были проанализированы АЧХ вертикальной, килевой и бортовой качки для различных случаев, и на основе настоящего анализа были даны рекомендации по относительному продольному положению.

Ключевые слова: лихтеровка; гидродинамическое взаимодействие; качка судов; RANS; CFD; OpenFOAM

Для цитирования: Али Р. Влияние относительного продольного расстояния на динамическое поведение двух взаимодействующих судов при встречном волнении. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 185-196. DOI: 10.15514/ISPRAS-2022-34(6)-15

1. Introduction

Nowadays, there is a significant increase in marine applications that involve ships floating in very close proximity. Lightering operations and ship-tug maneuvering are considered typical examples of such investment operations involving multibody floating systems. Being in the waves complicates the hydrodynamic interactions of such a system since each body experiences the incident waves in addition to the diffracted and radiated wave systems of another body, which may result in unfavorable responses and lead to dangerous scenarios. Therefore, studying the motion responses of such a multi-body system in waves is extremely important.

Chongwei Zhang [1] studied the hydrodynamic interaction between two identical barges, one of which was fixed and the other was allowed to sway freely in head waves; the obtained results indicated that amplitudes of the barge motion and maximum wave elevation in the gap generally increased with the reduction in the gap width. V. Yu. Semenova and Aung Myo Thant [2] studied the influence of lateral separation between ships on the damping and added mass coefficients for two interacting ships in shallow water. Zhi-Ming Yuan [3] developed a 3-D panel code based on the Rankine source method to investigate the hydrodynamic interaction between two ships in shallow water. Yuan studied the influence of the lateral distance between two Wigley III hulls on the RAO "Response Amplitude Operator", and the obtained results indicated that the resonant frequency is greatly influenced by the gap between the ships. Yoshiyuki Inoue and Mir Tareque Ali [4] used the far-field approach to investigate the numerical accuracy of the computation of multi-body systems floating in deep water on regular waves. In their research, the numerical calculations were carried out for a parallelly connected FPSO "Floating Production Storage and Offloading" and LNG "Liquefied Natural Gas" carrier for different wave heading angles and for various separation distances between these ships. It was found that some of the motion results particularly roll and yaw motion exhibit considerable amount of deviation from the experimental one. Yoshiyuki Inoue and Mir Tareque Ali [5] extended their research using the 3-D source-sink method to study the hydrodynamic interactions and dynamic behavior of two rectangular barges freely floating in deep water. Effects of beam and head sea conditions were considered, the lateral distance between the barges had also been varied and its influence on the ship motions had been studied. Binbin Li [6] studied the hydrodynamic interaction of side-by-side ships under different wave headings. His work concentrated on the resonant characteristics of parallel and nonparallel configurations, and the obtained results indicated that the higher resonant mode shifts to a lower frequency in the nonparallel configuration.

Ohkusu [7] extended the classical solution for a single heaving circular cylinder to the case of two cylinders in a catamaran configuration. Faltinsen and Michelsen [8] used the panel method to simulate the wave effects on 3D floating bodies. The panel method was further extended for two independent bodies by van Oortmerssen [9]. Ali and Tryaskin [10,11] used the CFD "Computational Fluid Dynamics" method to investigate the hydrodynamic interaction of two bodies in deep water.

In this paper, the fluid flow around two close ships floating in close proximity on head waves in shallow water was simulated. A wide range of longitudinal spans were tested, and their effects on the frequency characters were obtained.

The paper starts by defining the problem in section 2, which includes a brief description of the geometrical and operational characteristics of the "Aleksey Kosygin" and "Novgorod" hulls in addition to the governing equations and turbulence modeling approaches. A detailed review of the computational domain, mesh specifications, numerical settings, and boundary conditions is described in section 3. Validation with experimental data is subsequently performed. Finally, results and discussion of the effects of relative longitudinal position on the frequency characters of two interacted ships are discussed in section 4, conclusions are presented in section 5.

2. Problem definition

2.1 Hull form

The ships under consideration are the lightering ship "Aleksey Kosygin" and the general cargo ship "Novgorod"; the displacement of both ships is 53000 tons and 17000 tons, respectively. In this study, two scaled models with a scale factor of k=0.0177 are used: the "Aleksey Kosygin" model plays the role of the ship to be lightered (STBL), while the "Novgorod" model plays the role of the service ship (SS). The hull shape and the principal dimensions and coefficients of both ships are shown in Table 1 and Fig. 1, respectively.

Table. 1. Main particulars of the "Aleksey Kosygin" and "Novgorod" ships

	LBP	*B*	*T*	*CB*
Aleksey Kosygin	*232.0*	*32.0*	*10.6*	*0.660*
Novgorod	*138.0*	*20.6*	*9.0*	*0.661*

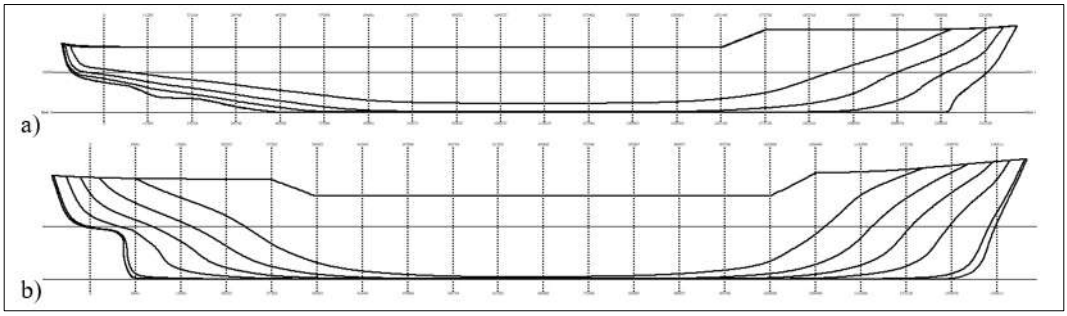

Fig. 1. Geometry of the interacting ships a) "Aleksey Kosygin" and b) "Novgorod "ships

2.2 Axis Conventions

Three sets of right-handed coordinate systems are used; an earth-fixed coordinate system S (x_0; y_0; z_0) with origin located at the still water surface and x_0 axis points to the direction of wave propagation, a fixed coordinate system O (x; y; z) linked to each ship, its origin is located at the intersection of the center-plane, midship plane, and water-plane, with X axis points to the ship bow, a body bounded coordinate system G (x_b; y_b; z_b) connected to the center of gravity G of each ship with x_b axis oriented to the ship bow.

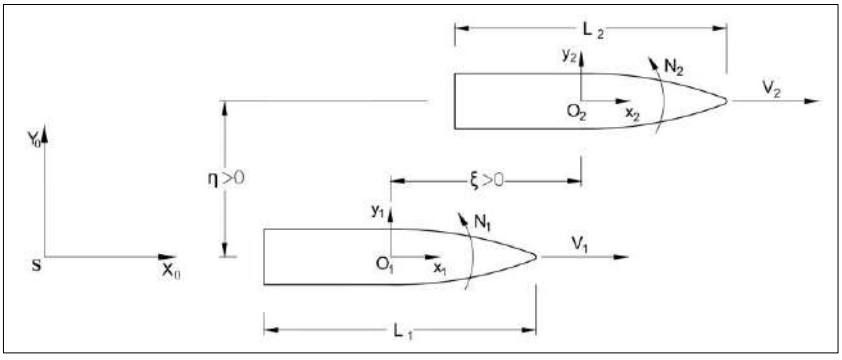

Fig. 2. Coordinate systems used for two interacting ships in waves

The vertical axes in all coordinate systems are directed upward. The fixed coordinate system is used to define the relative positions of each ship to another, as well as their linear and rotational motions. The corresponding right-handed coordinates are shown in Fig. 2.

2.3 The non-dimensionality terms

In this study, both ships float parallel on regular head waves without forward speed in shallow water $\frac{h}{T_{STBL}} = 1.8$, where h is the water depth [m]. The SS is always located at the starboard side of the STBL and its position is defined by the origin location of its fixed coordinate system in the STBL fixed coordinate system. To clearly describe the case and to reduce the number of variables associated with it, the relative position of the two ships is defined non-dimensionally as follows:

- The non-dimensional lateral distance (η), $\eta = dy/B_{avg}$ where dy is the lateral distance between the centerlines of the ships, B_{avg} is the average molded breadth of the SS and STBL;
- The non-dimensional longitudinal distance (ξ), $\xi = dx/L_{avg}$ where dx is the longitudinal distance between the mid-sections of the ships, L_{avg} is the average length between perpendiculars of the SS and STBL.

When the interacted ships float in symmetry position, their midship planes coincide and so the ξ=0.

The linear and rotational amplitudes of the heave, roll, and pitch motions of both ships will be computed and displayed in a dimensionless form as follows:

$$RAO_{heave} = \frac{F_3}{\zeta_a}, RAO_{roll} = \frac{F_4}{2\pi\zeta_a/L_{avg}}, RAO_{pitch} = \frac{F_5}{2\pi\zeta_a/L_{avg}}$$

where $F_3, F_4\ and\ F_5$ are the model response in the corresponding direction, ζ_a is the amplitude of incident waves[m].

During the study, the lateral distance η was kept constant and equal to 1.3, while the longitudinal distance ξ was varied to cover the following values: 0%, 15%, 30%, 50%, and100%.

The longitudinal distance ξ is used to indicate the position of the SS relative to the STBL. So, the relative positions of SS are classified into two main categories: front and rear positions. In the front positions, the service ship SS is shifted forward compared with the symmetry position (alongside position), so the nondimensional longitudinal positions will have a positive value $\xi > 0$, while in the rear positions, the service ship SS is displaced oppositely, and so $\xi < 0$.

3. Governing Equations, Turbulence Model

Continuity and Navier–Stokes equations are the basic equations governing the fluid flow, to describe the turbulent of viscous fluids with a logical computational resource consumption, the Reynolds average is applied over the continuity and Navier–Stokes equations, PDEs of RANS are described as follow:

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0, \tag{1}$$

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \bar{P}}{\partial x_j} + \vartheta \frac{\partial^2 \bar{u}_i}{\partial x_j \partial x_j} - \frac{\partial \overline{\acute{u}_i \acute{u}_j}}{\partial x_j} + g_i + F_{\sigma i} \tag{2}$$

where P is the pressure, u_i and g_i are the velocity and the gravitational acceleration components in the cartesian coordinate system, $\overline{\acute{u}_i \acute{u}_j}$ is Reynolds stress, ρ is the fluid density, ϑ is the kinematic viscosity and $F_{\sigma i}$ is the surface tension force.

The air-water interface is captured using the Volume of Fluid (VOF) method.

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (U\alpha) = 0 \tag{3}$$

where: α is the volume fraction, t denotes time, U is the fluid velocity.

RANS equations are closed with two equations $SST\ k{-}\omega$ turbulence model presented by Menter [12]. Menter Shear Stress Transport Turbulence Model $SST\ k{-}\omega$ is a two-equation eddy-viscosity model used for many hydrodynamic and aerodynamic applications, this model combines the well-known low Reynolds turbulence model $k{-}\omega$ and high Reynolds turbulence model $k{-}\varepsilon$, the former

is suitable for simulating flow in the viscous sub-layer, while the latter is ideal for predicting flow behavior in regions away from the wall, the built-in blinder of $SST\ k{-}\omega$ ensures the appropriate use of these models. The most recent version of the $SST\ k{-}\omega$ model consists of the following formulas:

$$\frac{\partial(\rho k)}{\partial t}+\frac{\partial(\rho u_i k)}{\partial x_i}=P_k-\beta^*\rho k\omega+\frac{\partial}{\partial x_i}\left[(\mu+\sigma_k\mu_t)\frac{\partial k}{\partial x_i}\right], \tag{4}$$

$$\frac{\partial(\rho\omega)}{\partial t}+\frac{\partial(\rho u_i\omega)}{\partial x_i}=\alpha\rho S^2-\beta\rho\omega^2+\frac{\partial}{\partial x_i}\left[(\mu+\sigma_\omega\mu_t)\frac{\partial\omega}{\partial x_i}\right]+2(1-F_1)\rho\sigma_{\omega^2}\frac{1}{\omega}\frac{\partial k}{x_i}\frac{\partial\omega}{\partial x_i} \tag{5}$$

where k is turbulent kinetic energy, μ dynamic viscosity, μ_t eddy viscosity, P_k production of turbulent kinetic energy, ω specific dissipation rate, S is the invariant measure of the strain rate, F_1 is a blending function and σ_k, β, β^*, σ_ω, σ_{ω^2} are closure coefficients.

4. Numerical Formulation

4.1 Computational Domain

During the study, a box-shaped domain was chosen to conduct the entire series of simulations. The boundaries of the domain were located in a way that coincided with the ITTC "International Towing Tank Conference" requirements for wave generation and absorption. CreatMesh.py, a home-built script, was used to prepare a structured hexahedral mesh and automate the procedure of adjusting the boundaries. During the grid creation, and to maintain the height of the propagating wave, 100 cells within the wavelength and 20 cells within wave height were adhered to.

For all the studied waves, the inlet and outlet patches were located at least at 2L and 5L, respectively, away from the nearest hull, as shown in Table 2. To simulate the shallow water case, the water phase was adjusted in a way that the criteria $h/\lambda < 1/2$ and $h/T \leq 2$ were achieved. The domain width was extended laterally by 3L, the air phase extended to $3L$ from the still water-free surface, where L and T are length and draft of STBL, h is water depth, λ is wave length. A general view highlighting different zones of the computational domains is shown in Fig. 3.

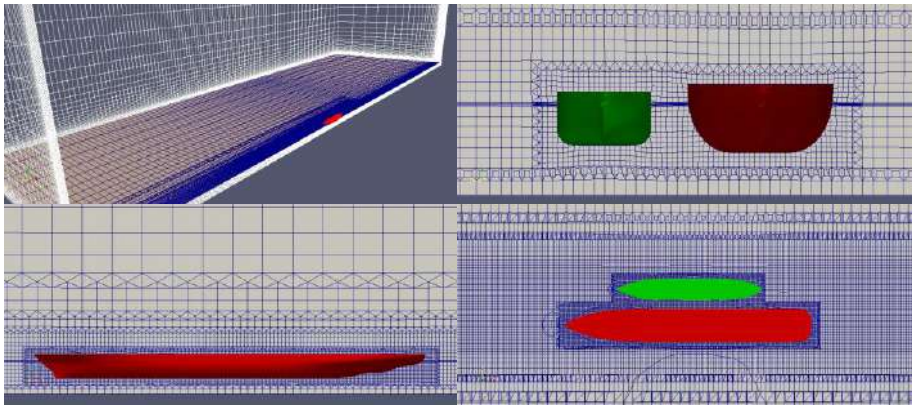

Fig. 3. Overview of computational domain and Grid structure around STBL and SS

Table 2. Inlet and outlet patches of the numerical wave tank

	$\lambda/L > 1$	$\lambda/L \leq 1$
Inlet ahead of front ship	$0.5L + 2\lambda$	$2L + 0.5\lambda$
Outlet behind the rear ship	$2L + 2\lambda$	$3L + 3\lambda$

4.2 Numerical settings

For complicated fluid flows, FVM is used to discretize the PDEs of the governing equations over the domain. in order to minimize the consumption of the computational resource, the discretized form of PDEs of RANS were solved using OpenFOAM CFD package.

Wave generating, propagating and absorbing in the NWT "Numerical Wave Tank" were conducted by waveFoam solver. The time discretization is based on the first-order Euler bounded scheme, and the spatial discretization is performed with a linear second-order central differencing scheme for the convection and diffusion terms.

The simulation was treated as unsteady. PIMPLE, a pressure-velocity coupling algorithm based on the finite volume method, was used. Piso algorithm was launched by eliminating the outer correctors, and pressure fields were corrected twice for each time step. Courant number was limited to $CFL \leq 0.25$, time step was left to OpenFOAM by turning on adjustTimeStep option. The total time of the simulation was adjusted to 12 wave periods. According to the results reported in [13], turbulence intensity Tu and eddy viscosity ratio μ_t/μ were set to 5 and 60, respectively.

The water condition was modeled as fresh water at 17°C, and the corresponding values of density ρ and kinematic viscosity nu are set to ρ=998.8 [kg/m^3] and 1.09×10^{-06} [m^2/sec], respectively.

The waveVelocity and waveAlpha boundary conditions set the values from respective wave models at the inlet of the CFD domain. No-Slip boundary condition was set to the stationary hull surfaces immersed in the viscous fluid. Pressure values on the walls were assumed to be fixed with zero gradients, and pressure equal to atmospheric pressure was set at the outlet of the flow domain. An appropriate wall functions for turbulent fields were set on hull surfaces.

5. Validations

To study the mesh convergence and investigate the discretization errors, three grids were created: a fine grid of 4265293 elements, a medium grid of 2760868 elements, and a coarse grid of 1461957 elements. In compliance with the requirements of wall functions, y+>30 has been adhered to for all grids. Smoothed grids have the same structural architecture, and cell sizes across subsequent grids are correlated according to a ratio of √2 in all domain regions; the cell-to-cell aspect ratio was kept at 1.2 in the stretching regions.

The heave RAO of the "Novgorod" vessel was evaluated on head waves using the three grids; the numerical results at wave period 2.79 seconds in model scale were compared with the published results of Semenova and Thant [2], and the relative error in the coarse, medium, and fine grids was 10.41%, 2.99%, and 1.07%, respectively. The true numerical solution was estimated using Richardson extrapolation and found to be 0.798. The grid convergence index (GCI%) was calculated for fine-medium grid and medium-coarse grid and found to be 0.83 and 3.15, respectively.

The grid numerical analysis indicates asymptotically approaching towards the converged solution. Based on the conducted study and to achieve a balance between computational accuracy and simulation time, the medium grid was chosen to perform the numerical simulation in this article.

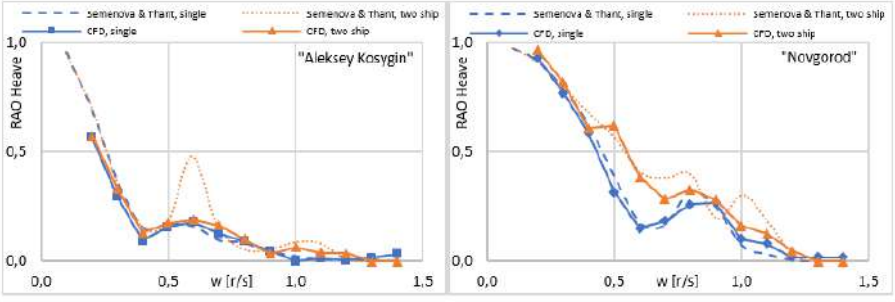

Fig. 4. Heave RAO for the two interacted ships, "Aleksey Kosygin" (left) and "Novgorod" (right)

However, Semenova and Thant's potential results for “Aleksey Kosygin” and “Novgorod” hulls, floating stationary on head waves and shallow water and located parallelly without longitudinal span at close lateral span η=1.3, were used as benchmarks to evaluate the CFD results [2].

The validations covered the heave RAO of a single ship as well as two interacting ships. The results of CFD simulations were illustrated in Fig. 4 against potential results, and they seem to be in good agreement which confirms the quality of the grid and the accuracy of the results obtained.

6. Results

The validation presented above provides a brief overview of the hydrodynamic interaction between two ships in close proximity; however, the frequency characteristics of two interacting ships are affected by their relative longitudinal positions. To clearly represent the results, the effect of the relative longitudinal position ξ on the ship's motions (heave, pitch, and roll) will be shown separately for each vessel, and a distinction will be made between the cases of forward and backward positioning of the SS with respect to the STBL. Calculations were performed for about two months on node of 20-cores, 3.00 GHz Intel Xeon CPU, 64 GB DDR3, Linux 4.19.0-21-amd64 (x86_64).

6.1 Effect of changing the relative longitudinal position on the heave motions of the interacting ships

Changing the relative longitudinal position ξ differently affects the amplitude of the vertical motions of each of the SS and STBL ships. It has a clear effect on the service ship SS, while its effect is limited on the STBL ship over the entire spectrum of the studied wave frequencies.

Fig. 5 and 6 show the effect of different relative longitudinal positions, including the symmetry position $\xi = 0$, on the heave motion of the two interacting vessels, SS and STBL.

The results obtained by the numerical simulation will be displayed and discussed according to the relative longitudinal position ξ of SS as follows:

6.1.1 Front positions

Figure 5 depicts the influence of SS forward positions on the heave's RAO of both SS and STBL.

For SS, the curves in Fig. 5 indicate that the effect of the forward positions on the heave motion is related to the wave frequency. The forward positions have a negligible effect at wave frequencies higher than $\omega = 1.2$ and a random effect within the frequency range $\omega \in [0.9,1.2]$. However, forward positions are beneficial and lead to a decrease in the heave motion, compared to symmetry position, at wave frequencies smaller than $\omega = 0.9$.

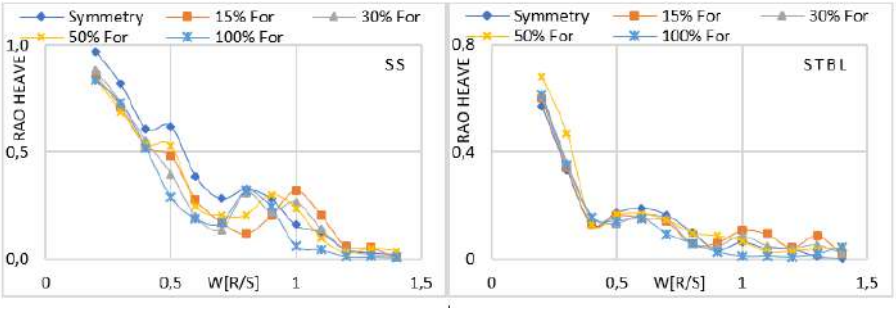

Fig. 5. RAO of heave motions for both SS and STBL ships at positive values of ξ

Changing the front longitudinal position of SS plays an effective role in reducing her vertical motions within the frequency range $\omega \in]0.4,0.9[$, while its effect is weak outside this range. The ability to reduce vertical motions by changing the forward relative position ranges from 14.8% at frequency 0.4 to 53.2% at frequency 0.5.

Regarding the STBL, the front positions of the service ship SS have a random negligible effect on the heave motion of the STBL. However, within the frequency range 0.5 to 0.8, the SS forward positions reduce STBL heave motions comparing to the symmetry position.

6.1.2 Rear positions

Shifting the SS backwards plays an effective role in reducing her vertical motion along a range of wave frequencies that extends from $\omega = 0.2$ to $\omega = 1.2$, Fig. 6.

The ability to reduce the SS vertical motions by altering her rear position ranges from 9.9% at frequency 0.4 to 53.8% at frequency 0.5.

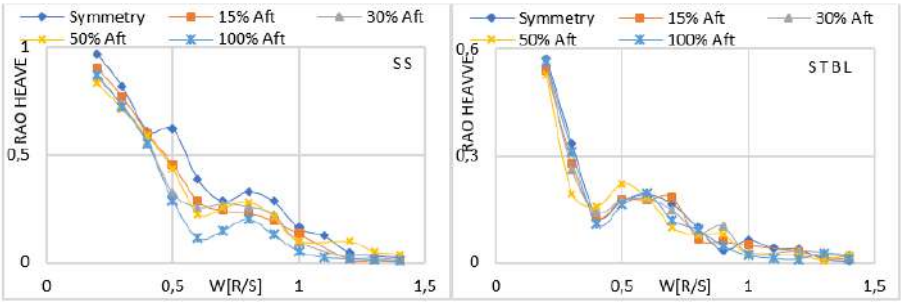

Fig. 6. RAO of heave motions for both SS and STBL ships at negative values of ξ

Regarding STBL, the rear positions of the service ship SS positively affect the vertical motions of the STBL for wave frequencies smaller than 0.4, while it has a random effect outside this range.

6.2 The effect of changing the relative longitudinal position on the pitch motions of interaction ships

The effect of the relative longitudinal position on the pitch motion was studied for two interacting ships floating stationary on head waves at a small transverse distance $\eta = 1.3$. The results are shown separately for front and rear positions in Fig. 7 and Fig. 8, respectively.

6.2.1 Front positions

The analysis of the numerical results shows that, in comparison with the symmetry position case, the forward position of the service ship SS slightly affects the pitch motions of each of the two interacting ships.

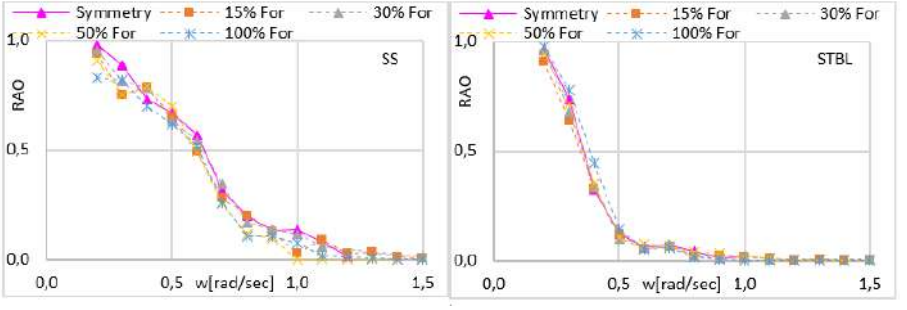

Fig. 7. RAO of Pitch motions for both SS and STBL ships at positive values of ξ

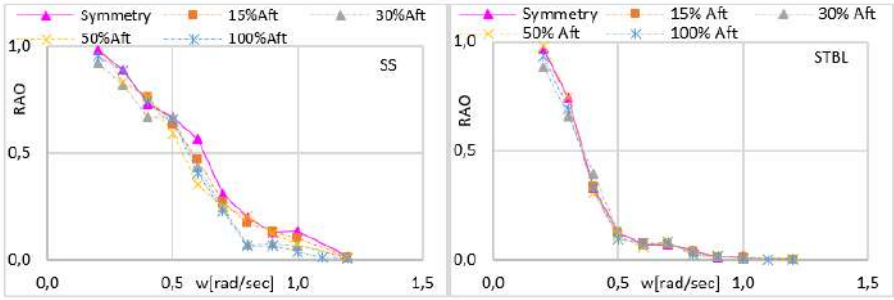

Fig. 8. RAO of pitch motions for both SS and STBL ships at negative values of ξ.

6.2.2 Rear positions

When it comes to rear positions, the curves presented in Fig.8 shows that, in comparison to the case of symmetry position $\xi = 0\%$, the rear positions of the service ship SS decrease her pitch motion while slightly affect the pitch motion of STBL. The positive effect of the rear position on the SS pitch motion appears clearly for wave frequencies that fall within the range $\omega \in [0.5, 1]$.

The curves of Fig. 8 confirm that a rear position of the service ship SS of $\xi = 50\%$ causes a reduction in her pitch motion by 11.4%, 37.9% and 21.5% for a wave frequency $\omega = 0.5,\ 0.6$ and 0.7 respectively.

6.3 The effect of changing the relative longitudinal position on the roll motions of interacting ships

6.3.1 Front positions

Forward positions of the SS reduce the peak of roll motion of both SS and STBL ship when compared to the case of symmetry position. An exception is the forward position $\xi = 30\%$, which is associated with an increase in the roll peak of the STBL by 19.3%, Fig. 9.

The curves in Fig. 9 confirm that the shift of the SS to a forward position greater than $\xi = 50\%$ causes a clear reduction in the values of the roll motions of both ships over the entire range of the studied wave frequencies.

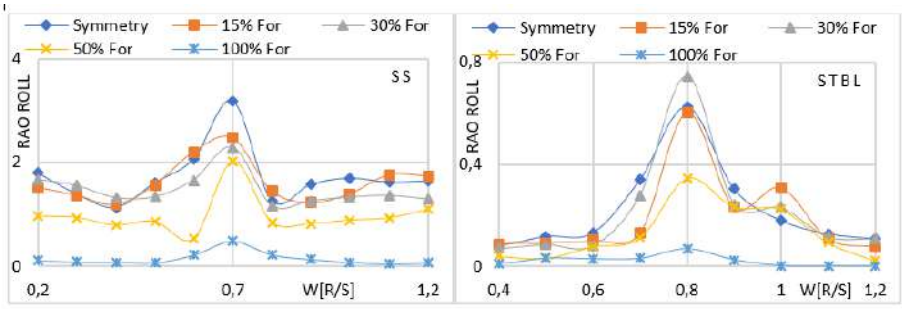

Fig. 9. RAO of roll motions for both SS and STBL ships at positive values of ξ.

In order to more clearly show the effect of the longitudinal position on the peak of roll motion, the change of peak values was represented in Fig. 11 as a function of the non-dimensional longitudinal position ξ for each ship.

Curves of Fig. 11 ensure that the forward relative longitudinal position $\xi \geq 50\%$ has a positive effect on both ships, $\xi = 50\%$ results in a 36.8% reduction in roll peak of the SS and 45% for the STBL.

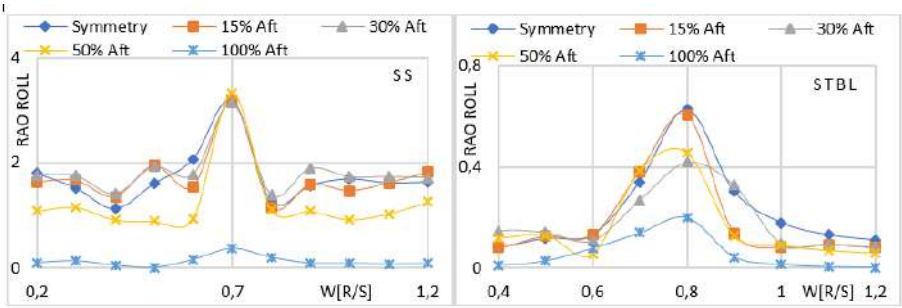

Fig. 10. RAO of roll motions for both SS and STBL ships at negative values of ξ.

6.3.2 Rear positions

Along a range of longitudinal positions extending from the symmetry position to $\xi = 50\%$ aft, the roll peak of the service ship SS showed a small influence, and its changes remained confined to the small range between -0.6% and 4% only. However, this is not the case for the STBL ship, where the relative longitudinal position $\xi = 30\%$ aft of SS is associated with a clear decrease in the STBL peak value up to 33%, Fig. 10 and Fig. 11.

Curves in Fig. 10 and Fig. 11 ensure that shifting the SS to a rear position $\xi > 50\%$ causes a clear and significant reduction in the roll motions of both ships.

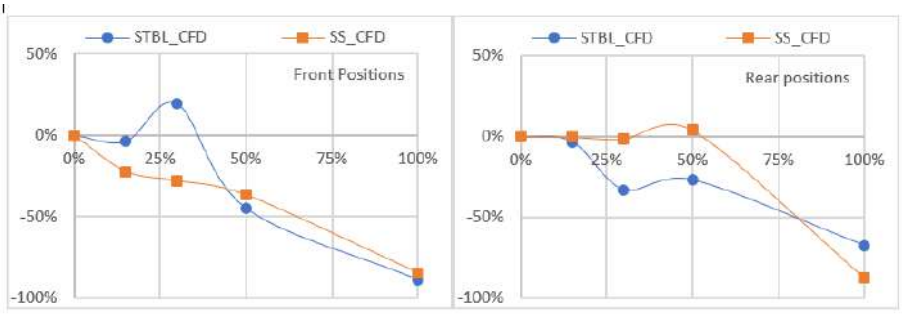

Fig. 11. The change of roll peak as a function of ξ for both SS and STBL

7. Conclusions and recommendations

OpenFOAM, an open-source CFD package, was used to simulate the motion response for two ships floating in close proximity in head waves and shallow water. RANS method was used for turbulence modeling and the well-known turbulent model k-ω SST was used to close RANS equations. The results of numerical simulation conducted on ships "Aleksey Kosygin" and "Novgorod" showed that the relative longitudinal position affects the RAO of the heave, pitch and roll motions of each of the two ships. The following results are obtained for ship length ratio 1.68, and lateral separations $\eta = 1.3$:

- The frequency characteristics of the ship motion are related not only to the frequency of the induced wave but also to the relative longitudinal position between the two ships.
- In general, the heave of STBL ship is less sensitive to the longitudinal change of position compared to the service ship SS. Shifting the service ship SS away from the symmetry position reduces her heave motion within the range of wave frequencies $\omega < 0.9$; maximum effectiveness takes place at frequencies $\omega \in]0.4,0.9[$.
- Changing the SS longitudinal position has a neglected effect on the pitch motion of the STBL ship. The rear positions decrease SS pitch motion when compared with the symmetry position.

- Forward and rear positions $\xi > 50\%$ ensures a reduction in the peak of roll motions for both ships.

References / Список литературы

[1] Zhang C., Sun X. et al. Hydrodynamics of a floating barge adjacent to fixed structure in transient wave fronts. Physics of Fluids, vol. 33, issue 10, 2021, article id. 107106, 49 p.

[2] Semenova V.Yu., Thant A.M. The investigation of the influence of the distance between the ships and the fairway depth on the hydrodynamics characteristics of their joint motions in shallow water. Research Bulletin by Russian Maritime Register of Shipping, issue 44/45, 2016, pp. 61-68 (in Russian) / Семенова В.Ю., Тхант А.М. Исследование влияния расстояния между судами и глубины фарватера на гидродинамические характеристики их совместной качки на мелководье. Научно-технический сборник Российского морского регистра судоходства, вып. 44/45, 2016 г., стр. 61-68.

[3] Yuan Z.-M., Incecik A., He S. Hydrodynamic Interaction Between Two Ships Arranged Side by Side in Shallow Water.In Proc. of the International Conference on Offshore Mechanics and Arctic Engineering, 2014, paper no: OMAE2014-23325, 10 p.

[4] Inoue Y., Ali M.T. A study on numerical accuracy for the prediction of motion responses and drift forces of multiple floating bodies. Journal of the Society of Naval Architects of Japan, vol. 2002, issue 192, 2002, pp. 289-298.

[5] Ali M.T., Inoue Y. On hydrodynamic interaction between two rectangular barges floating side-by-side in regular waves. In Proc. of the International Conference on Mechanical Engineering, 2005.

[6] Li B. Multi-body hydrodynamic resonance and shielding effect of vessels parallel and nonparallel side-by-side. Ocean Engineering, vol. 218, 2020, article no. 108188, 23 p.

[7] Ohkusu M. On the heaving motion of two circular cylinders on the surface of a fluid. Reports of Research Institute for Applied Mechanics, vol. XVII, issue 58, 1969, pp. 167-185.

[8] Faltinsen O. and Michelsen F. Motions of large structures in waves at zero Froude number. In Proc. of the International Symposium on the Dynamics of Marine Vehicles and Structures in Waves, 1974, pp. 91-106.

[9] van Oortmerssen G. Hydrodynamic interaction between two structures, floating in waves. In Proc. of the 2nd International Conference on Behavior of Offshore Structures, 1979, pp. 339-356.

[10] Ali R., Tryaskin N.V. Study the influence of the relative position of two vessels moving in parallel on their hydrodynamic characteristics. Marine intellectual technologies, issue 50, part 4, 2020, pp. 59-65 (in Russian) / Али Р., Тряскин Н.В. Изучение влияния относительного взаимного расположения двух движущихся параллельно судов на их гидродинамические характеристики. Морские интеллектуальные технологии, вып. 50, часть 4, 2020 г., стр. 59-65.

[11] Ali R., Tryaskin N.V. Study the influence of ship length ratio on ship to ship interaction on moving in parallel on their hydrodynamic characteristics. Marine intellectual technologies, issue 4, part 2, 2021, pp. 63-72 (in Russian)/ Али Р., Тряскин Н.В. Изучение влияния соотношения длин двух параллельно движущихся судов на их гидродинамические характеристики. Морские интеллектуальные технологии, Морские интеллектуальные технологии, вып. 4, часть 2, 2021 г., стр. 63-72.

[12] Menter F.R., Kuntz M., Langtry R. Ten Years of Industrial Experience with the SST Turbulence Model. In Proc. of the 4th International Symposium on Turbulence, Heat and Mass Transfer, 2003, pp. 625-632.

[13] Ali R, Tryaskin N.V. Numerical study on effect of the turbulence initial conditions on transition flow over 2D airfoil. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 203-214. DOI: 10.15514/ISPRAS-2019-31(6)-1.

Information about the author / Информация об авторе

Rami ALI is a Senior Lecturer of the Department of Hydroaeromechanics and Marine Acoustics. Research interests: Marin hydrodynamics, computational fluid dynamics, CFD simulation, fluid structure interactions.

Рами АЛИ – старший преподаватель кафедры гидроаэромеханики и морской акустики. Научные интересы: морская гидродинамика, вычислительная гидродинамика, CFD-моделирование, взаимодействие жидкость-структура.

DOI: 10.15514/ISPRAS-2022-34(6)-16

Математическое моделирование процесса течения газа в проточной части турбомолекулярного вакуумного насоса с использованием модели взаимодействия газа с поверхностью Черчиньяни-Лампис

[1] *У.С. Гордеева, ORCID: 0000-0002-3499-001X <Uliana.Gordeev@yandex.ru>*
[2] *Ф.М. Шарипов, ORCID: 0000-0001-9372-2915 <sharipov@fisica.ufpr.br>*

[1] *МГТУ им. Н.Э. Баумана,*
105005, Россия, Москва, 2-я Бауманская ул., д. 5, стр. 1.
[2] *Федеральный университет Параны,*
Бразилия 80060-000, штат Парана, Куритиба, ул. 15-го ноября, 1299

Аннотация. В работе моделируется процесс течения газа в проточной части турбомолекулярного вакуумного насоса с использованием модели Черчиньяни-Лампис (ЧЛ), которая использовалась в качестве новых граничных условий при расчете вероятности перехода. При моделировании использовался метод пробной частицы (метод Монте-Карло). Произведен расчет вероятности перехода молекул через лопаточный канал в прямом и обратном направлениях, результирующей вероятности перехода, степени повышения давления.

Ключевые слова: математическое моделирование; турбомолекулярные вакуумные насосы; метод Монте-Карло; динамика разреженного газа; модели взаимодействия газа с поверхностью; модель Максвелла; модель Черчиньяни-Лампис

Для цитирования: Гордеева У.С., Шарипов Ф.М. Математическое моделирование процесса течения газа в проточной части турбомолекулярного вакуумного насоса с использованием модели взаимодействия газа с поверхностью Черчиньяни-Лампис. Труды ИСП РАН, том 34, вып. 6, 2022 г., стр. 197-206. DOI: 10.15514/ISPRAS-2022-34(6)-16

Mathematical modeling the process of gas flow in the turbomolecular pump using the Cercignani-Lampis gas-surface interaction model

[1] *U.S. Gordeeva, ORCID: 0000-0002-3499-001X <Uliana.Gordeev@yandex.ru>*
[2] *F.M. Sharipov, ORCID: 0000-0001-9372-2915 <sharipov@fisica.ufpr.br>*

[1] *Bauman Moscow State Technical University,*
2-ya Baumanskaya st. 5, str. Moscow, 105005, Russia
[2] *Universidade Federal do Paraná,*
Rua XV de Novembro 1299, Curitiba, State of Parana, 80060-000 Brazil

Abstract. In this paper the process of gas flow in the flow path of a turbomolecular vacuum pump using the Cercignani-Lampis (CL) model was simulated. CL model was used as new boundary conditions when calculating the transition probability. The test particle method (Monte Carlo method) was used in the

simulation. The calculation of the molecules transition probability through the blade channel in the forward and reverse directions, the transition resulting probability, the compression ratio was made.

Keywords: mathematical modelling; turbo molecular vacuum pumps; Monte-Carlo method; rarefied gas dynamics; gas-surface interaction models; Maxwell model; Cercignani-Lampis model.

For citation: Gordeeva U.S., Sharipov F.M. Mathematical modeling the process of gas flow in the turbomolecular pump using the Cercignani-Lampis gas-surface interaction model. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 6, 2022. pp. 197-206 (in Russian). DOI: 10.15514/ISPRAS-2022-34(6)-16

1. Введение

Турбомолекулярный вакуумный насос (ТМН), изобретенный Беккером в 1957 году, стал выпускаться в 1958 году. С тех пор он нашел широкое применение во всех областях техники, где необходимо обеспечение высокого и сверхвысокого вакуума благодаря чистому, постоянному и создаваемому вакууму, простоте эксплуатации, и повышенной степени эксплуатационной надежности. ТМН является единственным механическим вакуумным насосом, который вместе с форвакуумным насосом может обеспечивать предельное остаточное давление не менее 10^{-9} Па.

Первая конструкция ТМН была предложена в 1913 г., когда Геде представил свой молекулярный насос, в котором эффект откачки обусловлен передачей импульса от быстро движущейся поверхности ротора частицам газа. Геде был одним из изобретателей диффузионного насоса, в котором использовалась передача количества движения от быстро движущихся молекул нагретого потока пара рабочей жидкости частицам газа.

При молекулярном течении газа частицы сталкиваются преимущественно с внутренними поверхностями, что приводит к эффективному процессу откачки. Поэтому насосы, использующие этот принцип работы, называются молекулярными насосами. При переходном и вязкостном течении газа действие насоса ограничивается частыми взаимными столкновениями частиц. Поэтому молекулярные насосы используются в сочетании с подходящими форвакуумными насосами для достижения эффективной работы и стабильных показателей.

В 1922 году Хольвеком была предложена улучшенная версия насоса Геде. Зигбан в 1940 году сконструировал еще один молекулярный насос с дискообразным ротором. Эти ранние молекулярные насосы не достигли широкого применения из-за их низкой быстроты действия и низкой надежности. Для достижения низкого предельного остаточного давления расстояния между вращающимися – роторными и неподвижными – статорными частями достигали сотые доли миллиметра. Поэтому любое изменение температуры или попадание твердых частиц могло привести к отказу насоса из-за останова ротора.

Конструкция ТМН Беккера позволила избежать этих недостатков [1]. В основном конструкция состоит из высокоскоростного ротора и неподвижного статора объединенных в корпусе. Вращающийся и неподвижный диски расположены последовательно. Расстояния между вращающимися и неподвижными частями находятся в диапазоне от нескольких десятых долей до нескольких миллиметров. Все диски имеют наклонные лопатки с каналами между ними, а лопатки роторов и статоров наклонены в противоположную сторону. Каждый канал на диске действует как элементарный молекулярный насос, подобный насосу Геде. Все каналы на одном диске соединены параллельно и вместе дают высокую быстроту действия.

В дальнейшем компаниями Edwards Vacuum, Pfeiffer Vacuum была предложена конструкция с комбинированной проточной частью ТМН, которая объединила в одном корпусе лопаточные колеса и рабочие диски молекулярного насоса. Эта конструкция обладает преимуществом обеспечения высокой быстроты действия и низкого предельного остаточного давления [2, 3]. Имеются и некоторые недостатки в виде сложности промышленного исполнения, требований к оборудованию для изготовления высокоточной

проточной части, а также необходимость проведения сложных расчетов для определения оптимальных основных параметров ТМН. Кроме того, создание математической модели течения газа в проточной части ТМН позволит минимизировать количество экспериментальных образцов и повысить качество производства и эффективность работы ТМН.

В настоящее время ТМН используются в самых разных областях науки и техники: в масс-спектрометрах, нанесение покрытий, ускорителях элементарных частиц, течеискателях, при имитации условий космического пространства [4, 5]. Основные области применения ТМН приведены на рис. 1.

Рис. 1. Области применения ТМН
Fig. 1. Applications of turbomolecular vacuum pumps

Задача совершенствования характеристик ТМН является достаточно актуальной в условиях необходимости получения технологического суверенитета Российской Федерации. Для ускорения процессов проектирования, разработки и экспериментальной обработки ТМН критическим условием является наличие точных математических моделей, описывающих процесс течения газа в проточной части ТМН.

На качество моделирования течения газа в проточной части ТМН достаточно сильно влияют особенности моделирования взаимодействия молекул откачиваемого газа с поверхностями насоса. В настоящее время в большом количестве работ в качестве граничного условия используется граничное условие Максвелла (диффузное рассеяние), что недостаточно точно отражает особенности взаимодействия газа с твердой поверхностью.

Таким образом, целью работы является моделирование процесса течения газа через проточную часть с применением нового граничного условия, которое показало свою перспективность в части более точного моделирования особенностей взаимодействия газа с твердой поверхностью – модель, предложенная Черчиньяни-Лампис, ЧЛ) (Cercignani-Lampis Model, CL) [6,7].

Для достижения цели работы необходимо выполнить следующие задачи:

- описание расчетной схемы моделирования процесса течения газа в проточной части ТМН при помощи метода пробной частицы с применением граничного условия Максвелла и Черчиньяни-Лампис;
- проведение численных экспериментов по определению основных параметров ТМН с использованием указанных граничных условий при различных геометрических параметрах межлопаточного канала и скорости вращения ротора ТМН;
- анализ результатов численных экспериментов и разницы в результатах моделирования, полученных при помощи граничных условий Максвелла и ЧЛ;

2. Описание расчетной модели

В ходе исследования моделировалась прямая K_I и обратная K_{II} вероятности прохождения молекул через межлопаточный канал ТМН.

Начальные условия моделирования: проводится численное решение уравнения Максвелла – Больцмана.

$$f_{(V_x,V_y,V_z)}(v_x, v_y, v_z) = \frac{f_M(\boldsymbol{v})}{\mathrm{n}} = \frac{1}{(2\pi RT)^{3/2}} \exp[-\frac{(\boldsymbol{v})^2}{2RT}] = f_{V_x}(v_x)\, f_{V_y}(v_y)\, f_{V_z}(v_z),$$

$$f_{V_x}(v_x) = \frac{1}{(2\pi RT)^{1/2}} \exp[-\frac{{v_x}^2}{2RT}].$$

моделирование взаимодействия между стенками межлопаточного канала и молекулами моделируется двумя типами граничных условий.

- Модель Максвелла (диффузное рассеяние);

$$R_M(\nu' \to \nu; \epsilon) = (1-\epsilon)\delta(\nu - (\nu' - 2\,\nu'_n n)) + \epsilon \frac{m^2 v_n}{2\pi(kT_S)^2} \exp\left(\frac{-m\nu^2}{2kT_S}\right),$$

  где ν' и ν скорости падащюей на поверхность и отраженной поверхностью молекулы,
  v_n - нормальная составляющая скорости,
  m – масса молекулы,
  T_s - температура поверхности,
  n – нормаль к поверхности,
  k – постоянная Больцмана,
  $\delta(x)$ - дельта функция Дирака

- Модель ЧЛ;
  При использовании модели ЧЛ скорости молекул после столкновения со стенкой рассчитываются следующим образом:

$$c_{ct1} = \sqrt{\alpha_t \cdot (2-\alpha_t)} \cdot c_* \cdot cos\ cos\ \theta\ + (1-\alpha_t) \cdot c'_{t1}$$

$$c_{ct2} = \sqrt{\alpha_t \cdot (2-\alpha_t)} \cdot c_* \cdot sin\ sin\ \theta\ + (1-\alpha_t) \cdot c'_{t2}$$

$$c_n = \left[\alpha_n \cdot c_*^2 + (1-\alpha_n) \cdot {c'_n}^2 + 2 \cdot \sqrt{\alpha_n \cdot (1-\alpha_n)} \cdot c_* \cdot c'_n \cdot cos\ cos\ \theta\ \right]^{\frac{1}{2}},$$

  где c'_{t1}, c'_{t2}, c'_n – компоненты скорости молекул до взаимодействия со стенкой;
  c_{t1}, c_{t2}, c_n – компоненты скорости молекул после взаимодействия со стенкой;
  α_t, α_n – аккомодации тангенциального импульса и нормальной энергии, соответственно.

При моделировании использовались следующие допущения:

- пространственное распределение молекул на стороне высокого вакуума – равномерное;
- высота межлопаточного канала является бесконечной;
- режим течения газа - молекулярный (отсутствуют столкновения между молекулами газа);
- скорости молекул подчиняются распределению Максвелла;
- взаимодействие молекул газа со стенками ТМН описывается моделью Черчиньяни–Лампис и моделью Максвелла.

Расчетная схема моделирования приведена на рис. 2.

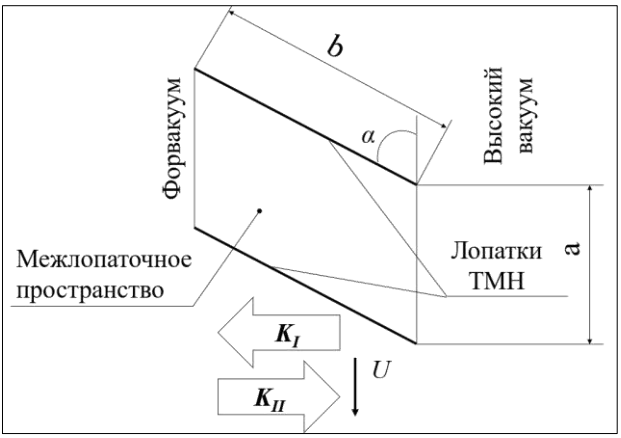

Рис. 2. Расчетная схема
Fig. 2. Calculation scheme

В данной работе рассмотрены следующие основные характеристики ТМН:

- результирующая вероятность перехода молекул газа через межлопаточный канал (пропорциональна быстроте действия насоса) K_{max};
- степень повышения давления τ_{max}.

Результирующая вероятность перехода K_{max} рассчитывается по следующей формуле:

$$K_{max} = K_I - K_{II},$$

где K_I – вероятность перехода молекул через межлопаточный канал в прямом направлении; K_{II} – вероятность перехода молекул через межлопаточный канал в обратном направлении;

Степень повышения давления τ_{max} рассчитывается по следующей формуле:

$$\tau_{max} = \frac{K_I}{K_{II}}.$$

Вероятности прохождения молекул в прямом и обратном направлении определяются методом пробной частицы (метод Монте-Карло), который показал свою эффективность при расчете характеристик ТМН.

Вероятность перехода через межлопаточный канал в прямом направлении рассчитывается по следующей формуле:

$$K_I = \frac{N_I}{N},$$

где N_I – количество частиц, которые перешли со стороны высокого вакуума на сторону форвакума, N – общее количество частиц при моделировании.

Вероятность перехода через межлопаточный канал в обратном направлении рассчитывается по следующей формуле:

$$K_{II} = \frac{N_{II}}{N},$$

где N_{II} – количество частиц, которые перешли со стороны форвакуума вакуума на сторону высокого.

Укрупненный алгоритм моделирования методом пробной частицы показан на рис. 3:

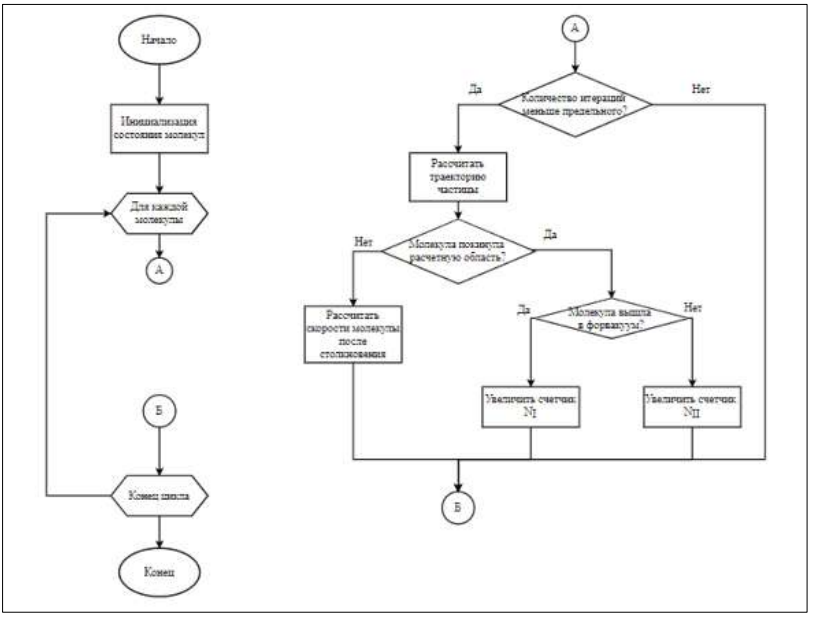

Рис. 3. Укрупненный алгоритм моделирования методом пробной частицы
Fig. 3. Enlarged modelling algorithm by the test particle method

Моделирование проводится в два этапа: моделирование прямого и обратного переходов молекул через межлопаточный канал. При прямом переходе частицы инициализируются на стороне высокого вакуума. При обратном переходе частицы инициализируются на стороне форвакуума. На каждом этапе моделируется движение каждой частицы в межлопаточном канале, подчитывается количество частиц, достигших форвакуума и высокого вакуума, соответственно.

Алгоритм реализован на языке GNU Octave (версия 5.1.0). При моделировании прямого и обратного перехода рассматривалось 10^6частиц.

3. Описание численного эксперимента

В ходе численного эксперимента рассматривались следующие комбинации геометрических параметров межлопаточного канала (угол наклона лопаток α, соотношения сторон межлопаточного канала $\frac{a}{b}$) и соотношения окружной скорости лопаток ТМН к наиболее вероятной тепловой скорости молекул C

- значения C = [0.5, 1.0, 1.5, 2.0];
- значения $\frac{a}{b}$ = [0.6 0.8 1. 1.2 1.4 1.6];
- значения α = [15, 30, 45].

Для каждой из моделей взаимодействия молекул и стенок межлопаточного канала (модель Максвелла и модель ЧЛ) моделируется процесс течения газа в молекулярном режиме при всех возможных вышеуказанных комбинациях геометрических параметров межлопаточного канала и соотношения скоростей.

Моделирование проводится отдельно с применением модели Максвелла и отдельно с применением модели ЧЛ со следующими значениями коэффициентов аккомодации тангенциального импульса $\alpha_t = 0.9$ и нормальной энергии $\alpha_n = 1.0$.

4. Результаты и обсуждения численного эксперимента

Результаты моделирования для случая угла наклона лопаток межлопаточного канала α=15° показаны на рис. 4.

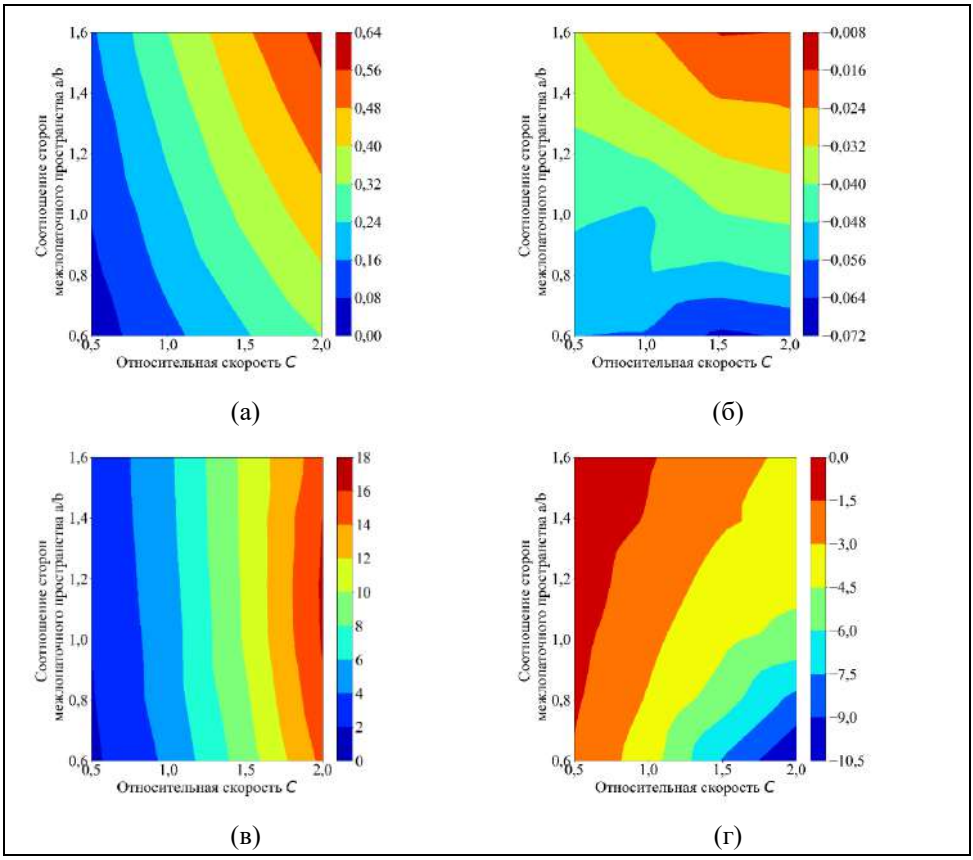

Рис. 4. Результаты моделирования для случая угла наклона лопаток межлопаточного канала α=15°: (а) Значения K_{max}, полученные при помощи модели ЧЛ, (б) Разница между значениями K_{max}, полученными при помощи модели ЧЛ и значениями, полученными при помощи модели Максвелла; (в) Значения T_{max}, полученные при помощи модели ЧЛ, (г) Разница между значениями T_{max}, полученными при помощи модели ЧЛ и значениями, полученными при помощи модели Максвелла

Fig. 4. Results of modelling for the case of the angle of inclination of the blades of the interscapular channel α=15°: (a) K_max values obtained with the CL model, (b) Difference between the K_max values obtained with the CL model and those obtained with the Maxwell model; (c) T_max values obtained using the CL model, (d) Difference between the T_max values obtained using the CL model and those obtained using the Maxwell model

Результаты моделирования для случая угла наклона лопаток межлопаточного канала α=30° показаны на рис. 5.

Результаты моделирования для случая угла наклона лопаток межлопаточного канала α=45° показаны на рис. 6.

Из анализа полученных графиков можно сделать вывод, что особенности моделирования процесса течения газа с применением граничных условий, описанных моделью ЧЛ имеют большее воздействие применительно к параметру τ_{max} при меньших углах наклона межлопаточного пространства и больших значениях соотношения скоростей U.

При малом соотношении сторон межлопаточного канала a/b значительно возрастает количество частиц, ударяющихся о стенки межлопаточного канала, скорости которых после столкновения рассчитываются при помощи модели ЧЛ, которая значительно отличается от модели Максвелла (диффузное рассеяние). Следовательно, чем больше относительная длина межлопаточного канала, тем больше будет разница между результатами моделирования с граничным условием ЧЛ и граничным условием Максвелла.

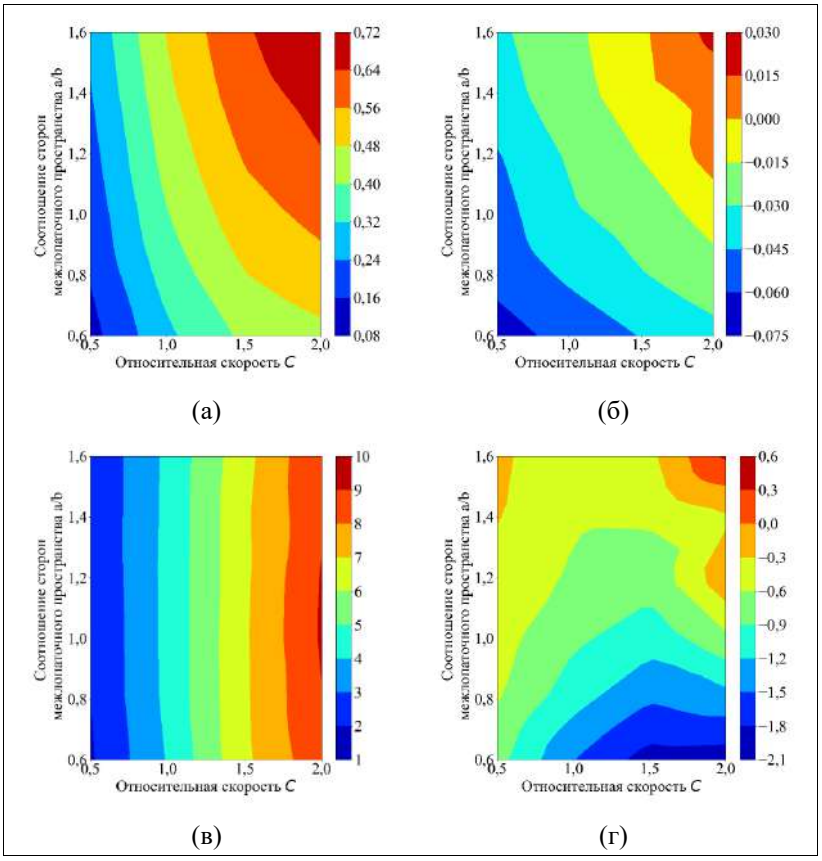

Рис. 5. Результаты моделирования для случая угла наклона лопаток межлопаточного канала α=30º: (а) Значения K_{max}, полученные при помощи модели ЧЛ, (б) Разница между значениями K_{max}, полученными при помощи модели ЧЛ и значениями, полученными при помощи модели Максвелла; (в) Значения T_{max}, полученные при помощи модели ЧЛ, (г) Разница между значениями T_{max}, полученными при помощи модели ЧЛ и значениями, полученными при помощи модели Максвелла

Fig. 5. Results of modelling for the case of the angle of inclination of the blades of the interscapular channel α=30º: (a) K_max values obtained with the CL model, (b) Difference between the K_max values obtained with the CL model and those obtained with the Maxwell model; (c) T_max values obtained using the CL model, (d) Difference between the T_max values obtained using the CL model and those obtained using the Maxwell model

Результаты проведенных численных экспериментов показывают существенное расхождение между результатами моделирования, полученными при помощи моделей Максвелла и ЧЛ. Данное обстоятельство необходимо учитывать при проектировании новых и совершенствовании существующих образцов ТМН.

При этом необходимо проведение дополнительных экспериментальных исследований для определения точных значений коэффициентов аккомодации в рамках модели ЧЛ применительно к требуемым типам откачиваемых газов (аргон, азот, гелий).

Кроме того, в рамках данной работы были проведены расчеты исключительно в молекулярном режиме. В дальнейших работах необходимо произвести исследования влияния новых граничных условий не только в молекулярном режиме, но и в переходном и вязкостном режимах.

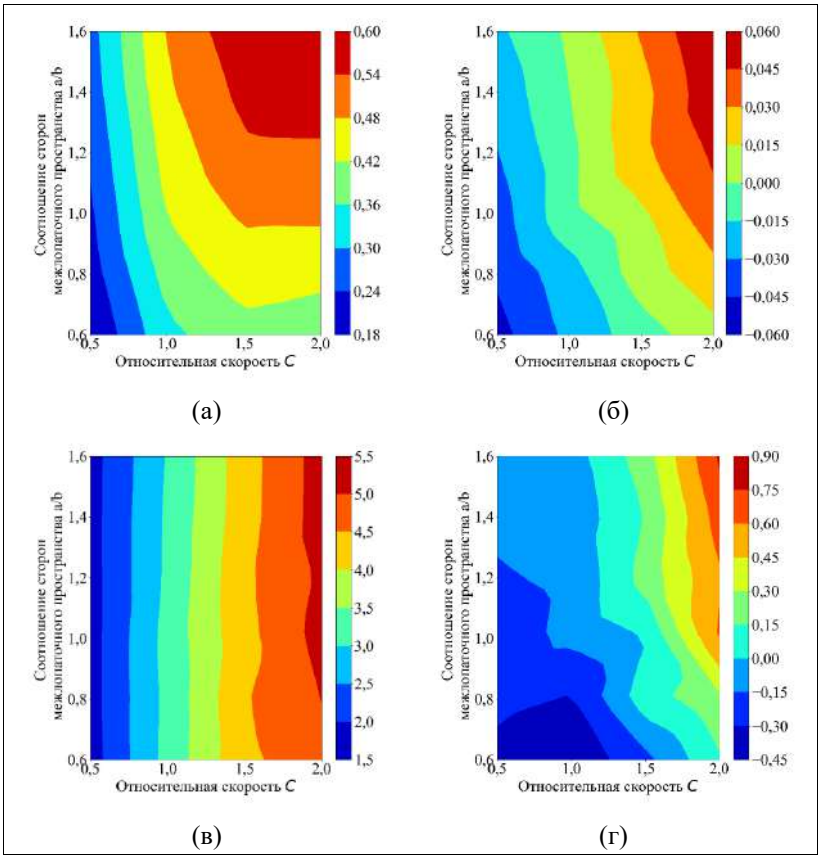

Рис. 6. Результаты моделирования для случая угла наклона лопаток межлопаточного канала α=45º: (а) Значения K_{max}, полученные при помощи модели ЧЛ, (б) Разница между значениями K_{max}, полученными при помощи модели ЧЛ и значениями, полученными при помощи модели Максвелла; (в) Значения T_{max}, полученные при помощи модели ЧЛ, (г) Разница между значениями T_{max}, полученными при помощи модели ЧЛ и значениями, полученными при помощи модели Максвелла

Fig. 6. Results of modelling for the case of the angle of inclination of the blades of the interscapular channel α=45º: (a) K_max values obtained with the CL model, (b) Difference between the K_max values obtained with the CL model and those obtained with the Maxwell model; (c) T_max values obtained using the CL model, (d) Difference between the T_max values obtained using the CL model and those obtained using the Maxwell model

5. Выводы

В работе было проведено численное моделирование процесса течения газа в проточной части ТМН при помощи метода пробной частицы при помощи двух подходов к моделированию взаимодействия молекул газа и с поверхностью ТМН: модели Максвелла (диффузное рассеяние) и модели Черчиньяни-Лампис. При помощи вычислительного эксперимента было оценено влияние вышеуказанных моделей взаимодействия газа с твердой поверхностью в широком диапазоне геометрических (угол наклона межлопаточного канала и соотношение сторон межлопаточного канала) и кинематических (соотношение скорости лопаток и

наиболее вероятной тепловой скорости движения молекул) параметров на основные характеристики ТМН (степень повышения давления и максимальная быстрота действия).

Результаты вычислительного эксперимента показывают существенную разницу между результатами, полученными при помощи модели Максвелла и модели Черчиньяни-Лампис, для определенных комбинаций параметров насоса, что необходимо учитывать при разработке и проектировании образцов ТМН.

Список литературы / References

[1]. Becker W. The turbomolecular pump, its design, operation and theory; calculation of the pumping speed for various gases and their dependence on the forepump. Vacuum, vol. 16, issue 11, 1966, pp, 625-632.

[2]. Sawada T. Performance of a Rotor with a Single Blade Row in the Transition Flow Regime. Bulletin of JSME, vol. 16, issue 96, 1973, pp. 993-1001.

[3]. Heo J.-S., Hwang Y.-K. DSMC calculations of blade rows of a turbomolecular pump in the molecular and transition flow regions, Vacuum, vol. 56, issue 2, 2000, pp. 133-142.

[4]. Bird G.A. Molecular gas dynamics and the direct simulation of gas flows, second edition. Clarendon Press, 1994, 484 p.

[5]. Demikhov K.E., Pyzhov I.I. A model for the gas-molecule transfer process by the intervane channels of the rotor of a turbomolecular vacuum pump. Chemical and Petroleum Engineering, vol. 13, issue 1, 1977, pp. 49-52.

[6]. Sharipov F. Application of the Cercignani-Lampis scattering kernel to calculations of rarefied gas flows. I. Plane flow between two parallel plates. European Journal of Mechanics - B/Fluids, vol. 21, issue 1, 2002, pp.113-123:

[7]. Фролова А.А. Численное сравнение обобщенной модели Максвелла и модели Черчиньяни–Лэмпис. Журнал вычислительной математики и математической физики, том 60, вып. 12, 2020 г., стрю 2162–2176 / Frolova A.A. Numerical comparison of the generalized Maxwell and Cercignani–Lampis models. Computational Mathematics and Mathematical Physics, vol. 60, issue 12, 2020, pp. 2094-2107.

Информация об авторах / Information about authors

Ульяна Саидовна ГОРДЕЕВА – соискатель, выпускница кафедры «Вакуумная и компрессорная техника» МГТУ им. Н.Э. Баумана. Сфера научных интересов: разработка вакуумных систем, вычислительная газодинамика, математическое моделирование.

Uliana Saidovna GORDEEVA – PhD research, graduate of the department " Vacuum and Compressor Equipment" Bauman Moscow State Technical University. Research interests: vacuum systems' development, computational gas dynamics, mathematical modelling.

Феликс Маратович ШАРИПОВ – профессор кафедры физики Федерального университета штата Парана, Бразилия. Сфера научных интересов: численные методы динамики разреженного газа в микргидродинамике, вакуумной технике и аэротермодинамике.

Felix Maratovich SHARIPOV – professor at Department of Physics Federal University of Parana, Brazil. Research interests: numerical methods of rarefied gas dynamics applied to microfluidics, vacuum technology and aerothermodynamics.