Научная статья
УДК 004.4
https://doi.org/10.31854/1813-324X-2025-11-6-88-100
EDN:TQFFYA

Реверс-инжиниринг программного обеспечения методом смарт-перебора: прототип и эксперимент

Константин Евгеньевич Израилов[1] ✉, konstantin.izrailov@mail.ru
Михаил Викторевич Буйневич[2], bmv1958@yandex.ru

[1]Санкт-Петербургский университет государственной противопожарной службы МЧС России,
Санкт-Петербург, 196105, Российская Федерация
[2]МИРЭА – Российский технологический университет,
Москва, 119454, Российская Федерация

Аннотация

Актуальность. *Одним из подходов к поиску уязвимостей в программах является их преобразования из выполняемого машинного кода в человеко-ориентированный исходный, более пригодный для работы эксперта по информационной безопасности. Ранее авторами был получен соответствующий метод «умного» перебора вариантов исходного кода на предмет установления экземпляра, компилируемого в заданный машинный. Логичным продолжением исследования должна стать реализация программного прототипа для проверки работоспособности метода и экспериментального получения ряда характеристик.*

Цель исследования: *реализовать программный прототип смарт-перебора вариантов исходного кода (согласно разработанному методу), экспериментально оценить его работоспособность и границы применимости.*

Методы: *программная инженерия, эксперимент, аппроксимация значений.*

Результаты: *создан программный прототип подбора экземпляра исходного кода по заданному машинному, реализующий соответствующий метод, полученный в предыдущих авторских исследованиях. Осуществлено применение прототипа для получения исходного кода математического выражения по его машинному коду с использованием части формального синтаксиса языка программирования (заданного в форме графа синтаксических правил). Проведена серия экспериментов для оценки характеристик прототипа путем определения следующих зависимостей: количество всех вариантов исходного кода от синтаксической разнородности синтаксиса и максимальной глубины обхода его графового представления, время поиска определенного исходного кода от данной глубины обхода. Тестирование прототипа показало его базовую работоспособность и гипотетический потенциал, что обосновывает и саму возможность осуществления реверс-инжиниринга противоположным к классическому способом – от исходного кода, а не машинного.*

Практическая значимость: *текущая версия прототипа может непосредственно применяться для осуществления декомпиляции небольших частей машинного кода, при этом, без «привязки» к конкретному языку программирования и процессорной архитектуре (поскольку требуется лишь средство компиляции).*

Обсуждение: *существенным усовершенствованием «умного» перебора может стать его качественная оптимизация путем применения искусственного интеллекта в части генетических алгоритмов.*

Ключевые слова: *информационная безопасность, реверс-инжиниринг, декомпиляция, метод, смарт-перебор, исходный код, машинный код, прототип, эксперимент*

Ссылка для цитирования: Израилов К.Е., Буйневич М.В. Реверс-инжиниринг программного обеспечения методом смарт-перебора: прототип и эксперимент // Труды учебных заведений связи. 2025. Т. 11. № 6. С. 88–100. DOI:10.31854/1813-324X-2025-11-6-88-100. EDN:TQFFYA

Original research
https://doi.org/10.31854/1813-324X-2025-11-6-88-100
EDN:TQFFYA

Reverse Engineering of Software Using the Smart Brute Force Method: Prototype and Experiment

Konstantin E. Izrailov[1]✉, konstantin.izrailov@mail.ru
Mikhail V. Buinevich[2], bmv1958@yandex.ru

[1]Saint-Petersburg University of State Fire Service of EMERCOM of Russia,
St. Petersburg, 196105, Russian Federation
[2]MIREA – Russian Technological University,
Moscow, 119454, Russian Federation

Annotation

Introduction. *one approach to finding vulnerabilities in programs is converting the executable machine code into human-oriented source code, which would be more suitable for an information security expert. The authors previously developed a corresponding method for «smart» enumeration of source code variants to identify a copy that compiles to a given machine code. A logical continuation of this research would be the implementation of a software prototype to test the method's performance and experimentally determine some characteristics.*
Purpose: *implementing the software prototype of smart exhaustive search of source code variants (according to the described method), as well as the experimental evaluation of its operability and the limits of its applicability.*
Methods: *software engineering, experimentation, approximation of values.*
Results. *the creation of a software prototype for selecting an instance of the source code according to the set machine code, obtained in previous author's studies. The prototype was used to obtain the source code of a mathematical expression from its machine code using part of the formal syntax of a programming language (defined in the form of a graph of syntactic rules). A series of experiments was conducted to evaluate the characteristics of the prototype by determining the following dependencies: the number of all source code variants on the syntactic heterogeneity of the syntax and the maximum depth of traversal of its graphical representation, as well as the search time for a specific source code from a set depth of traversal. These tests proved the basic functionality of the prototype and its hypothetical potential, which also justifies the possibility of opposite reverse engineering as compared to the traditional method - from source code, rather than machine code.*
Practical significance: *the current version of the prototype can be used practically to decompile small parts of machine code, without being dependent on a specific programming language and processor architecture (since only a compilation tool is required).*
Discussion: *the qualitive optimization of the "smart" exhaustive search through the use of artificial intelligence in terms of genetic algorithms can significantly improve the search.*

Keywords: *information security, reverse engineering, decompilation, method, smart brute-force, source code, machine code. prototype, experiment*

For citation: Izrailov K.E., Buinevich M.V. Reverse Engineering of Software Using the Smart Brute Force Method: Prototype and Experiment. *Proceedings of Telecommunication Universities.* 2025;11(6):88–100. (in Russ.) DOI:10.31854/1813-324X-2025-11-6-88-100. EDN:TQFFYA

Введение

Существенное влияние на безопасность программного обеспечения (далее – ПО) оказывают содержащиеся в нем уязвимости, приводящие к соответствующим информационным угрозам [1]. И если базовые уязвимости могут быть найдены статическими анализаторами кода (например, по PE-заголовкам программ [2]), то пока для их обнаружения в более высокоуровневых представлени-

ях (алгоритмах, архитектуре и т. п.) требуется участие эксперта. Для этого необходимо произвести анализ исходного кода (далее – ИК), выявить в нем «слабые» места и затем устранить их. Ситуация существенно усложняется тем, что, как правило, программа имеет вид машинного кода (далее – МК), который содержит инструкции центрального процессора управления (далее – ЦПУ), сложно воспринимаемые человеком; при этом восстановление логики работы такого кода становится проблемный вопросом для специалиста любого уровня. Следовательно, преобразование МК в его представление в виде ИК, называемое реверс-инжинирингом (далее – РИ) или декомпиляцией, является важнейшей задачей данной предметной области.

Ранее авторами в [3] было произведено качественное сравнение различных подходов к РИ, среди которых был выделен полный перебор, научно-необоснованно отвергнутый предыдущими исследователями; в результате была поставлена научная задача исследования возможности конструирования ИК при параметрическом задании синтаксиса языка программирования (далее – ЯП) для получения варианта его экземпляра, компилируемого в заданный МК. Обзор работ (см. соответствующий раздел в [3]) показал практически полное отсутствие подходящих решений этого.

В интересах решения поставленной задачи был предложен метод «умного» (смарт-метод) перебора ИК на основе формального синтаксиса ЯП, преобразуемого в специальный граф синтаксических правил (далее – ГСП); подобные синтаксисы часто применяются в генераторах анализаторов текста (например, в ANTLR [4]). Идея метода заключается в конструировании различных вариантов ИК, пока один из таких экземпляров не будет компилироваться в МК, тождественный исследуемому. В случае нахождения такого ИК его можно будет считать результатом решения задачи декомпиляции или РИ [5, 6] соответствующего МК.

Продолжая начатое исследование возможности применения полного перебора ИК в интересах РИ, далее будет предложена конкретная реализация Метода и произведен ряд экспериментов с получением качественно-количественных оценок.

Прототип «умного» перебора

В рамках настоящего исследования Метод был реализован в виде соответствующего программного прототипа (далее – Прототип) на языке Python 3.11. Алгоритм Прототипа хотя и соответствует схеме Метода (см. рисунок 3 в [3]), однако является более сложным из-за нетривиальности логики Шагов 2, 3 и 6 в Методе, предложенном в [3], и будет изложен в дальнейших публикациях. Поэтому представим лишь общий алгоритм Прототипа в виде интуитивно понятного псевдокода без детализации (листинг 1).

Листинг 1. Алгоритм Прототипа (в виде псевдокода)

Listing 1. Prototype's Algorithm (in pseudocode)

```
Input:
  LanguageSyntax – формальный синтаксис ЯП
  LanguageLexica – формальная лексика ЯП
  MaxDeep – максимальная длина пути по ГСП
  MachineCodeTarget – МК, для которого ищется ИК
                      (т. е. производится РИ)

Output:
  SourceCodeTarget – искомый ИК или «None», если РИ не
удалось произвести

Begin
    // Шаг 1
1:  SRG = BuildSRG(LanguageSyntax, LanguageLexica);

    // Шаг 2
2:  Path = InitPath(SRG, MaxDeep);

3:  While (True):
      // Шаг 3
4:    SourceCode = GenerateSourceCode(Path, SGR);

      // Шаг 4
5:    MachineCode = Compile(SourceCode);

      // Шаг 5
6:    If (MachineCode == MachineCodeTarget) Then
        // Successfull result
7:      SourceCodeTarget = SourceCode;
8:      Break
9:    End If

      // Шаг 6
10:   Path = GetNextPath(Path, SGR, MaxDeep)

      // Шаг 7
11:   If (Path == []) Then
        // Unsuccessfull result
12:     SourceCodeTarget = None;
13:     Break
14:   End If

15:  End While

16:  Return SourceCodeTarget;
End
```

Опишем более детально каждую строку псевдокода, номер которых указан с постфиксом «:».

В строке 1 строится ГСП.

В строке 2 создается первый путь по ГСП.

В строке 3 начинается бесконечный цикл перебора путей для обхода ГСП (прерывание цикла происходит внутри него).

В строке 4 генерируется ИК в соответствии с текущем путем по ГСП.

В строке 5 получается МК из ИК путем компиляции.

В строке 6 сравнивается полученный и заданный МК (побайтно, сопоставлением ассемблером или с помощью более сложных техник [7]).

В строке 7, в случае выполнения условия в строке 6, сохраняется найденный ИК.

В строке 8 происходит выход из цикла (с успешным статусом).

В строке 9 оканчивается блок условия, начатый в строке 6.

В строке 10 получается новый путь по ИК.

В строке 11 осуществляется проверка того, не является ли путь последним (т. е. содержащим пустую последовательность).

В строке 12, в случае выполнения условия в строке 10, указывается, что ИК не был найден.

В строке 13 происходит выход из цикла (с неуспешным статусом).

В строке 14 оканчивается блок условия, начатый в строке 11.

В строке 15 оканчивается блок цикла, начатый в строке 3.

В строке 16 из алгоритма возвращается найденный ИК или значение None, если код не был найден.

Продемонстрируем работу Прототипа для простого примера, что позволит оценить базовую работоспособность Метода. В качестве синтаксиса (включающего и лексику) ЯП, воспользуемся приведенным на Листинге в [3] (см. стр. 135), но ограничив идентификаторы только тремя следующими: «*x*», «*y*» и «*z*»; формальная запись такого модифицированного синтаксиса в форме Бэкуса – Наура (далее – БНФ) [8] представлена ниже (Листинге 2).

Листинг 2. Синтаксиса в форме Бэкуса – Наура (пример 1, модифицированный)

Listing 2. Syntax in Backus – Naur form (Example 1, Modified)

```
1: expr_asgn ::= ident, '=', expr_oper ;
2: expr_oper ::= ident, oper, ident ;
3: oper ::= '+' | '-' | '*' | '/' ;
4: ident ::= 'x' | 'y' | 'z' ;
```

Отладочный вывод ГСП при работе Прототипа приведен на Листинге 3 («*» означает ссылку на уже заданный нетерминал графа, а число после «ID:» – внутренний идентификатор вершины ГСП). Анализ данного листинга позволяет сделать вывод, что он определяет подмножество представления ГСП на рисунке 4 из [3].

Отладочный вывод Прототипа, содержащий все возможные пути по ГСП и соответствующие им варианты ИК, представлен на Листинге 4 («…» используется для пропуска промежуточных строк). В каждой строчке выводится путь (во внутренней нотации), а после «->» соответствующий ему ИК. Формат пути имеет JSON-записи и содержит последовательность выбора альтернатив (т. е. веток ГСП, по которым может пойти синтаксический разбор или генерация), каждая из которых записана, как словарь со полями «c_idx», «c_max», «n_id».

Листинг 3. Отладочный вывод графа синтаксических правил

Listing 3. Debug Output of the Syntax Rule Graph

```
AsNode(ID:6)
  AsNode(ID:2)
    AsAlts(ID:1)
      Alt_1()
        'x'
      Alt_2()
        'y'
      Alt_3()
        'z'
  '='
  AsNode(ID:5)
    AsNode(ID:2)*
    AsNode(ID:4)
      AsAlts(ID:3)
        Alt_1()
          '+'
        Alt_2()
          '-'
        Alt_3()
          '*'
        Alt_4()
          '/'
    AsNode(ID:2)*
```

Листинг 4. Отладочный вывод работы Прототипа (для режима полного перебора)

Listing 4. Debug Output of Prototype (for Full Enumeration Mode)

```
[{"c_idx": 0, "c_max": 3, "n_id": 1}, {"c_idx": 0,
"c_max": 3, "n_id": 1}, {"c_idx": 0, "c_max": 4, "n_id":
3}, {"c_idx": 0, "c_max": 3, "n_id": 1}] -> x=x+x
[{"c_idx": 0, "c_max": 3, "n_id": 1}, {"c_idx": 0,
"c_max": 3, "n_id": 1}, {"c_idx": 0, "c_max": 4, "n_id":
3}, {"c_idx": 1, "c_max": 3, "n_id": 1}] -> x=x+y
...
[{"c_idx": 2, "c_max": 3, "n_id": 1}, {"c_idx": 2,
"c_max": 3, "n_id": 1}, {"c_idx": 3, "c_max": 4, "n_id":
3}, {"c_idx": 1, "c_max": 3, "n_id": 1}] -> z=z/y
[{"c_idx": 2, "c_max": 3, "n_id": 1}, {"c_idx": 2,
"c_max": 3, "n_id": 1}, {"c_idx": 3, "c_max": 4, "n_id":
3}, {"c_idx": 2, "c_max": 3, "n_id": 1}] -> z=z/z
```

Поле «c_idx» – индекс выбранной ребра-альтернативы ГСП, начиная с 0-го, по которому «идет» алгоритм Прототипа (используется для генерации ИК и получения следующего пути); поле «c_max» – максимальное число альтернатив для текущей вершины ГСП (используется для получения следующего пути); поле «n_id» – идентификатор вершины ГСП с альтернативами (используется для отладочных целей, а также других алгоритмов, таких как авторский генетический реверс-инжиниринг).

Запуск Прототипа в режиме перебора всех возможных ИК (т. е. без компиляции и сравнения с заданным МК) позволил установить общее число всех их комбинаций, как 108. Данное значение соответствует теоретическим расчетам, поскольку ИК в форме «Идентификатор = Идентификатор Оператор Идентификатор» при заданных синтаксисом условиях имеет также 3 × 1 × 3 × 4 ×3 = 108 вариаций.

Эксперимент

Проведем расширенные эксперименты с Прототипом, используя более сложный синтаксис ЯП, ГСП которого имеет циклы. Данный синтаксис

описывает приравнивание к переменной множества математических выражений, которые состоят из бинарной математической операции над двумя операндами, второй из которых может быть как другой переменной, так и вложенным математическим выражением (взятым в скобки); например, данному синтаксису соответствуют следующие варианты ИК: «$x = y$», «$z = x + y$», «$y = x – (y + (z * (x / y)))$» и т. п. При этом, количество таких ИК может быть сколь угодно большим, что существенно усложняет задачу по сравнению с предыдущей, заданной через ГСП на Листинге 2. Соответственно, синтаксис в БНФ, используемый далее в эксперименте, приведен на Листинге 5.

Листинг 5. Синтаксиса в форме Бэкуса – Наура (пример 2)

Listing 5. Syntax in Backus – Naur Form (Example 2)

```
1: expr_assign ::= ident, '=', expr ;
2: expr ::= ident | expr_oper ;
3: expr_oper = ident, oper, ident | ident, oper, '(',
expr_oper, ')' ;
4: oper = '+' | '-' | '*' | '/' ;
5: ident = 'x' | 'y' | 'z' ;
```

ГСП для синтаксиса на Листинге 5 представлен на рисунке 1 (для упрощения восприятия пунктирные линии связей между идентичными нетерминалами опущены кроме одной, отражающей цикличность графа).

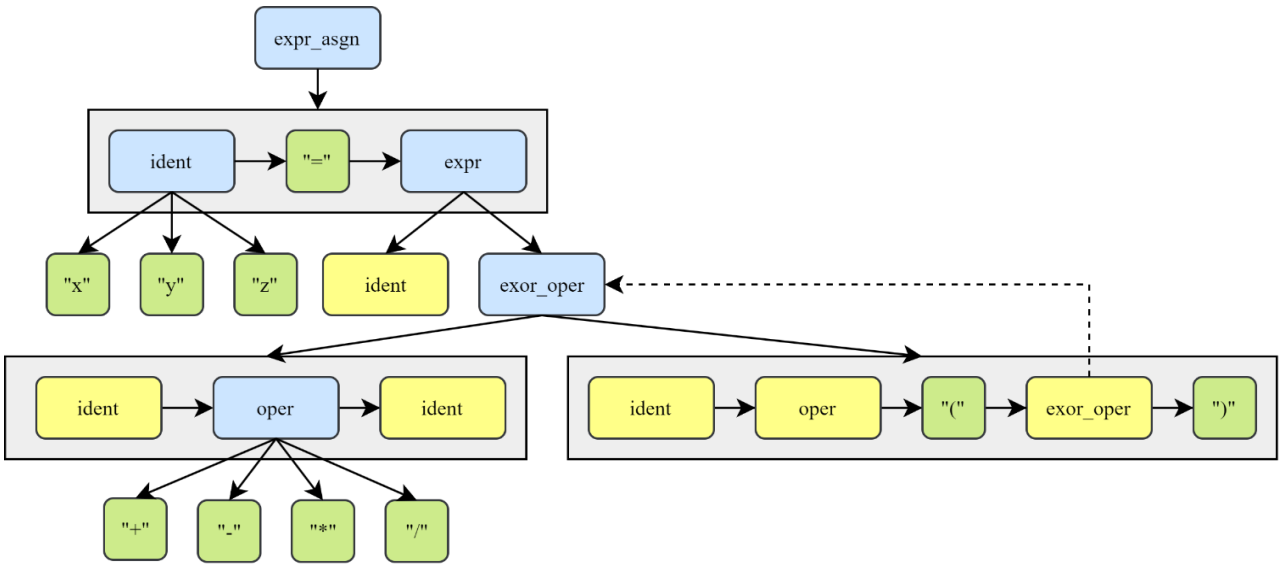


Рис. 1. Представление графа синтаксических правил для Листинга 5

Fig. 1. Representation of the Syntax Rules Graph for Listing 5

Рекурсивность синтаксиса отражается на рисунке 1 тем, что элемент «expr_oper» в нижней правой части графа является дочерним для такого же элемента в центральной части графа; очевидно приводя к бесконечному циклическому обходу ГСП, что разрешается указанием алгоритму максимальной длины пути (далее – МДП), по которому данный обход производится.

Текущая реализация Прототипа может непосредственно применяться для РИ блоков МК с небольшими математическими уравнениями в соответствующий ему блок ИК. Тогда предположим, что имеется исследуемый МК со следующим ассемблерным кодом тела функции – т. е. текстовым представлением инструкции ЦПУ для программы [9] (для упрощения приведена только основная часть функции, а также вручную добавленные комментарии с назначением инструкций ЦПУ в каждой точке после «;»):

```
mov eax, DWORD PTR _x$[ebp]  ; перемещение значения пе-
ременной «x» в регистр EAX
cdq                          ; преобразование знакового
двойного слова из EAX
                             ; в знаковое четверное сло-
во в EDX:EAX
idiv DWORD PTR _y$[ebp]      ; знаковое деление регистра
EAX на переменную «y»
mov ecx, DWORD PTR _y$[ebp]  ; перемещение значения пе-
ременной «y» в регистр ECX
sub ecx, eax                 ; вычитание из регистра ECX
регистра EAX
imul ecx, DWORD PTR _x$[ebp] ; знаковое умножение реги-
стра ECX на переменную «x»
mov DWORD PTR _z$[ebp], ecx  ; перемещение значения ре-
гистра ECX в переменную «z»
```

Высококвалифицированный эксперт по безопасности ПО (далее – Эксперт) после ручного анализа такого ассемблерного кода сможет восстановить его ИК, а именно следующее математическое выражение (из 13 символов текста, без пробелов): «$z = x * (y – (x / y))$».

И несмотря на то, что данная задача считается решаемой Экспертом среднего уровня, для ее выполнения потребуется знание архитектуры ЦПУ,

детальное понимание его инструкций и определенное время (исходя из практического опыта авторов и консультаций с сотрудниками данной инженерной области – около 3–5 минут для опытного и 15 минут для начинающего специалиста).

Для решение такой задачи РИ с помощью Прототипа необходимо автоматически перебрать все возможные комбинации ИК согласно синтаксису на Листинге 5, скомпилировать их в МК и сравнить с исследуемым. Для корректной компиляции будем «размещать» генерируемый ИК (отмечено далее красным курсивом) в теле функции следующего шаблона:

```
int f(int x, int y) {
  int z;
  Генерируемый ИК;
  return z;
}
```

Непосредственное применение Прототипа (с указанием МДП, равной 12 и выбранной эмпирически) получит следующий отладочный вывод, представленный на Листинге 6.

Листинг 6. Отладочный вывод работы Прототипа (для режима реверс-инжиниринга машинного кода)

Listing 6. Debug Output of Prototype (for Machine Code Reverse Engineering Mode)

```
1) {'c_idx': [0, 0, 0], 'c_max': [3, 2, 3], 'n_id': [1,
7, 1]}: x=x
2) {'c_idx': [0, 0, 1], 'c_max': [3, 2, 3], 'n_id': [1,
7, 1]}: x=y
3) {'c_idx': [0, 0, 2], 'c_max': [3, 2, 3], 'n_id': [1,
7, 1]}: x=z
4) {'c_idx': [0, 1, 0, 0, 0, 0], 'c_max': [3, 2, 2, 3,
4, 3], 'n_id': [1, 7, 5, 1, 3, 1]}: x=x+x
5) {'c_idx': [0, 1, 0, 0, 0, 1], 'c_max': [3, 2, 2, 3,
4, 3], 'n_id': [1, 7, 5, 1, 3, 1]}: x=x+y
6) {'c_idx': [0, 1, 0, 0, 0, 2], 'c_max': [3, 2, 2, 3,
4, 3], 'n_id': [1, 7, 5, 1, 3, 1]}: x=x+z
...
20118) {'c_idx': [2, 1, 1, 0, 2, 1, 1, 1, 0, 0, 2, 2],
'c_max': [3, 2, 2, 3, 4, 2, 3, 4, 2, 3, 4, 3], 'n_id':
[1, 7, 5, 1, 3, 5, 1, 3, 5, 1, 3, 1]}: z=x*(y-(x*z))
20119) {'c_idx': [2, 1, 1, 0, 2, 1, 1, 1, 0, 0, 3, 0],
'c_max': [3, 2, 2, 3, 4, 2, 3, 4, 2, 3, 4, 3], 'n_id':
[1, 7, 5, 1, 3, 5, 1, 3, 5, 1, 3, 1]}: z=x*(y-(x/x))
20120) {'c_idx': [2, 1, 1, 0, 2, 1, 1, 1, 0, 0, 3, 1],
'c_max': [3, 2, 2, 3, 4, 2, 3, 4, 2, 3, 4, 3], 'n_id':
[1, 7, 5, 1, 3, 5, 1, 3, 5, 1, 3, 1]}: z=x*(y-(x/y))
!!! FOUND !!!
Duration: 0:09:40.685736
```

Таким образом, с помощью Прототипа без участия Эксперта удалось найти искомый ИК (т. е. используемый для компиляции исследуемого МК), который не только идентичен найденному Экспертом, но и получен за время около 10 минут (для рабочей машины – «Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz»), что всего в 2–3 раза больше ручного проведения РИ «редким» высококвалифицированным специалистом. Решение было синтезировано на 20120-й итерации, каждая из которых соответствует таким базовым действиям, как получение следующего пути по ГПС, генерация ИК (на основе пути), компиляция с получением МК и его сравнение с исследуемым. При этом время работы Прототипа может быть существенно уменьшено следующими способами:

– параллельное выполнение операций компиляции и сравнения полученного МК с исследуемым;

– проведение компиляции ИК не через запуск отдельного процесса утилиты сборки (т. е. «cl.exe» для среды Microsoft Visual Studio), а путем вызова функций предварительно загруженных динамических библиотек компилятора (т. е. «c1xx.dll» и «c2.dll»);

– применение пакетной обработки (когда на вход компилятора подается сразу группа исходных кодов, вместо одного);

– использование более мощного аппаратного обеспечения;

– оптимизация обхода ГСП в глубину (например, путем частичного движения по графу в ширину для первоочередного перебора ИК меньшего размера);

– использование иных форм внутреннего представления синтаксиса ЯП [10–12];

– учет семантики кода за счет создания и обхода соответствующих графов и иных структур, аналогичных ГСП.

Следовательно, можно сделать вывод, что по крайне мере для небольших блоков с кодом (исходного, машинного и ассемблерного) данный Прототип (а, следовательно, и Метод) даже в таком простейшем не оптимизированном виде применим для проведения РИ.

Проведем далее ряд исследований Прототипа для различных сценариев работы, используя синтаксис на Листинге 5; это позволит понять основные характеристики Метода с позиции оперативности – чем больше выполняется итераций, тем дольше работает РИ.

Сценарий 1. Зависимость всех вариантов ИК от МДП

В случае ГСП с циклами на количество путей, которые необходимо пройти Прототипу для достижения нужного варианта ИК (т. е. итераций его работы, включающих наиболее времязатратную операцию – компиляцию), влияет МДП, заданная через параметр Метода, поскольку она ограничивает обход графа в глубину [13]. Используя такой сценарий был проведен эксперимент, в котором вычислялось количество всех возможных генерируемых ИК для заданного синтаксиса при различных значениях данного параметра. также, была получена соответствующая формальная зависимость средствами Microsoft Excel (рисунок 2).

Согласно полученным результатам, количество вариантов ИК (y) в зависимости от МДП (x) растет экспоненциально: $y = 1{,}2062 \cdot e^{0{,}8272 \cdot x}$. Данная формула является достаточно точной, поскольку погрешность аппроксимации составляет 0,9866, т. е. близко к идеальной – единице.

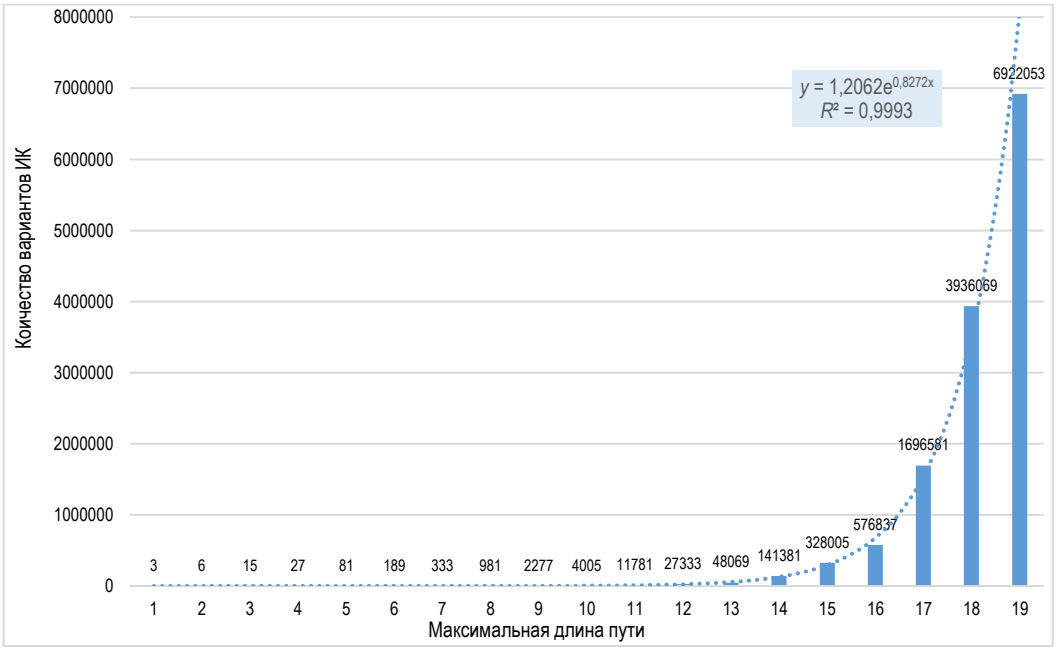


Рис. 2. Зависимость количества вариантов исходного кода от максимальной длины пути по графу синтаксических правил

Fig. 2. Dependence of Source Code Variants Number on the Maximum Path Length Along Syntax Rules Graph

Сценарий 2. Зависимость всех вариантов ИК от синтаксической разнородности

На время работы Прототипа, очевидно, влияет и размер самого ГСП. Прямая зависимость в данном случае вряд ли будет существовать, поскольку граф представляет собой нелинейную структуру, и количество обходов очевидно зависит от его топологии. Тем не менее, примерную оценку можно произвести для определенного синтаксиса, меняя количество простейших альтернатив, ведущих к терминалам; естественно, МДП также должна варьировать. Согласно синтаксису, на Листинге 5 для этого достаточно менять количество следующих вершин-«детей» ГСП: символов алфавита для «ident» (т. е. различных идентификаторов) и математических операций для «oper».

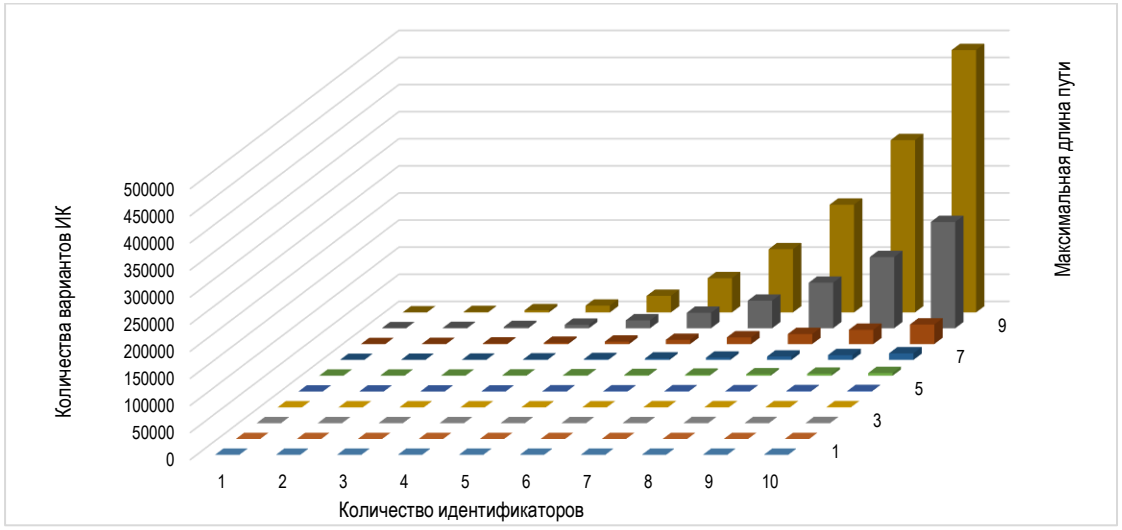
Такие зависимости, построенные с помощью Прототипа, приведены на рисунке 3. Визуальный анализ полученных закономерностей, а также влияние МДП на количество вариантов ИК (рисунок 2) позволяет сделать вывод, что зависимость от синтаксической разнородности в виде вариаций количества идентификаторов и математических операций также имеет экспоненциальный вид.

Сценарий 3. Зависимость времени поиска определенного ИК от МДП

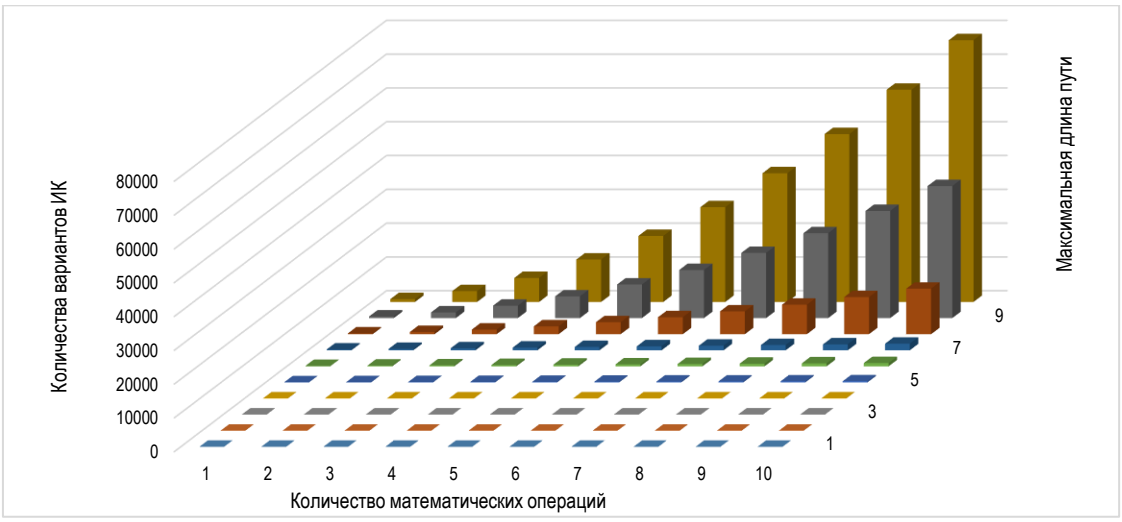
Поскольку для обнаружения ИК, соответствующего исследуемому МК, нет прямой необходимости перебора абсолютно всех путей по ГСП (в силу того, что ИК может быть найден раньше, вплоть до первых итераций) то целесообразным будет оценка того, как МДП будет влиять на оперативность подбора одного конкретного варианта ИК. Для этого выберем условно средний ИК, как и ранее состоящий из 13 символов (без учета пробелов) и построенный по синтаксису на Листинге 5, но полученный выбором средних путей в каждой из групп альтернатив. Таким образом, каждый идентификатор ИК будет равен «*y*», поскольку это средняя ветка ГСП согласно строке синтаксиса: «5: ident = 'x' | 'y' | 'z' ;». Так как в синтаксисе присутствует 4 оператора (нетерминал «oper»), то для корректного получения средней ветки данной группы альтернатив упростим правило до следующего – «4: oper = '+' | '-' | '*' ;» путем избавления от 4-й математической операции «/»; тогда все математическое операции в искомом ИК будут равны «-». В результате этого итоговый ИК, который будет искаться с помощью Прототипа, является следующим: $y = y - (y - (y - y))$.

Результат применения Прототипа в режиме вычисления количества итераций, необходимых для генерации по ГСП определенного выше ИК, в зависимости от МДП, представлен на рисунке 4.

Как хорошо видно, зависимость также является экспоненциальной, что вполне закономерно; тем не менее, количество итераций (для МДП, равной 12) сократилось до 13283 по сравнению с 20120, которое было получено ранее для синтаксиса с четырьмя математическими операциями и другими выборами альтернатив. Таким образом, оперативность Метода достаточно чувствительна к индексу альтернативы в ГСП, определяющих искомый ИК.



a)



b)

Рис. 3. Зависимость количества вариантов исходного кода от максимальной длины пути по графу синтаксических правил и количества идентификаторов (a), или математических операций (b)

Fig. 3. Dependence of Source Code Variants Number on the Maximum Path Length Along Syntax Rules Graph and Identifiers Number (a) or Mathematical Operations Number (b)

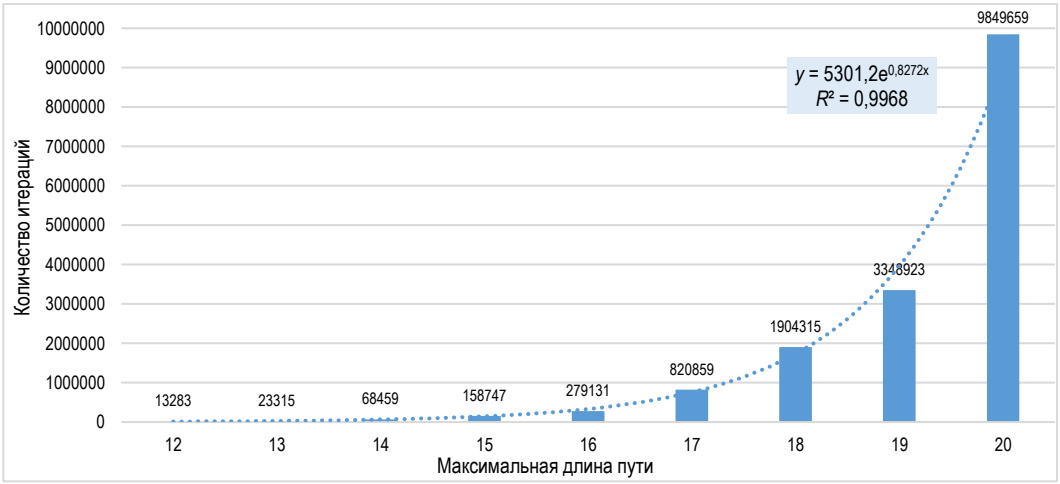


Рис. 4. Зависимость количества итераций для генерации искомого исходного кода от максимальной длины пути по графу синтаксических правил

Fig. 4. Dependence of Iterations Number for Generating the Required Source Code on the Maximum Path Length Along Syntax Rules Graph

Обсуждение результатов

Результаты базовых экспериментов показали, что количества итераций относительно различных параметров Метода растет экспоненциально, что существенно усложняет применение подхода «умного» перебора на практике. Различные же оптимизационные способы (например, параллельный запуск потоков или процессов) вряд ли позволят «сдержать» такое увеличение, поскольку носят линейный характер. Тем не менее, возможен переход от «умного» перебора к интеллектуальному путем применения генетических алгоритмов [14], что было продемонстрировано в различных статьях авторов, посвященных направлению генетического РИ [15, 16].

Одним из условий решения задачи РИ путем «умного» перебора является корректный выбор МДП по ГСП (например, для синтаксиса на Листинге 5 такая длина была получена эмпирически). Однако, МДП коррелирует с длиной самого ИК (через синтаксис) [17], которая, в свою очередь, согласно предыдущим авторским исследованиям, может быть получена из длины МК [18]. Данная связь позволяет прогнозировать МДП для генерации ИК, исходя из размера исследуемого МК.

Принцип работы Шага 1 «Построение ГСП» в работе практически не был рассмотрен; однако его реализация является чисто технической задачей [19], поскольку существует достаточное количество программных библиотек, осуществляющих разбор БНФ-синтаксиса (например, проекты на Интернет-ресурсах https://github.com/shnewto/bnf, https://bnf-parser.ajanibilby.com и др.).

По очевидным причинам конкретные имена идентификаторов ИК в процессе компиляции теряются, поскольку МК оперирует регистрами, стеком и областями памяти; таким образом, восстановить идентификаторы из МК не представляется возможным. Например, следующие 2 ИК для сложения пары чисел после компиляции будут иметь одинаковый МК:

```
int s(int x, int y) {
  return x + y;
}
и
int sum(int first, int second) {
  return first + second;
}
```

Однако, Прототип вместо попытки «угадывания» реальных имен переменных может использовать автоматически генерируемые, и, в частности, состоящие из одного символа (поскольку с большой вероятностью 52 символа окажется достаточным для именования переменных в любой, даже сверх сложной, программе); для назначения имен функциям может быть использовать такой же подход. При этом, если частично отказаться от инвариантности Метода к синтаксису ЯП, то существуют решения по автоматической генерации имен функций на основании метаинформации из ИК [20, 21].

Отдельным развитием смарт-перебора в сторону качественно иной интеллектуализации может оказаться использование больших языковых моделей (в англоязычной литературе более принято сокращение от термина «large Language Model» – LLM, используемое далее), заменяющих в некотором смысле экспертное управление алгоритмом и настройку его параметров. Так, «сырая» обработка ассемблерного (или даже бинарного) кода с помощью LLM позволит предположить наиболее вероятные значение подбираемых констант. А предварительное обучение модели заданным синтаксисом ЯП позволит Методу выбрать более предпочтительные ветви ГСП или отдельные последовательности альтернатив, что существенно сократит время подбора нужного ИК по заданному МК.

Важно отметить, что уже сейчас применение LLM для непосредственной декомпиляции МК имеет определенную эффективность, позволяющую за короткое время получать ИК. Тем не менее, у данного подхода есть ряд существенных проблемных вопросов. Во-первых, из-за эффекта «галлюционирования» в непредсказуемых случаях LLM будет генерировать ИК, логика которого существенно отличаются от логики исходного. Во-вторых, отсутствует гарантия, что полученный ИК действительно компилируется в заданный МК, поскольку для такой проверки по крайне мере придется корректно определить первоначальный тип компилятора, его версию и параметры. И, в-третьих, в случае малоизвестной или новой процессорной архитектуры, LLM не сможет адекватно производить декомпиляцию, поскольку модель банально не была обучена на нужных датасетах.

В приведенных примерах использовался ИК лишь с математическими выражениями в одной функции без сложных потоков управления, полученных в случае использования циклов, вызовов других функций и т. п.; однако это не является критичным по следующим причинам. Во-первых, осуществлялась общая проверка работоспособности Метода и Прототипа, чтобы получить общее понимание – есть ли обоснованное развитие у предложенного подхода. Во-вторых, проводился ряд оценок Метода для выявления его наиболее негативных сторон, что вначале целесообразнее было произвести на простых примерах, поскольку полученная для них экспоненциальная зависимость времени работы Метода будет иметь не «лучшую» закономерность и на сложных примерах. И, в-третьих, даже в случае ограничения Метода подбором только математических выражений (без условных переходов и вызовов функций), такое решение по РИ, безусловно, будет иметь

практическую значимость, поскольку восстановление подобного МК также является сложной задачей для Эксперта; при этом, существует достаточное количество решений по преобразованию графов потоков управления [22] для МК в визуальные блок-схемы, элементы которых как раз и содержат математические выражения с проверками условий для переходов.

Заключение

В работе завершен цикл исследования по применению полного перебора экземпляров ИК с целью получения МК, идентичного исследуемому – т. е. решение задачи РИ с помощью «неклассического» подхода, отличающегося сверхвысоким временем работы [23]; как результат, рассмотрена возможность декомпиляции МК автоматическим способом в ИК, гарантированно в него компилируемый. Также, произведена оптимизация полного перебора путем генерации ИК согласно заданного синтаксиса ЯП, что делает перебор «умным» (т. е. оптимальным, но не интеллектуальным). Проведенные эксперименты позволили получить общие характеристики подхода и выявили его слабые стороны.

Основным научным результатом является алгоритм работы Метода (заданный в форме интуитивно-понятного псевдокода), реализованный в виде прототипа «умного» перебора. Базовое тестирование показало работоспособность Прототипа (и, следовательно, Метода), а ряд экспериментов позволил количественно (в границах выбранных сценариев) оценить его оперативность и влияющие на нее факторы.

Отличительной особенностью Прототипа (как и самого Метода) является то, что для его работы, помимо утилиты компиляции, необходимы лишь синтаксис и лексика ЯП, значение МДП по ГСП, а также МК, РИ которого необходимо произвести. Таким образом, в отличие от аналогов, прототип может работать без участия человека и не зависеть как от ЯП, так и от архитектуры ЦПУ (так, например, продукт для РИ IDA Pro [24] на момент июля 2024 г. поддерживает лишь 5 типов ЦПУ – x86, x64, ARM32, ARM64 и PowerPC, восстанавливая из МК лишь ИК на С-подобном ЯП).

Теоретическая значимость исследования состоит в непосредственном получении алгоритма работы Прототипа, являющегося некоторым шаблоном для создания различных реализаций вариаций подхода полного перебора. Кроме того, инвариантность логики работы Прототипа именно от синтаксиса ИК позволит продолжить его развитие и для более высокоуровневых представлений программ, таких, как алгоритмы, архитектура и пр. [25].

Практическая значимость исследования заключается в возможности непосредственного использования Прототипа для проведения РИ реальных экземпляров МК [26], которые, хотя и должны удовлетворять ряду ограничений (имеется ввиду поддержка только математических выражений без условных переходов), но все также пока являются существенной проблемой при проведении РИ. При этом от оператора не требуется обладание существенными знаниями про нотацию как ИК, так и МК, что существенно снижает потребность в редких и высококвалифицированных экспертах.

Продолжением работы будет переход от метода «умного» перебора ИК в интересах РИ к методу интеллектуального перебора (т. е. дальнейшая, качественно новая, оптимизация [27]) на основе генетического РИ с использованием решений, представленных в текущем исследовании. На данный момент у авторов уже имеется программный прототип такого решения, показывающий существенно лучшие результаты, чем прямой перебор ИК; детальная информация же на эту тему будет опубликовано в ближайшее время.

Кроме того, отдельным направлением исследований авторов будет общее применение LLM для РИ как в качестве расширения приведенного смарт-перебора, так и при создание новых подходов к декомпиляции.

Список источников

1. Комаров В.В., Мезинова Н.А., Евдокимова Е.А. Анализ условий реализации угроз безопасности информации через эксплуатацию уязвимостей информационных активов // Научно-аналитический журнал «Вестник Санкт-Петербургского университета Государственной противопожарной службы МЧС России». 2024. № 2. С. 126–135. DOI:10.61260/2218-130X-2024-2-126-135. EDN:NRXXGV

2. Ле Ч.Д., Фам М.Х., Динь Ч.З., До Х.Ф. Применение алгоритмов машинного обучения для обнаружения вредоносных программ в операционной системе Windows с помощью PE-заголовка // Информационно-управляющие системы. 2022. № 4(119). С. 44–57. DOI:10.31799/1684-8853-2022-4-44-57. EDN:YFIBQJ

3. Израилов К.Е., Буйневич М.В. Реверс-инжиниринг программного обеспечения методом смарт-перебора: пошаговая схема // Труды учебных заведений связи. 2025. Т. 11. № 4. С. 129–142. DOI:10.31854/1813-324X-2025-11-4-129-142. EDN:UOKLHB

4. Putro H.P., Yuhana U.L., Yuniarno E.M., Purnomo M.H. Source Code Statement Classification Using ANTLR and Random Forest // Proceedings of the International Seminar on Intelligent Technology and Its Applications (ISITIA, Surabaya, Indonesia, 26–27 July 2023). IEEE, 2023. PP. 60–65. DOI:10.1109/ISITIA59021.2023.10220999

5. Fu J., Zhang K., Zheng J., Li W., Zhu Y. Research and Application of Grey Box Detection Technology Based on Reverse Engineering and Dynamic Pollution Diffusion // Proceedings of the 7th Information Technology and Mechatronics Engineering Conference (ITOEC, Chongqing, China, 15–17 September 2023). IEEE, 2023. PP. 2380–2384. DOI:10.1109/ITOEC57671.2023.10291380

6. Devine T.R., Campbell M., Anderson M., Dzielski D. SREP+SAST: A Comparison of Tools for Reverse Engineering Machine Code to Detect Cybersecurity Vulnerabilities in Binary Executables // Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI, Las Vegas, USA, 14–16 December 2022). IEEE, 2022. PP. 862–869. DOI:10.1109/CSCI58124.2022.00156

7. Hu Y., Wang H., Zhang Y., Li B., Gu D. A Semantics-Based Hybrid Approach on Binary Code Similarity Comparison // Transactions on Software Engineering. 2021. Vol. 47. Iss. 6. PP. 1241–1258. DOI:10.1109/TSE.2019.2918326. EDN:ILNITT

8. Adamchuk N., Schlüter W. Automatic Acceptor Generation Based on EBNF Grammar Definition // Proceedings of the 11th International Conference on Advanced Computer Information Technologies (ACIT, Deggendorf, Germany, 15–17 September 2021). IEEE, 2021. PP. 618–622. DOI:10.1109/ACIT52158.2021.9548492

9. Савченко А.А., Минеева Т.А. Язык программирования ассемблер. Разница низкоуровневых и высокоуровневых языков // Тенденции развития науки и образования. 2022. № 92-10. С. 131–135. DOI:10.18411/trnio-12-2022-502. EDN:QMZFNE

10. Нечесов А.В. Некоторые вопросы полиномиально вычислимых представлений для порождающих грамматик и форм Бэкуса-Наура // Математические труды. 2022. Т. 25. № 1. С. 134–151. DOI:10.33048/mattrudy.2022.25.106. EDN:SFDFPB

11. Рязанов Ю.Д., Назин С.В. Построение синтаксических анализаторов на основе синтаксических диаграмм с многовходовыми компонентами // Прикладная дискретная математика. 2022. № 55. С. 102–119. DOI:10.17223/20710410/55/8. EDN:XHAFEV

12. Третьяк А.В., Третьяк Е.В., Верещагина Е.А. Разработка когнитивно-эргономического синтаксиса для нового аппаратно-ориентированного языка программирования // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки. 2020. № 7. С. 145–153. DOI:10.37882/2223-2966.2020.07.33. EDN:GVAAGG

13. Костенко М.С., Цицарева В.В. Применение эффективных методов обхода графа (поиск в глубину, поиск в ширину) при решении задач второго этапа республиканской олимпиады по учебному предмету «Информатика» // Современное образование Витебщины. 2024. № 2(44). С. 24–26. EDN:WRZIGQ

14. Загинайло М.В., Фатхи В.А. Генетический алгоритм как эффективный инструмент эволюционных алгоритмов // Инновации. Наука. Образование. 2020. № 22. С. 513–518. EDN:UTMAEL

15. Израилов К.Е. Концепция генетической деэволюции представлений программы. Часть 1 // Вопросы кибербезопасности. 2024. № 1(59). С. 61–66. DOI:10.21681/2311-3456-2024-1-61-66. EDN:CBCKRF

16. Израилов К.Е. Концепция генетической деэволюции представлений программы. Часть 2 // Вопросы кибербезопасности. 2024. № 2(60). С. 81–86. DOI:10.21681/2311-3456-2024-2-81-86. EDN:JUBPML

17. He H., Lin L., Yu T., Zhong X. CloneBAS: A Code Clone Detection Method Based on Abstract Syntax Tree and Simhash // Proceedings of the 3rd International Conference on Data Science and Computer Application (ICDSCA, Dalian, China, 27–29 October 2023). IEEE, 2023. PP. 1539–1544. DOI:10.1109/ICDSCA59871.2023.10392292

18. Izrailov K. GREMC: Genetic Reverse-Engineering of Machine Code to Search Vulnerabilities in Software for Industry 4.0. Predicting the Size of the Decompiling Source Code // Proceedings of the International Russian Smart Industry Conference (SmartIndustryCon, Sochi, Russian Federation, 25–29 March 2024. IEEE, 2024. PP. 622–628. DOI:10.1109/ SmartIndustryCon61328.2024.10515515

19. Миронов С.В., Батраева И.А., Дунаев П.Д. Библиотека для разработки компиляторов // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 77–88. DOI:10.15514/ISPRAS-2022-34(5)-5. EDN:JPGPIY

20. Qu Z., Hu Y., Zeng J., Cai B., Yang S. Method Name Generation Based on Code Structure Guidance // Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER, Honolulu, USA, 15–18 March 2022). IEEE, 2022. PP. 1101–1110. DOI:10.1109/SANER53432.2022.00127

21. Petukhov M., Gudauskayte E., Kaliyev A., Oskin M., Ivanov D., Wang Q. Method Name Prediction for Automatically Generated Unit Tests // Proceedings of the International Conference on Code Quality (ICCQ, Innopolis, Russian Federation, 23 April 2022). IEEE, 2022. PP. 29–38. DOI:10.1109/ICCQ53703.2022.9763112. EDN:TOCMXI

22. Бородин А.В., Юдина М.А., Васильева М.А. О задаче классификации на окрестности корня графа потока управления программы в контексте процесса размножения файловых компьютерных вирусов // Современные наукоемкие технологии. 2019. № 1. С. 31–35. EDN:VUCEWK

23. Куделя В.Н. Методы перечисления путей в графе // Наукоемкие технологии в космических исследованиях Земли. 2023. Т. 15. № 5. С. 28–38. DOI:10.36724/2409-5419-2023-15-5-28-38. EDN:HQEASN

24. Кусаинов А.Р., Глазырина Н.С. Обзор инструментов статического анализа программного кода // Colloquium-Journal. 2020. № 32-1(84). С. 48–52. EDN:JXSKQX

25. Kotenko I., Izrailov K., Buinevich M., Saenko I., Shorey R. Modeling the Development of Energy Network Software, Taking into Account the Detection and Elimination of Vulnerabilities // Energies. 2023. Vol. 16. Iss. 13. P. 5111. DOI:10.3390/en16135111. EDN:CFRQLO

26. Пичугова Л.Н. Перспективные технологии реверс-инжиниринга и быстрого прототипирования // Фундаментальные основы механики. 2023. № 11. С. 43–48. DOI:10.26160/2542-0127-2023-11-43-48. EDN:CYVEES

27. Аралбаев Р.А., Тарасов А.А. Задачи оптимизации и применение алгоритмов генетический алгоритм на практике // Инновации. Наука. Образование. 2021. № 48. С. 1645–1653. EDN:VGUBIH

References

1. Komarov V., Mesinova N., Evdokimova E. Analysis of the conditions for the implementation of information security threats through the exploitation of information asset vulnerabilities. *Scientific and Analytical Journal «Vestnik Saint-Petersburg University of State Fire Service of EMERCOM of Russia»*. 2024;2:126–135. (in Russ.) DOI:10.61260/2218-130X-2024-2-126-135. EDN:NRXXGV

2. Le T.D., Pham M.H., Dinh T.D., Do H.P. Applying machine learning algorithms for PE-header-based malware detection on the Windows operating system. *Information and Control Systems.* 2022;4(119):44–57. (in Russ.) DOI:10.31799/1684-8853-2022-4-44-57. EDN:YFIBQJ

3. Izrailov K.E., Buinevich M.V. Reverse Engineering of Software Using the Smart Brute Force Method: Step-by-Step Scheme. *Proceedings of Telecommunication Universities.* 2025;11(4):129–142. (in Russ.) DOI:10.31854/1813-324X-2025-11-4-129-142. EDN:UOKLHB

4. Putro H.P., Yuhana U.L., Yuniarno E.M., Purnomo M.H. Source Code Statement Classification Using ANTLR and Random Forest. *Proceedings of the International Seminar on Intelligent Technology and Its Applications, ISITIA, 26–27 July 2023, Surabaya, Indonesia*. IEEE; 2023. p.60–65. DOI:10.1109/ISITIA59021.2023.10220999

5. Fu J., Zhang K., Zheng J., Li W., Zhu Y. Research and Application of Grey Box Detection Technology Based on Reverse Engineering and Dynamic Pollution Diffusion. *Proceedings of the 7th Information Technology and Mechatronics Engineering Conference, ITOEC, 15–17 September 2023, Chongqing, China*. IEEE; 2023. p.2380–2384. DOI:10.1109/ITOEC57671.2023.10291380

6. Devine T.R., Campbell M., Anderson M., Dzielski D. SREP+SAST: A Comparison of Tools for Reverse Engineering Machine Code to Detect Cybersecurity Vulnerabilities in Binary Executables. *Proceedings of the International Conference on Computational Science and Computational Intelligence, CSCI, 14–16 December 2022, Las Vegas, USA*. IEEE; 2022. p.862–869. DOI:10.1109/CSCI58124.2022.00156

7. Hu Y., Wang H., Zhang Y., Li B., Gu D. A Semantics-Based Hybrid Approach on Binary Code Similarity Comparison. *Transactions on Software Engineering.* 2021;47(6):1241–1258. DOI:10.1109/TSE.2019.2918326. EDN:ILNITT

8. Adamchuk N., Schlüter W. Automatic Acceptor Generation Based on EBNF Grammar Definition. *Proceedings of the 11th International Conference on Advanced Computer Information Technologies, ACIT, 15–17 September 2021, Deggendorf, Germany*. IEEE; 2021. p.618–622. DOI:10.1109/ACIT52158.2021.9548492

9. Savchenko A.A., Mineeva T.A. Assembly Programming Language. The Difference Between Low-Level and High-Level Languages. *Tendencii razvitija nauki i obrazovanija.* 2022;92-10:131–135. (in Russ.) DOI:10.18411/trnio-12-2022-502. EDN:QMZFNE

10. Nechesov A.V. Some questions on polynomially computable representations for generating grammars and Backus-Naur forms. *Mathematical works*. 2022;25(1):134–151. (in Russ.) DOI:10.33048/mattrudy.2022.25.106. EDN:SFDFPB

11. Ryazanov Yu.D., Nazina S.V. Building parsers based on syntax diagrams with multiport components. *Applied Discrete Mathematics*. 2022;55:102–119 (in Russ.) DOI:10.17223/20710410/55/8. EDN:XHAFEV

12. Tretyak A.V., Tretyak E.V., Vereshchagina E.A. Development of cognitive-ergonomic syntax for a new hardware-oriented programming language. *Modern Science: Actual Problems of Theory and Practice". Series "Natural and Technical Sciences"*. 2020;7:145–153. (in Russ.) DOI:10.37882/2223-2966.2020.07.33. EDN:GVAAGG

13. Kostenko M.S., Cicareva V.V. Application of Efficient Graph Traversal Methods (Depth-First Search, Breadth-First Search) in Solving Problems of the Second Stage of the Republican Olympiad in the Subject "Computer Science". *Sovremennoe obrazovanie Vitebshhiny*. 2024;2(44):24–26. (in Russ.) EDN:WRZIGQ

14. Zaginajlo M. V., Fathi V. A. Genetic Algorithm as an Effective Tool for Evolutionary Algorithms. *Innovacii. Nauka. Obrazovanie.* 2020;22:513–518 (in Russ.) EDN:UTMAEL

15. Izrailov K.E. The genetic de-evolution concept of program representations. Part 1. *Voprosy kiberbezopasnosti*. 2024;1(59):61–66. (in Russ.) DOI:10.21681/2311-3456-2024-1-61-66. EDN:CBCKRF

16. Izrailov K.E. The genetic de-evolution concept of program representations. Part 2. *Voprosy kiberbezopasnosti*, 2024;2(60):81–86. (in Russ.) DOI:10.21681/2311-3456-2024-2-81-86. EDN:JUBPML

17. He H., Lin L., Yu T., Zhong X. CloneBAS: A Code Clone Detection Method Based on Abstract Syntax Tree and Simhash. *Proceedings of the 3rd International Conference on Data Science and Computer Application, ICDSCA, 27–29 October 2023, Dalian, China*. IEEE; 2023. p.1539–1544. DOI:10.1109/ICDSCA59871.2023.10392292

18. Izrailov K. GREMC: Genetic Reverse-Engineering of Machine Code to Search Vulnerabilities in Software for Industry 4.0. Predicting the Size of the Decompiling Source Code. *Proceedings of the International Russian Smart Industry Conference, SmartIndustryCon, 25–29 March 2024, Sochi, Russian Federation*. IEEE; 2024. p.622–628. DOI:10.1109/SmartIndustryCon61328.2024.10515515

19. Mironov S.V., Batraeva I.A., Dunaev P.D. Library for development of compilers. *Proceedings of the Institute for System Programming of the RAS.* 2022;34(5):77–88 (in Russ.). Doi:10.15514/ISPRAS-2022-34(5)-5. EDN:JPGPIY

20. Qu Z., Hu Y., Zeng J., Cai B., Yang S. Method Name Generation Based on Code Structure Guidance. *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering, SANER, 15–18 March 2022, Honolulu, USA*. IEEE; 2022. p.1101–1110. DOI:10.1109/SANER53432.2022.00127

21. Petukhov M., Gudauskayte E., Kaliyev A., Oskin M., Ivanov D., Wang Q. Method Name Prediction for Automatically Generated Unit Tests. *Proceedings of the International Conference on Code Quality, ICCQ, 23 April 2022, Innopolis, Russian Federation*. IEEE; 2022. p.29–38. DOI:10.1109/ICCQ53703.2022.9763112. EDN:TOCMXI

22. Borodin A.V., Yudina M.A., Vasileva M.A. About the problem of classification on the neighborhood of the root of the control flow graph of the program in the context of process of reproduction of file computer viruses. *Modern High Technologies*. 2019;1:31–35 (in Russ.) EDN:VUCEWK

23. Kudelya V.N. Methods for enumerating paths in a graph. *H&ES Research.* 2023;15(5):28–38 (in Russ.) DOI:10.36724/2409-5419-2023-15-5-28-38. EDN:HQEASN
24. Kussainov A.R., Glazyrina N.S. Overview of static program code analysis tools. *Colloquium-Journal*. 2020;32-1(84):48–52. (in Russ.) EDN:JXSKQX
25. Kotenko I., Izrailov K., Buinevich M., Saenko I., Shorey R. Modeling the Development of Energy Network Software, Taking into Account the Detection and Elimination of Vulnerabilities. *Energies.* 2023;16(13):5111. DOI:10.3390/en16135111. EDN:CFRQLO
26. Pichugova L.N. Perspective technologies of reverse engineeringand fast prototyping. *Fundamentalnye osnovy mehaniki.* 2023;11:43–48. (in Russ.) DOI:10.26160/2542-0127-2023-11-43-48. EDN:CYVEES
27. Aralbaev R.A., Tarasov A.A. Optimization Problems and Application of Genetic Algorithms in Practice. *Innovacii. Nauka. Obrazovanie.* 2021;48:1645–1653. (in Russ.) EDN:VGUBIH

Статья поступила в редакцию 09.07.2025; одобрена после рецензирования 16.12.2025; принята к публикации 22.12.2025.
The article was submitted 09.07.2025; approved after reviewing 16.12.2025; accepted for publication 22.12.2025.

Информация об авторах:

ИЗРАИЛОВ Константин Евгеньевич
кандидат технических наук, доцент, профессор кафедры прикладной математики и безопасности информационных технологий Санкт-Петербургского университета государственной противопожарной службы МЧС России
https://orcid.org/0000-0002-9412-5693

БУЙНЕВИЧ Михаил Викторович
доктор технических наук, профессор, профессор кафедры КБ-4 МИРЭА – Российского технологического университета
https://orcid.org/0000-0001-8146-0022

Буйневич М.В. является членом редакционного совета журнала «Труды учебных заведений связи» с 2016 г., но не имеет никакого отношения к решению опубликовать эту статью. Статья прошла принятую в журнале процедуру рецензирования. Об иных конфликтах интересов авторы не заявляли.

Buinevich M.V. has been a member of the journal "Proceedings of Telecommunication Universities" Editorial Council since 2016, but has nothing to do with the decision to publish this article. The article has passed the review procedure accepted in the journal. The authors have not declared any other conflicts of interest.