Научная статья
УДК 004.41
https://doi.org/10.31854/1813-324X-2025-11-1-84-98
EDN:URSGXI

Проблемные вопросы генетической деэволюции представлений программы для поиска уязвимостей и рекомендации по их разрешению

Константин Евгеньевич Израилов, konstantin.izrailov@mail.ru

Санкт-Петербургский Федеральный исследовательский центр Российской академии наук,
Санкт-Петербург, 199178, Российская Федерация

Аннотация

Актуальность *темы обосновывается отсутствием методологии реверс-инжиниринга программного обеспечения, необходимой для разрешения следующего научного противоречия области (как противопоставления потребности vs возможности): с одной стороны поиск уязвимостей наиболее эффективен в тех представлениях программы, в которых они были внедрены (например, исходный код, алгоритмы или архитектура); с другой стороны, для анализа имеется, как правило, лишь машинный код, слабо подходящий для выявления высокоуровневых уязвимостей (т.е. из более ранних представлений). Созданию составляющих данной методологии (концепции, модели, метода, алгоритмов, метрики, а также их реализаций) и посвящено основное авторское исследование, заключительный этап которого приводится в статье.*

Целью *настоящей статьи является обсуждение 25 проблемных вопросов (так называемая научная дискуссия), возникших в основном исследовании, посвященном развитию направления реверс-инжиниринга программного обеспечения на базе генетических алгоритмов. Основным применением результатов исследования является как получение представления программы, подходящего для экспертного (и иного) анализа на предмет наличия в нем уязвимостей, так и их непосредственный поиск встроенным сигнатурным методом. При этом разрешение даже части вопросов позволит существенно* ***повысить эффективность*** *такого генетического реверс-инжиниринга.*

В работе использованы следующие ***методы****: анализ результатов основного исследования для выделения проблемных вопросов, синтез путей их разрешения, а также систематизация и балльное сравнение вопросов с позиции путей устранения для общей оценки завершенности научной работы.*

Детальное ***изучение*** *причин возникновения каждого из вопросов позволило определить пути их разрешения, реализуемость которых обосновывает и результаты основного исследования. В частности, проблемные вопросы* ***базируются*** *как на отсутствии одних теоретических инструментов, необходимых для генетического реверс-инжиниринга, так и на недостаточной практической эффективности других.*

Научная новизна *проблемных вопросов заключается в том, что практически каждый из них озвучен впервые.*

Теоретическая значимость**:** *развитие каждого проблемного вопроса может как открыть отдельное научное исследование (или даже направление), так и получить новые значимые результаты.*

Практическая значимость *заключается в возможности создания программных решений по разрешению выявленных вопросов, которые могут быть также применены и для смежных задач.*

Ключевые слова: *информационная безопасность, уязвимость, программа, реверс-инжиниринг, генетический алгоритм, деэволюция, декомпиляция, проблемные вопросы*

Источник финансирования: *Работа выполнена при частичной финансовой поддержке бюджетной темы FFZF-2025-0016.*

Ссылка для цитирования: Израилов К.Е. Проблемные вопросы генетической деэволюции представлений программы для поиска уязвимостей и рекомендации по их разрешению // Труды учебных заведений связи. 2025. Т. 11. № 1. С. 84–98. DOI:10.31854/1813-324X-2025-11-1-84-98. EDN:URSGXI

Original research
https://doi.org/10.31854/1813-324X-2025-11-1-84-98
EDN:URSGXI

Problematic Issues of a Program Representations Genetic De-Evolution for Search Vulnerabilities and Recommendations for Its Resolution

Konstantin E. Izrailov, konstantin.izrailov@mail.ru

Saint-Petersburg Federal Research Center of the Russian Academy of Sciences,
St. Petersburg, 199178, Russian Federation

Annotation

The relevance *of the topic is justified by the software reverse engineering methodology lack required to resolve the following scientific contradiction in the field (as a contrast between need and opportunity): on the one hand, vulnerability search is most effective in those program representations in which they were implemented (e.g. source code, algorithms or architecture); on the other hand, as a rule, only machine code is available for analysis, which is poorly suited for identifying high-level vulnerabilities (i.e. from earlier representations). The main author's study, the final stage of which is given in the article, is devoted to the creation of the this methodology elements (concept, model, method, algorithms, metrics, as well as their implementations).*

The purpose *of this article is to discuss 25 problematic issues (the so-called scientific discussion) that arose in the main study devoted to the software reverse engineering development based on genetic algorithms. The main application of the research results is both obtaining a program representation suitable for expert (and other) analysis for vulnerabilities, and their direct search using the built-in signature method. At the same time, resolving even a part of the issues will significantly* ***increase the efficiency*** *of such genetic reverse engineering.*

The following ***methods*** *were used in the work: the main research results analysis to identify problematic issues, ways to resolve them synthesis, as well as issues systematization and scoring from the standpoint of ways to eliminate them for an overall assessment of the scientific work completeness.*

A causes of each issue detailed ***research*** *allowed us to determine ways to resolve them, the feasibility of which also justifies the main research results. In particular, problematic issues are* ***based both on*** *some absence of theoretical tools necessary for genetic reverse engineering, and on the insufficient practical efficiency of others.*

The scientific novelty *of the issues lies in the fact that almost each of them is voiced for the first time.*

The theoretical significance *lies in the fact that the development of each problematic issue can both open a separate scientific study (or even a direction), and obtain new significant results.*

The practical significance *lies in the possibility of creating software solutions to resolve identified issues, which can also be applied to related tasks.*

Keywords: *information security, vulnerability, program, reverse engineering, genetic algorithm, de-evolution, decompilation, problematic issues*

Funding: *The work was partially funded by the budget project FFZF-2025-0016.*

For citation: Izrailov K.E. Problematic Issues of a Program Representations Genetic De-evolution for Search Vulnerabilities and Recommendations for Its Resolution. *Proceedings of Telecommunication Universities.* 2025;11(1): 84–98. (in Russ.) DOI:10.31854/1813-324X-2025-11-1-84-98. EDN:URSGXI

Введение

Наличие уязвимостей в программном обеспечении (далее – ПО) является актуальнейшей проблемой в области информационных технологий. Несмотря на наличие определенного пула качественно разных способов их поиска (например, применяя сигнатурный и эвристический анализ [1], нечеткие хэши [2] и пр.), все они обладают соб-

ственными недостатками, что не позволяет создать «идеальный» способ. Как результат, исследования в данном направлении информационной безопасности (далее – ИБ) еще далеки от завершения; притом, в условиях, когда злоумышленники, сознательно внедряющие уязвимости в программный код, непрерывно совершенствуют свои методы и инструменты. Одним из применяемых подходов является *реверс-инжиниринг ПО*, классически заключающийся в преобразовании машинного кода программы (далее – МК), слабо пригодного для ручного анализа, в более высокоуровневое представление исходного кода (далее – ИК), которое поддается анализу на предмет наличия уязвимостей эксперту по безопасности ПО (далее – Эксперт); такой подход носит название *декомпиляции* программы.

Концепция качественно нового авторского способа реверс-инжиниринга ПО (далее – Концепция), основанная на применении генетического алгоритма (далее – ГА), уже имеет определенную доказательную базу в виде большого пула публикаций, посвященных теоретическим изысканиям и практическим реализациям в виде программного прототипа (далее – Прототип), преобразующего при ряде ограничений заданный МК в соответствующий ему ИК. Тем не менее, как и для любой инновационной работы, в реализации Концепции существуют ряд проблемных вопросов, которым и посвящена данная статья.

Результаты исследования

Прежде чем кратко описать полученные результаты в рамках создания Концепции, укажем используемые в ней термины и понятия.

Терминология

Поскольку в основу Концепции положено применение ГА, то часть основных связанных с этим термином имеет слово «генетический».

По аналогии с существующей в области терминологией, для получения ИК из МК применяется *генетическая декомпиляция* (далее – ГДК), суть которой заключается в получении предыдущего представления программы из текущего, которое имеет бинарный вид инструкций процессора (естественно, с помощью ГА [3]).

Расширение Концепции на любые формы программы, используемые в рамках программной инженерии (например, алгоритмы и архитектуру), позволяют говорить о получении каждого предыдущего представления из текущего (а не только ИК из МК), т. е. о *генетической деэволюции* пары представлений (далее – ГДЭ) [4, 5]. Необходимость в ГДЭ более высокоуровневых представлений обосновывается тем, что поиск уязвимостей наиболее эффективен в том представлении, в котором они были заложены в программу (как сознательно, так и случайно).

И, наконец, весь процесс последовательности преобразований пар представлений (например, от МК до концептуальной модели программы) в рамках Концепции назван генетическим *реверс-инжинирингом* (далее – ГРИ). Данный термин и определяет предметную область авторского исследования – проведение ГРИ программы в интересах поиска в ней уязвимостей. Очевидно, что ГРИ является диаметрально противоположным процессом классической программной инженерии.

Соответствие основных «генетических» терминов предметной области отображено на рисунке 1; обозначение «Представление P_i» соответствует *i*-му представлению программы. При этом отличительной особенностью ГРИ от аналогичных решений является гипотетическая независимость его алгоритмов от синтаксисов представлений, а также получение программы, в точности собираемой (в случае ИК – компилируемой) в исследуемую.

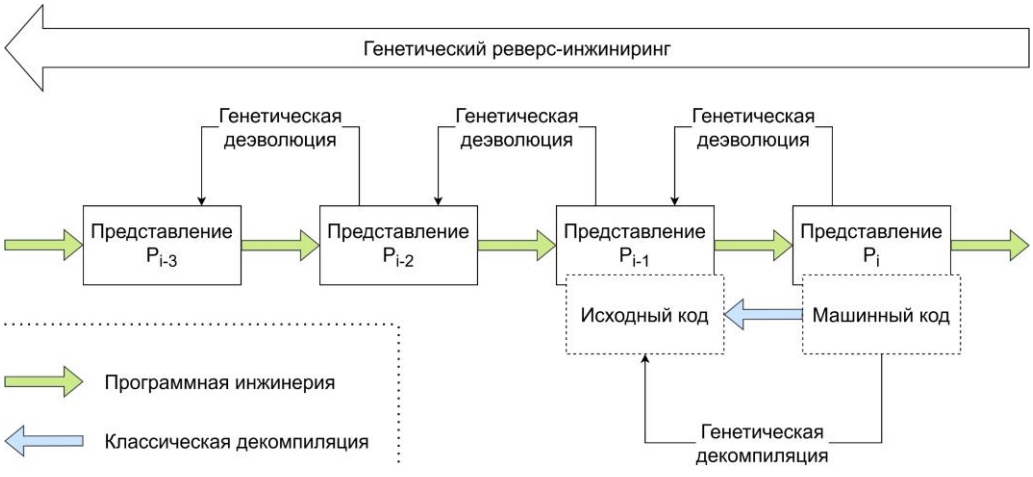

Рис. 1. Соответствие основных терминов предметной области

Fig. 1. Correspondence of the Subject Area Main Terms

Кроме того, несмотря на то, что современные декомпиляторы в ряде случаев справляются с получением ИК по МК существенно лучше (хотя и получают зачастую лишь псевдо ИК, не всегда компилируемый), однако их реализации работают с крайне ограниченным набором пар представлений. К примеру, плагин Hex-Rays, входящий в продукт IDA Pro, в максимальной комплектации позволяет декомпилировать только 6 процессорных архитектур (x86, ARM, MIPS, PowerPC, ARC и RICS) и только в C-подобный псевдокод (https://hex-rays.com/pricing), хотя общее количество как процессоров, так и языков программирования (далее – ЯП) достаточно большое, не говоря уже об их комбинации.

На текущий момент в области поиска уязвимостей путем реверс-инжиниринга основной задачей является именно получение ИК из МК, поскольку большинство ПО, используемого в критических областях (например, в киберфизических системах или встроенных устройствах) не имеет открытых текстов ИК. Поэтому, хотя далее и будут приведены основные полученные результаты и проблемные вопросы для ГДЭ (составляющих весь ГРИ), основной целью доработки Прототипа и Концепции является осуществление именно ГДК.

Генетическая деэволюция представлений

Общий принцип ГДЭ (являющейся обобщением ГДК на любые представления и составляющей весь процесс ГРИ), как было указано, основан на работе ГА, которым посвящено достаточное количество публикаций [6]. Так, суть ГА заключается в итеративном решении оптимизационной задачи по нахождению экстремума целевой функции путем генерации большого количества особей популяции, каждая из которых задает одно из возможных (вначале неверных) решений задачи. Каждая особь сопоставляется с некоторой хромосомой, гены которой в некотором смысле являются параметрами такой функции приспособленности (в англоязычной литературе называемой *Fitness*), поскольку она показывает «успешность» особи или ее близость к идеальной эталонной – т. е. той, которая и является искомым решением.

Для сопоставления особи хромосомой применяются следующие основные операции:

– генерация первоначальной популяции особей (состоящих из случайных хромосом);

– скрещивание (получение новой особи из хромосом ее родителей);

– мутация (случайное изменение ген особи);

– селекция (отбор в новую популяцию наилучших особей, т. е. имеющих максимальную приспособленность).

Естественно, в ГА присутствуют определенные вариации, такие, как получение нескольких особей при скрещивании или отбор не всегда наилучших особей. Выбор частоты скрещивания и мутации имеет принципиальное значение для скорости схождения алгоритмов [7].

Для понимания ГДЭ, опишем ее в терминах ГА, используя примеры, характерные для ГДК. Сама оптимизационная задача заключается в поиске экземпляра программы в предыдущем представлении, который бы при сборке ПО (например, компиляции) преобразовывался в некоторую эталонную программу в текущем представлении; таким образом, можно будет говорить о деэволюции этой программы в более высокоуровневое представление (например, путем получения искомого ИК из эталонного МК). Сама функция приспособленности в этом случае оценивает близость программы, полученной при сборке из текущего представления, с эталонной (например, некоторый ИК компилируется в МК и сравнивается с эталонным МК, близость к которому и определяет его приспособленность а, следовательно, и шансы на «выживание» в новой популяции). Под особями же, таким образом, понимаются экземпляры программ в требуемом представлении, а их хромосомы и гены определяют способ ее записи (например, через токены языка программирования или более сложные конструкции). В рамках ГДЭ для соответствия генов особи конкретному экземпляру программы используется граф синтаксических правил (далее – ГСП), задающий в формальном виде синтаксис ЯП или нотацию для представления, каждый ген в котором задает один из выборов возможного передвижения по графу (например, хромосома ИК с текстом «*x* – *z*» для синтаксиса с идентификаторами «*x*, *y*, *z*» и математическими операциями «+, –, *, /» может состоять из генов «1, 2, 3», которые последовательно задают выбор 1-го идентификатора «*x*», 2-й операции «–» и 3-го идентификатора «*z*») – таким образом, программа задается с помощью пути по ГСП. Тривиальный и интуитивно понятный пример записи формального синтаксиса для приравнивания двух переменных в форме Бэкуса – Наура (далее – ФБН) [8] приведен в листинге 1, где префиксами до «::=» указаны номера правил синтаксиса, а символом «…» – незаконченность строки в виде аналогичного продолжения идентификаторов.

Листинг 1. Пример синтаксиса приравнивания двух переменных (в форме Бэкуса – Наура)

Listing 1. Example of Syntax for Equating Two Variables (in Backus – Naur Form)

```
1: assign ::= identifier operation identifier ;
2: identifier ::= ‘a’ | ‘b’ | ‘c’ | ... ;
```

Согласно синтаксису в Листинге 1, первой строкой задается правило приравнивания «assign» (в левой части), соответствующее (в правой части) двум идентификаторам «identifier» с операцией «operation» между ними. Во второй строке задаются возможные значения для идентификаторов

(символы «*a*», «*b*», «*c*» и т. д.) – т. е. альтернативные раскрытия правила. Представление данного синтаксиса в графической форме ГСП (в интересах компактности, для трех идентификаторов) представлено на рисунке 2.

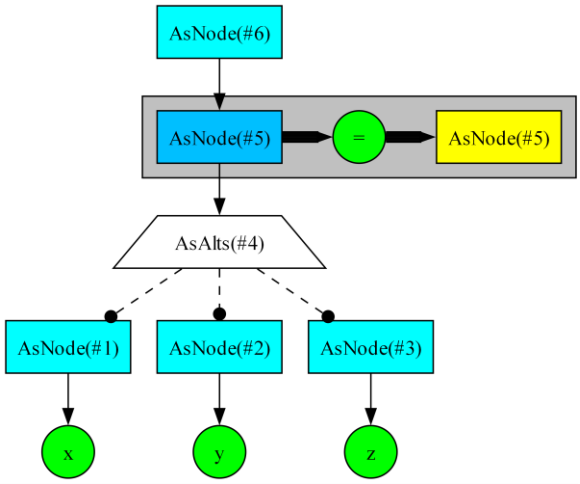

Рис. 2. Пример синтаксиса приравнивания двух переменных (в форме графа синтаксических правил)

Fig. 2. Example of Syntax for Equating Two Variables (in the Form of a Syntax Rules Graph)

Необходимо отметить, что хотя синтаксическое правило «assign» задается через 2 идентификатора, каждый из которых раскрывается тремя символами (см. листинг 1), в ГСП (см. рисунок 2) символы раскрываются только для первого идентификатора (узел «AsNode(#5)» синего цвета) с нисходящими ветками из дочернего узла «AsAlts(#4)»), а второй идентификатор представлен только одним узлом (узел «AsNode(#5)» желтого цвета), поскольку идентичен такому же узлу на 2 узла левее; такая запись существенно улучшает восприятие сложных графов, делая его визуально деревом (т. е. без циклов). Также несколько путей из узла «AsAlts(#4)» соответствуют тому, что из него продвижение по графу может пойти по нескольким путям – т. е. альтернативам.

Поскольку все передвижения по правилам в ФБН и ГСП, кроме достижения родительских узлов с альтернативами, идут по безусловному пути (например, после идентификатора «AsNode(#5)» обязательно следует символ «=» и еще один идентификатор «AsNode(#5)»), то основной вариативностью в генерации или «парсинге» ИК может быть один из выборов этих альтернатив; таким образом, именно порядковый номер альтернативы и был выбран в качестве гена соответствующей хромосомы экземпляра программы.

Важной особенностью ГДЭ, отличной от большинства решений на базе ГА, является переменность размера хромосомы, поскольку сама программа может состоять из различного количества конструкций (например, ИК может быть как тривиальным «$x = y$», так и комплексным математическим выражением с вызовом функций «$x = y * (z - \text{funct}(x, y, z))$»). При этом операции селекции и мутации в ГДЭ имеют достаточно сложную логику и состоят из качественно различных изменений формы и содержания программы (например, в ИК может как мутировать отдельный идентификатор или операция без изменения длины хромосомы, так и производиться принципиальная «перестройка» крупных конструкций условных переходов). Соответственно, задача ГДЭ считается решенной, когда получена программа, которая после преобразования в заданное представление тождественна эталонной (например, если был получен ИК, компилируемый в МК, побайтно совпадающий с эталонным, то, следовательно, этот ИК и является результатом декомпиляции). Такая идентичность результата компиляции ИК к эталонному МК заключается не только в гарантированности корректно проведенной ГДЭ, но и в том, что повторная сборка программы с устраненной в ИК уязвимостью позволит получить соответствующий МК, отличный от исследуемого лишь отсутствием этой уязвимости, не затрагивая иной функционал.

Сигнатурный поиск уязвимостей

Несмотря на то, что целевым предназначением ГРИ является отображение уязвимостей в каждом из представлений, дополнительные исследования показали [9], что поиск уязвимостей может производиться непосредственно на структурах ГДЭ путем сигнатурного анализа. Так, поскольку уязвимость в некотором представлении задается теми же конструкциями, что и основная программа, то ее сигнатура может быть задана, как часть пути по ГСП (например, деление на 0 в ГСП отражается, как последовательный переход на операцию деления, вторым операндом которой является константа с нулевым значением). А поскольку полная хромосома программы задается путем по ГСП, то ее часть (как подпоследовательность ген) может быть отождествлена с сигнатурой уязвимости.

Проблемные вопросы

Приведем далее основные проблемные вопросы (с префиксом «ПВ»), выявленные в результате исследования и развития ГРИ, а также опишем рекомендации (т. е. фактически обоснованные пути разрешения) для их устранения; для большей понятности часть вопросов будет касаться только ГДК, хотя путем обобщения они могут быть отнесены и к любым представлениям в процессе ГРИ.

ПВ_1. Формирование входного синтаксиса

Поскольку одним из входных параметром в ГДЭ является синтаксис ЯП, подходящий для обработки алгоритмами ГА, то возникает проблемный вопрос его формирования. Это может быть достигнуто

парсингом и преобразованием формальной записи синтаксиса, напрямую используемого в компиляторе (который, собственно, и применяется для получения из ИК соответствующего МК, сравниваемого с эталонным). В результате, синтаксис может быть переведен во внутренние структуры конкретной реализации ГДЭ и использоваться для построения и обхода ГСП.

ПВ_2. Восстановление неоптимального ИК

Исходя из того, что один и тот же ИК может соответствовать нескольким МК, его восстановленные вариации гипотетически имеют различную оптимальность (как по производительности, так и по лаконичности). Например, три таких ИК, как «$x = y$», «$x = y + 0$» и «$x = 2 * y - y$», идентичны с позиции результатов своей работы и теоретически могут после компиляции привести к одному МК. Впрочем, во-первых, без применения опций оптимизации 2-й и 3-й ИК скорее всего не будут упрощены компилятором и приведут к излишним инструкциям сложения и умножения, что сделает их существенно отличными от эталонного МК. А, во-вторых, добавление и учет метаинформации в синтаксисе ЯП может снизить появление излишних конструкций в ИК (например, невозможность получения после скрещивания или мутации выражений со сложением, где одним из аргументов является 0).

ПВ_3. Восстановление слабо интерпретируемого ИК

Крайне интересным с научной и практической точки зрения является абсолютно корректное получение ИК, который, хотя в точности и соответствует эталонному МК, но его понимание Экспертом является сложной задачей из-за эволюционного получения такой особи, логика работы которой (заданная ее хромосомой, отраженной в элементах ИК) строится на иных принципах, нежели человеческая логика. Данная проблема характерна для искусственных нейронных сетей, которые также имеют слабую интерпретируемость [10]. С этой позиции, одним из положительно-побочных эффектов области ГДЭ может стать развитие области искусственно-интеллектуальной логики, качественно отличной от человеческой, которая также получена эволюционным способом, но в синтетической среде (как совокупности «выживания» особей ИК, стремящихся к приспособленности в виде близости к эталонному МК, как некоторому «условному прообразу цифрового идеала»). Повышения же интерпретируемости можно добиться как модификацией ГСП, так и дополнительной алгоритмической обработкой ИК (в том числе, исходя из корректировок Эксперта).

ПВ_4. Попадание в локальный максимум

Одним из проблемных вопросов любого ГА считается возможность попадания в так называемый локальный максимум (или минимум) [11], суть которой заключается в нахождении псевдооптимального решения. В контексте ГДЭ это означает, что будет сформировано поколение подобных друг другу экземпляров ИК, которые компилируются в МК, близкий, но не тождественный эталонному. При этом операции скрещивания и мутации не позволяют выйти из данной ситуации, поскольку стенки такой локальной «ямы» слишком велики (т. е. удаленность функции приспособленности для ИК, более близкого к истинному решению, окажется критичной для ГА). Тем не менее для решения данного вопроса существует ряд техник, таких как оптимизация параметров ГА (например, увеличение «силы» мутации) или применение искусственного интеллекта для контроля качества популяции, чтобы избежать «застревания» эволюции).

ПВ_5. Рост количества итераций ГДЭ от размера ГСП

Размер ГСП (т. е. количество узлов, связей и т. п.) может критически сказаться на количестве итераций эволюции, что негативно повлияет и на оперативность работы ГДЭ. В итоге время решения оптимизационной задачи восстановления программы может оказаться выше предельно допустимого. Для качественного ускорения эволюции путем анализа МК имеет смысл использовать не весь синтаксис ЯП (на альтернативах которого определены гены особей), а выбрать его минимально необходимое подмножество – подграф синтаксических правил которого окажется существенно меньше основного графа. Например, если в МК нет инструкций условного перехода (соответствующих, например, конструкциям «if–else» в ЯП С), то нет смысла использовать соответствующие правила и при выполнении операций скрещивания и мутации.

ПВ_6. Рост количества итераций ГДЭ от вложенности синтаксиса

Наличие в синтаксисе ЯП конструкций, описывающих сложные вложенные выражения, будет иметь такое же негативное влияние на длительность эволюции, как и размер ГСП. Так, например, ИК вложенного сложения «$x + (x + (x + (x + x)))$» может быть записан с помощью синтаксических правил так, как показано в листинге 2.

Листинг 2. Пример синтаксиса выражения с вложенным сложением (в форме Бэкуса – Наура)

Listing 2. Example of Syntax for Expression with Nested Addition (in Backus – Naur Form)

```
1: expression ::= identifier '+' identifier_or_expression ;
2: identifier_or_expression ::= 'x' | '(' expression ')' ;
```

ГСП для такого синтаксиса приведен на рисунке 3; во второй строке узлов указаны ссылки на соответствующие правила. Вложенность синтаксиса, соответствующая рекурсии его правил, в данном случае заключается в том, что «expression» (или узел «AsNode(#4)») выражается через правило «identifier_or_expression» (узел «AsNode(#2)»), вто-

рая альтернатива которого (узел «AsNode(#6)», дочерний к «AsAlts(#1)») содержит это же правило «expression». Соответственно, алгоритмы ГДЭ гипотетически могут тратить значительное количество эволюционных итераций на обход данной рекурсии в глубину даже для выражений без больших вложенностей. Решением этого проблемного вопроса может стать упрощение синтаксиса (и, соответственно, ГСП) с помощью ограничения рекурсивности путем ее «распаковки» в аналогичные последовательности простых выражений. Например, одна строка ИК с комплексным выражением «$w = x + (y + z)$» может быть записана, как две строки с приравниванием промежуточных значений к временны́м переменным: «$t1 = y + z$; $w = x + t1$». Значит, для записи второго варианта ИК рекурсия для указания вложенных выражений не требуется. Естественно, полностью от рекурсий отказаться не удастся, поскольку любой ИК программы представляет собой теоретически бесконечную последовательность выражений (которые зачастую соответствуют одной строке ИК); однако эволюционный подбор такой циклической по ГСП конструкции является более решаемой задачей для ГДЭ, поскольку каждое вложенное выражение ИК на этапе компиляции, как правило, разворачивается в последовательность простых, оперирующих промежуточными значениями и временны́ми переменными.

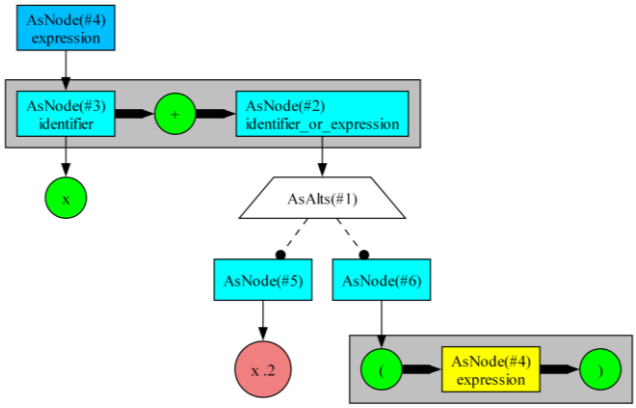

Рис. 3. Пример синтаксиса выражения с вложенным сложением (в форме графа синтаксических правил)

Fig. 3. Example of Syntax for Expression with Nested Addition (in the Form of a Syntax Rules Graph)

ПВ_7. Сложность подбора идентификаторов

Сложность подбора идентификаторов в ИК заключается в том, что они уже отсутствуют в МК и, соответственно, не могут быть из него восстановлены. Также идентификаторы, как правило, отсутствуют и в синтаксисе ЯП, поскольку процесс разбора имени переменной типовым компилятором происходит на ранней фазе парсинга кода – лексическим анализатором [12], который передает последующему синтаксическому анализатору лишь тип токена – идентификатор. Впрочем, само название не имеет принципиальной важности как для ГДК, так и для дальнейшего анализа ИК Экспертом. Также, поскольку необходимым для решения текущей оптимизационной задачи является получение ИК, в точности компилируемого в эталонный МК, то важно восстановить именно зависимость данных межу переменными, а не сами их имена. Таким образом, в синтаксис возможно введение некоторого разумного пула альтернативных имен переменных, среди которых ГА и будет выбирать необходимую.

Так, например, для небольшого размера ассемблерного кода (далее – АК) достаточным может оказаться наличие в ИК всего пяти переменных, задаваемых следующим синтаксическим правилом:

identifier ::= ‘var1’ | ‘var2’ | ‘var3’ | ‘var4’ | ‘var5’.

Также, используя синтаксис АК, в котором четко указано место имени идентификатора, возможно предварительно провести разбор его текста, определить используемые идентификаторы, модифицировать синтаксис ИК и, уже используя его, провести ГДЭ.

ПВ_8. Сложность подбора числовых константных значений

Одним из сложнейших проблемных вопросов в ГДЭ по формальному синтаксису ЯП является подбор числовых константных значений, которые для 32-х битных систем могут принимать огромные значения – от 0 до 4294967295. Естественно, даже с учетом мутаций подбор таких значений займет недопустимо большое время (а наличие дробных чисел в программе, зачастую представляемых посредством целых в специальном формате, еще существеннее усложнит данную задачу). Разрешение данной ситуации может лежать в плоскости следующих двух способов.

Первый способ связан с тем, что учет частоты используемых константных значений в типовых программах позволит изначально выбирать те, которые с наибольшей вероятностью являются верными; данная статистика уже была собрана ранее и определила Топ-10 таких констант – 0, 1, 2, 3, 4, 10, 5, 8, 16, 7 [13]. Естественно, «угадать» абсолютно любую константу вряд ли окажется возможным, что несколько ограничивает область применения ГДК.

Вторым способом является анализ АК или МК на предмет присутствия в нем константных значений, которые логичным образом должны присутствовать и в ИК (по аналогии с разрешением ПВ_7); например, если в АК присутствуют выражения, оперирующие числами 3, 7 и 11, то и в ИК константами будут именно они. Таким образом, альтернативы в используемом синтаксисе ЯП (и, соответственно, ГСП) могут состоять лишь из ведущих к узлам с этими числами: number ::= ‘3’ | ‘7’ | ‘11’.

ПВ_9. Сложность подбора строковых константных значений

Еще более сложной задачей, чем подбор идентификаторов и числовых константных значений, является восстановление текстовых строк, используемых в программе, поскольку они могут состоять из огромного количества символов, подбор которых займет практически неограниченное время. Частично данный проблемный вопрос может быть разрешен аналогичным для подбора числовых констант образом. Например, в Топ-50 наиболее часто используемых строк входят следующие: «%d», «%d\n», «\n» и «"»». При этом анализ бинарной формы программы на предмет наличия в ней строк (хранящихся, допустим, в секции «.rodata» для выполняемого файла формата ELF [14]), а также соответствующая модификация ГСП, позволит качественно ускорить деэволюцию МК.

ПВ_10. Сложность восстановления группы функций вместо ИК одной

Несмотря на то, что теоретические исследования и практические эксперименты показали возможность ГДЭ отдельных участков МК, однако его полноценное применение для целой программы, состоящей из набора функций (хотя более точно говорить о подпрограммах), требует дополнительного исследования по данной ветке. Причина этого заключается в том, что отображение функций в синтаксисе ЯП, а также в МК и АК программы, является более сложным, чем подобное отображение конструкций внутри нее. Так, помимо разделения программы на функции, каждая из них имеет аргументы с разными способами передачи (через регистры, стек или глобальную память) и опционально возвращаемые значения, может вызывать другие функции, в том числе по указателю (ссылке) и т. п. Базовым решением вопроса целесообразно рассмотреть применение соответствующих методов детектирования как самих участков функций, так и информации об их входных и выходных параметрах; что, впрочем, снизит общую инвариантность ГДЭ.

Другим решением может стать введение нового (возможно, промежуточного) представления между ИК и МК, которое бы представляло собой некоторый граф вызовов функций и их декларации. Тогда сам ГДЭ вначале будет восстанавливать программу в этом представлении, оставляя без изменений МК внутри функций, а затем уже производить деэволюцию каждой отдельной функции.

ПВ_11. Генерация первоначальной популяции

Первоначальная популяция, как и ее размер, хотя и не имеет решающее значение, однако существенно влияет на скорость проведения деэволюции программы, поскольку, чем ближе ее особи окажутся к искомой эталонной, тем быстрее сойдется ГА. В этой ситуации случайная генерация особей не позволит соптимизировать ГДЭ. Возможным решением может быть распознавание шаблона программы в текущем представлении (например, с применением машинного обучения) и генерация первоначальной популяции в соответствии с ним. Также с помощью существующих методов возможно определение ее оптимального размера [15].

ПВ_12. Выбор первоначального размера хромосомы

Размер хромосомы можно считать таким же важным оптимизационным фактором, как и гены представителей первоначальной популяции, поскольку чем сильнее отличается количество ее ген от того, которое позволяет получить эталонную особь (т. е. приводит к завершению ГДЭ), тем более долгий путь придется пройти эволюции для «увеличения» или «уменьшения» длины хромосомы. Так, например, если эталонный МК получен из ИК, который задается хромосомой из 10 ген, а в первоначальной популяции были особи с 1 и 20 генами, то количество итераций ГДЭ в среднем будет больше, чем если бы хромосомы такой популяции содержали бы 9 или 11 (а в идеале, 10) ген. Для предсказания же их количества возможно использовать зависимость (очевидно, существующую) между ИК и получаемым из него МК; что уже было получено ранее [16].

ПВ_13. Выбор частот скрещивания и мутации

Выбор параметров ГА, таких, как частота скрещивания пары особей или мутации одной из них (как, впрочем, и применение той или иной разновидности этих операций) очевидно оказывает влияние на получение более «удачного» поколения и, следовательно, общее время работы ГДЭ. Как правило, такие параметры выбираются эмпирически и, гипотетически, будут иметь схожие значения для типовых синтаксисов или шаблонов программ. Следовательно, опыт предыдущих восстановлений ИК из группы может быть применен и для ГДЭ других подобных ИК. Впрочем, это требует отдельного теоретического исследования и практических экспериментов для разнородных входных данных.

ПВ_14. Влияние качества функции приспособленности на сходимость

Одним из решающих факторов успешного эволюционирования особей в ГДЭ является качество функции приспособленности, которая определяет близость программ, полученных преобразованием (в случае ИК – компиляцией) экземпляров популяции предыдущего представления к эталонной особи текущего представления. При этом точные критерии близости сложно определимы, а подобная задача принципиально не рассматривалась другими учеными. Однако для проверки Концепции, сравнение двух программ в представлении АК осуществлялось с помощью авторской метрики,

принимающей на вход список из двух строк, состоящих из последовательности символов [17]; метрика также способна учитывать отдаленность строк и символов от начала содержащих их множеств, что отражает соответствие последовательности логики функционирования в ИК и АК. Данная метрика с достаточной чувствительностью позволяет оценивать, как близость одной текстовой строки к другой, так и одного их упорядоченного списка к другому. Также, качественным развитием механизма является непосредственный учет в сравнении синтаксиса текущего представления, которым в случае АК может являться TASM, NASM, MASM, FASM, YASM, ASM-51 и др.

Так, если имеются две следующие строки АК:

```
mov eax, DWORD PTR _x$[ebp]  # занесение в
регистр EAX значения переменной _x
```

и

```
add edx, DWORD PTR _x$[ebp]  # добавление к
регистру EDX значения переменной _x
```

то посимвольное сравнение будет не совсем точным, поскольку более существенное отличие этих строк в том, что используются принципиально различные процессорные инструкции – приравнивание (MOV) и добавление (ADD). Такую разницу возможно определить путем учета синтаксиса АК, в котором строчки будут определяться различными подграфами соответствующего ГСП.

ПВ_15. Время-затратность вычисления функции приспособленности

Практически в любой реализации Концепции наиболее длительной по времени операцией является вычисление функции приспособленности [18], поскольку она требует наличие МК, соответствующего ИК, согласно генам особи. Для этого производится ресурсозатратный (не только по времени, но и по аппаратным ресурсам рабочей станции) вызов компилятора программы. Данная ситуация отличается от классической, в которой зачастую эта функция производит ряд несложных аналитических действий, хорошо распараллеливаемых. Впрочем, существенного ускорения компиляции можно достигнуть тремя способами.

Во-первых, ведение базы уже скомпилированных экземпляров ИК (например, в форме словаря, где ключом является последовательность ген ИК, а значением – приспособленность его МК) предотвратит повторное вычисление функции.

Во-вторых, компиляция группы ИК (так называемый, пакетный режим) будет несколько быстрее, поскольку ряд операций (запуск процесса, инициализация компилятора и т. п.) вызовутся 1 раз для всей группы. При этом компиляцию можно вызывать для всей популяции, т. к. приспособленность их особей требуется только перед операцией селекции – т. е. один раз за эпоху эволюции.

И, в-третьих, наличие открытого ИК утилиты компиляции позволит внедрить ее алгоритмы напрямую в ГДЭ, что еще более снизит накладные расходы.

Первые 2 способа были реализованы на практике в авторском Прототипе и показали свою эффективность.

ПВ_16. Идентификация средства преобразования особи

При проведении ГРИ программы, информация о которой отсутствует (например, уничтожена злоумышленником или иными деструктивными воздействиями), определение средства ее преобразования из предыдущего представления в текущее (для сравнения особей с эталонной) является важным проблемным вопросом. В ином случае, например, когда компилятор ИК или его опции были выбраны некорректно, ГДЭ может осуществляться критически высокое время (а в ряде случаев, и не завершиться вовсе). Однако существуют исследовательские разработки, которые позволяют идентифицировать такие средства по метаинформации в программах [19].

ПВ_17. Идентификация синтаксиса представления

Аналогично идентификации средства преобразования особей, требуется определение синтаксиса текущего представления для более корректного выбора или настройки функции приспособленности. Так, например, в случае восстановления ИК из МК необходимо понимание процессорной архитектуры. Однако современные наработки с достаточно большой эффективностью позволяют создавать «цифровые портреты» (на основании специфики распределения байтов МК) для каждой такой архитектуры, обеспечивая, тем самым, ее идентификацию (что было продемонстрировано в авторских [20, 21] и иных [22] научно-практических исследованиях).

ПВ_18. Использование АК вместо МК

Достаточно большая часть как теоретических исследований, так и практических реализаций ГДЭ проведена в интересах восстановления ИК по его АК, поскольку использование вместо этого в качестве эталонного классического МК является одним из проблемных вопросов. Хотя получение АК из МК и осуществляется в полной мере так называемыми утилитами дизассемблирования [23], получаемый ими АК все же будет отличен от такого же, сгенерированного при непосредственной компиляции ИК. Так, например, ИК функции сложения двух аргументов, представленный в листинге 3 в процессе компиляции преобразуется в АК в листинге 4, хотя дизассемблирование в продукте IDA Pro полученного МК дает АК, представленный в листинге 5.

Листинг 3. Исходный код функции сложения двух аргументов

Listing 3. Source Code for the Function Adding Two Arguments

```
int f(int x, int y) {
  return x + y;
}
```

Листинг 4. Ассемблерный код функции сложения двух аргументов (после компиляции исходного кода)

Listing 4. Assembly Code of the Two Arguments Addition Function (After Compilation the Source Code)

```
_x$ = 8
_y$ = 12
_f  PROC
; Line 1
  push ebp
  mov ebp, esp
; Line 2
  mov eax, DWORD PTR _x$[ebp]
  add eax, DWORD PTR _y$[ebp]
; Line 3
  pop ebp
  ret 0
_f  ENDP
```

Листинг 5. Ассемблерный код функции сложения двух аргументов (после дизассемблирования машинного кода)

Listing 5. Assembly Code of the Two Arguments Addition Function (After Disassembling the Machine Code)

```
_f proc near
arg_0 = dword ptr 8
arg_4 = dword ptr 0Ch

push ebp
mov ebp, esp

mov eax, [ebp + arg_0]
add eax, [ebp + arg_4]

pop ebp
retn

_f  endp
_text$mn ends
end
```

Впрочем, отличия таких АК (см. листинги 4 и 5) носят несущественный характер – формат строк с инструкциями практически идентичен за исключением метаинформации, создаваемой утилитами (пометки о границах функций, аргументах и т. п.), доступа к значению переменных (DWORD PTR вместо конструкции «[…]») и самих имен переменных (исходные «*_x*» и «*_y*» вместо автоматически сгенерированных «arg_0» и «arg_4»). Соответственно, поскольку оба АК в точности соответствуют одному МК, то и инструкции в их строках идентичны по содержанию, а отличия в формах могут быть скорректированы достаточно тривиальным образом. Потеря же имен переменных при получении АК из МК не будет критичной, поскольку конкретные их идентификаторы не имеют существенного значения для успешности ГДЭ.

ПВ_19. Отсутствие формальных синтаксисов ряда представлений

Следуя масштабности описанной Концепции, она гипотетически предназначена для получения из МК не только ИК или алгоритмов, но и архитектуры программы, ее концептуальной модели или даже самой идеи – например, в форме текстового описания сущности целой программы, ее отдельных функций и назначения. Тем не менее, наиболее распространенными и проработанными синтаксисами представлений на сегодняшний день остаются 3 следующих – МК, АК и ИК; хотя формализация алгоритмов (как более абстрактного и человекоориентированного представления ИК) в форме блок-схем или псевдокода также применяется. Тем не менее, судя по общему усложнению области программной инженерии, а также внедрению в этот процесс искусственного интеллекта, можно спрогнозировать появление формализации других представлений программы (за счет их формального синтаксиса или иных моделей), что повлечет за собой и возможность проведения соответствующей ГДЭ. С этой позиции, ГРИ представляет собой перспективное направление поиска уязвимостей программ в тех представлениях, где они были заложены.

ПВ_20. Проверка Концепции ГДЭ на ограниченном количестве представлений

Исходя из того, что формальные представления и средства преобразования на сегодняшний день наиболее развиты лишь для двух (точнее трех) представлений – ИК и МК, преобразуемого в АК, проверка Концепции на остальных носит больше теоретический характер. Однако качественная подобность всех представлений и создание алгоритмов ГДЭ, как независимых (в пределе) от специфики программ, позволяют полагаться на работоспособность Концепции и в более широких границах применения. Так, например, если будет создан полноценный синтаксис алгоритмов программы (как в форме блок-схем, так и псевдокода), позволяющий по ним генерировать ИК за счет подбора альтернатив, то ГДЭ между этими двумя представлениями будет возможна также, как и ГДК. В качестве близкого примера можно привести графический язык программирования логических контроллеров (FBD, *аббр. от англ.* Function Block Diagram) [24], программы на котором как раз и преобразуются в псевдо ИК или АК. Проведение ГДЭ таких графических блок-схем можно рассматривать в качестве одной из ближайших целей обоснования Концепции для ГРИ МК в более высокоуровневые формы.

Также стоит отметить, что на первый взгляд узкая задача по восстановлению ИК из МК или АК является крайне принципиальной с точки зрения ИБ, поскольку экспертный (а в ряде случае, и автоматический) анализ этих представлений качественно отличен с позиции эффективности – бинарная форма практически не подходит для ручного поиска уязвимостей, а соответствующие методики анализа по ИК существуют и давно используются. При этом в области программной инженерии основная работа заключается именно в создании ИК

и его преобразовании в МК, который уже непосредственно выполняется на устройствах.

ПВ_21. Зависимость сигнатур уязвимостей от синтаксиса

Существенным ограничением широкого применения предложенной сигнатурной записи уязвимостей является их зависимость от конкретного синтаксиса представления – т. к. сигнатура описывается начальным узлом и выбором альтернатив по заданному ГСП. Однако даже такую запись можно считать более расширенной, чем классическая, когда сигнатура задает строгую последовательность байт или инструкций в МК для конкретного процессора выполнения. Так, например, типовое выявление вредоносного кода в МК, собранного под N процессорных архитектур, потребовало бы наличия такого же количества сигнатур. Использование же ГДЭ позволяет восстановленный из них ИК записать с помощью одной хромосомы (соответствующей пути на выбранном ГСП), что потребует наличия лишь одной сигнатуры уязвимости. Аналогичная ситуация будет и в случае создания уязвимости на одном из N языков программирования. Таким образом, гипотетически можно предположить востребованность в создании единого (или их ограниченной группы) синтаксиса, адаптированного для отображения на нем сигнатур уязвимостей и деэволюции в них МК программы для большого количества процессорных архитектур.

ПВ_22. Отсутствие учета семантики и динамики при поиске уязвимостей

Очевидно, что сигнатура уязвимостей задает исключительно ее синтаксические особенности (например, последовательностью конструкций ЯП) и не позволяет выявлять более сложные случаи. Так, два следующих ИК потенциально приводят к делению на 0:

1) «$y = x / 0$»;
2) «if ($z == 0$) { $y = x / z$; }».

Первый ИК выявляется синтаксическим анализом (по факту наличия константы «0» после оператора деления), а второй – семантическим или динамическим (условие будет выполняться только при нулевом значении «z», что и приведет к делению на 0). Впрочем, сигнатуры по ГСП изначально не были предназначены для выявления более сложных «логических» уязвимостей, что следует из ограничений сигнатурного анализа (по крайне мере, его статической формы). Тем не менее, гипотетически можно развить ГДЭ и для восстановления семантики кода, например, путем введения графа семантических правил, сигнатур на нем и необходимого набора алгоритмов.

ПВ_23. Неполный охват обнаруживаемых уязвимостей

Следуя отсутствию «глубоких» алгоритмов ГДЭ анализа уязвимостей, определяемых внутренней логикой функционирования программы (а не ее внешней синтаксической формой), необходимым условием выявления можно считать их локализацию в коде. Так, например, если часть одной уязвимости расположена в начале тела функции, часть – в середине, а часть – в конце, то ее запись через сигнатуру (как последовательность узлов ГСП) вряд ли будет возможна. Однако частичным разрешением данного проблемного вопроса может быть применение более сложных форм сигнатур, построенных, например, как шаблоны, определяющие лишь необходимые части синтаксиса. Впрочем, без учета семантики или динамики выполнения кода (следуя ПВ_22), даже применение сигнатурных шаблонов будет достаточно ограниченным.

ПВ_24. Частичная инвариантность алгоритмов от синтаксисов представлений

Обеспечение полной инвариантности алгоритмов ГДЭ от синтаксических и иных особенностей программы во всех представлениях, хотя и является одной из теоретических особенностей Концепции, тем не менее имеет определенные трудности в практической реализации. Так, выбор размера хромосом особи требует предварительного определения зависимости между размерами программ в ближайших представлениях; предсказание константных значений для мутационного подбора – аналогичного составления статистики по частоте их использования или применения иных специализированных методов (например, как было указано ранее – выявление в АК констант, которые затем должны перебираться в ИК); адаптация существующих синтаксисов представлений – частичного учета соответствующих им семантических и иных правил; развитие поиска уязвимостей – отражение в их сигнатурах большего количества метаинформации о программе и т. д. Впрочем, повышение инвариантности может быть обеспечено оптимизационными и иными способами, которые имеют лишь технические сложности в реализации, однако не нарушают общую концепцию и не снижают ее обоснованность. Так, например, внедрение моделей и методов машинного обучения, которые после каждой компиляции ИК в МК будет обучаться «понимать», какая последовательность ген для каких узлов ГСП приводит к каким синтаксическим конструкциям в машинных инструкциях, гипотетически позволит существенно снизить зависимость алгоритмов ГДЭ от специфики программ их представлений.

ПВ_25. ГДЭ оптимизированной и обфусцированной программы

Применение при сборке программы различных техник оптимизации кода (например, по скорости работы или размеру образа МК) приводит к некоторому усложнению отображения его логики в получаемом представлении и, как результат, затруднению всего ГДЭ. Например, все функции могут быть «слиты» в одну для уменьшения накладных расходов на их вызов. Применение же техник обфускации, изначально предназначенных для запутывания кода, качественно усложнит проведение ГДЭ [25]. Впрочем, это является общей проблемой реверс-инжиниринга ПО. Однако большой авторский практический опыт позволяет утверждать, что достаточно часто программы, применяемые в критических областях и устройствах, собираются без использования таких техник, поскольку риск случайных ошибок в коде, возникающих от их применения, оказывается выше эффекта снижения размера, увеличения скорости или изменения иных характеристик программы. В качестве же гипотетического разрешения самого проблемного вопроса можно рассмотреть возможность доработки алгоритмов ГДЭ и настройку их параметров, а также соответствующую адаптацию ГСП, применимых именно для такого нетипового МК.

Систематизация проблемных вопросов

Проведем систематизацию всех приведенных проблемных вопросов с позиции основных путей их устранения (таблица 1), указывая для этого в качестве критериев часть ГРИ для доработки:
«К.» – концепция (например, введение новых этапов или изменение взаимосвязи существующих);
«А.» – алгоритмы (в том числе разработка новых);
«П.» – представления (включая синтаксис ЯП и ГСП);
«У.» – уязвимости (например, их сигнатуры).

Значение же уровня требуемой доработки (и их балльную оценку) обозначим следующим образом:
« » – отсутствует (0 баллов);
«+/–» – количественная или частичная (0,5 балла);
«+» – качественная или полная (1 балл).

Просуммируем эти значения для каждой части ГРИ (в последней строке таблицы), что позволит оценить наиболее «проблематичные» и требующие внимания из них. Так, например, указание для проблемного вопроса по критерию «А.» значения «+/–» говорит о том, что для его разрешения требуется подстройка параметров алгоритмов.

Анализ критериальной систематизации проблемных вопросов (см. таблицу 1) позволяет сделать следующие выводы. Во-первых, Концепция показала свою устойчивость или обоснованность (при минимальном суммарном количестве баллов – 1,0).

ТАБЛИЦА 1. Систематизация проблемных вопросов генетического реверс-инжиниринга программы с позиции путей их устранения

TABLE 1. The Problematic Issues Systematization of a Program Genetic Reverse Engineering from the Standpoint of Ways to Eliminate Them

Проблемный вопрос	Части ГРИ для изменения			
	К.	А.	П.	У.
ПВ_1. Формирование входного синтаксиса			+	
ПВ_2. Восстановление неоптимального ИК			+/–	
ПВ_3. Восстановление слабо интерпретируемого ИК			+/–	
ПВ_4. Попадание в локальный максимум		+/–		
ПВ_5. Рост количества итераций ГДЭ от размера ГСП			+/–	
ПВ_6. Рост количества итераций ГДЭ от вложенности синтаксиса			+/–	
ПВ_7. Сложность подбора идентификаторов		+	+/–	
ПВ_8. Сложность подбора числовых константных значений		+	+/–	
ПВ_9. Сложность подбора строковых константных значений		+	+/–	
ПВ_10. Сложность восстановления группы функций вместо ИК одной	+/–	+		
ПВ_11. Генерация первоначальной популяции		+/–		
ПВ_12. Выбор первоначального размера хромосомы		+		
ПВ_13. Выбор частот скрещивания и мутации		+/–		
ПВ_14. Влияние качества функции приспособленности на сходимость		+		
ПВ_15. Время-затратность вычисления функции приспособленности		+		
ПВ_16. Идентификация средства преобразования особи		+		
ПВ_17. Идентификация синтаксиса представления		+		
ПВ_18. Использование АК вместо МК		+	+/–	
ПВ_19. Отсутствие формальных синтаксисов ряда представлений			+	
ПВ_20. Проверка Концепции ГДЭ на ограниченном количестве представлений			+	
ПВ_21. Зависимость сигнатур уязвимостей от синтаксиса			+/–	+/–
ПВ_22. Отсутствие учета семантики и динамики при поиске уязвимостей		+/–	+	+
ПВ_23. Неполный охват обнаруживаемых уязвимостей		+/–		+
ПВ_24. Частичная инвариантность алгоритмов от синтаксисов представлений	+/–	+/–	+/–	+/–
ПВ_25. ГДЭ оптимизированной и обфусцированной программы		+/–	+/–	
Всего	1,0	13,5	9,5	3,0

Поскольку ни один из проблемных вопросов не требует ее принципиального пересмотрения – отсутствует необходимость в качественной ее доработке (т. е. отсутствует значение «+» в столбце

«К.»). Частичное же изменение концепции (следуя ПВ_10 и ПВ_24) на данный момент представляется решаемой задачей, требующей, однако, проведения дополнительных научно-практических изысканий.

Во-вторых, наиболее требуемыми доработки могут считаться алгоритмы (имеющие максимальное суммарное количество баллов – 13,5), что вполне закономерно, поскольку за их счет происходит расширение и / или улучшение имеющегося функционала до уровня, необходимого для проведения ГДЭ. На 2-м месте по рейтингу суммарного количества баллов (равного 9,5) расположена доработка представлений, что, хотя является и трудоемкой из-за некоторой монотонности, но вполне решаемой задачей (как вручную, так и применением автоматических средств трансформации).

В-третьих, доработка уязвимостей в части их сигнатур позволит разрешить соответствующую немногочисленную группу проблемных вопросов (с суммарным количеством баллов в 3,0). Наиболее же масштабным с позиции изменения частей ГРИ может считаться ПВ_24 (0,5 балла по каждому критерию), что объясняется возможностью повышения инвариантности каждой из четырех указанных частей ГРИ без каких-либо качественных изменений уже полученных авторских решений.

Заключение

Проведенное исследование является заключительной частью более крупного и основополагающего, посвященного методологии ГРИ программ в интересах поиска в них уязвимостей. Приводятся 25 основных проблемных вопросов, выявленных при переходе от концепции ГРИ (состоящей из отдельных ГДЭ) к ее непосредственной реализации в виде Прототипа для проведения ГДК. Для каждого такого вопроса указан путь его разрешения, что позволяет говорить об отсутствии «научно-практического тупика» и в основном исследовании.

Основным полученным научным результатом является систематизация выделенных проблемных вопросов с позиции необходимых доработок частей ГРИ с использованием балльных оценок. Новизна результата заключается в том, что большинство этих вопросов подняты впервые, что, впрочем, объясняет оригинальность самого ГРИ.

Теоретическая значимость результата состоит в том, что каждый из затронутых вопросов, фактически, является отправной (а, точнее, проблемно-постановочной) точкой для начала нового полноценного исследования, проведение которого позволит получить новые научно-практические результаты, выходящие за рамки авторского ГРИ. Так, например, разрешение ПВ_12 разовьет достаточно редко используемое направление ГА, оперирующее переменной длиной хромосом; а изучение ПВ_22 может привести к созданию метасигнатур уязвимостей, которые, с одной стороны, не зависят от ЯП и процессора выполнения программы, а с другой стороны, учитывают логику ее функционирования; исследование же ПВ_3 гипотетически откроет новые парадигмы программирования [26], на данный момент не адаптированные для восприятия человеком.

Практическая же значимость результатов заключается в том, что для каждого такого вопроса указано общее направление его разрешения, не только теоретически, но и путем создания соответствующего практического инструментария.

Продолжением работы может стать выбор и разрешение наиболее «интересных» проблемных вопросов, в том числе, другими исследователями.

Список источников

1. Гетьман А.И., Горюнов М.Н., Мацкевич А.Г., Рыболовлев Д.А. Сравнение системы обнаружения вторжений на основе машинного обучения с сигнатурными средствами защиты информации // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 111–126. DOI:10.15514/ISPRAS-2022-34(5)-7. EDN:LDJOUO

2. Израилов К.Е., Гололобов Н.В., Краскин Г.А. Метод анализа вредоносного программного обеспечения на базе Fuzzy Hash // Информатизация и связь. 2019. № 2. С. 36–44. EDN:DUIUJM

3. Израилов К.Е. Концепция генетической декомпиляции машинного кода телекоммуникационных устройств // Труды учебных заведений связи. 2021. Т. 7. № 4. С. 95–109. DOI:10.31854/1813-324X-2021-7-4-95-109. EDN:AIOFPM

4. 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

5. 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

6. Скобцов Ю.А. От генетических алгоритмов к метаэвристикам // Информатика и кибернетика. 2021. № 1-2(23-24). С. 101–107. EDN:ILCTUW

7. Тотухов К.Е., Романов А.Ю., Лукьянов В.И. Исследование эффективности работы генетических алгоритмов с различными методами скрещивания и отбора // Электронный сетевой политематический журнал "Научные труды КубГТУ". 2022. № 6. С. 98–109. EDN:DPRWIJ

8. Емельянов А.А. Рефлексивная распознающая грамматика // Вестник Волжской государственной академии водного транспорта. 2016. № 46. С. 23–32. EDN:VPBBLP

9. Буйневич М.В., Израилов К.Е. Сигнатурный поиск уязвимостей в машинном коде на базе генетической декомпиляции // Защита информации. Инсайд. 2025. № 2(122). С. 2–11. (в печати)

10. Микулик И.И., Уткин Л.В., Голубева И.Э. Разработка и исследование методов локальной интерпретации сиамской нейронной сети на основе объяснительного интеллекта // Математические методы в технике и технологиях – ММТТ. 2020. Т. 10. С. 88–91. EDN:SBFYYB

11. Силенко Д.И., Лебедев И.Г. Алгоритм глобальной оптимизации, использующий деревья решений для выявления локальных экстремумов // Проблемы информатики. 2023. № 2(59). С. 21–33. DOI:10.24412/2073-0667-2023-2-21-33. EDN:MLGKOX

12. Пырнова О.А., Никоноров Д.П., Шарифуллина А.Ю. Разработка статического анализатора программного кода // Научно-технический вестник Поволжья. 2023. № 12. С. 522–525. EDN:AVFOIE

13. Израилов К.Е. Исследование распределения константных значений в исходном коде программ на языке С // Труды учебных заведений связи. 2024. Т. 10. № 5. С. 119–129. DOI:10.31854/1813-324X-2024-10-5-118-128. EDN:KARAVM

14. Hu W., Chen T., Zhang N., Ma J. Adjust ELF Format for Multi-core Architecture // Proceedings of the International Conference on Electronic Computer Technology (Macau, China, 20–22 February 2009). IEEE, 2009. PP. 388–391. DOI:10.1109/ICECT.2009.73

15. Цыганков В.А., Шабалина О.А., Катаев А.В. Исследование воздействия размера популяции на быстродействие генетического алгоритма // Известия ЮФУ. Технические науки. 2024. № 3(239). С. 168–176. DOI:10.18522/2311-3103-2024-3-168-176. EDN:IAFWKU

16. Израилов К.Е. Прогнозирование размера исходного кода бинарной программы в интересах ее интеллектуального реверс-инжиниринга // Вопросы кибербезопасности. 2024. № 4(62). С. 13–25. DOI:10.21681/2311-3456-2024-4-13-25. EDN:NRFCND

17. Буйневич М.В., Израилов К.Е. Авторская метрика оценки близости программ: приложение для поиска уязвимостейс помощью генетической деэволюции // Программные продукты и системы. 2025. Т. 38. № 1. С. 197–206. DOI:10.15827/0236-235X.149.197-206

18. Пикалов М.В., Письмеров А.М. Настройка параметров генетического алгоритма при помощи анализа ландшафта функции приспособленности и машинного обучения // Известия ЮФУ. Технические науки. 2024. № 2(238). С. 221–228. DOI:10.18522/2311-3103-2024-2-221-228. EDN:EFIXDB

19. Pan Z., Yan Y., Yu L., Wang T. Identification of binary file compilation information // Proceedings of the IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (Chongqing, China, 16–18 December 2022). IEEE, 2022. PP. 1141–1150. DOI:10.1109/IMCEC55388.2022.10019958

20. Kotenko I., Izrailov K., Buinevich M. Analytical Modeling for Identification of the Machine Code Architecture of Cyberphysical Devices in Smart Homes // Sensors. 2022. Vol. 22. Iss. 3. P. 1017. DOI:10.3390/s22031017. EDN:WPXNDJ

21. Kotenko I., Izrailov K., Buinevich M. The Method and Software Tool for Identification of the Machine Code Architecture in Cyberphysical Devices // Journal of Sensor and Actuator Networks. 2023. Vol. 12. Iss. 1. PP. 11. DOI:10.3390/jsan12010011. EDN:POQUEB

22. Beckman B., Haile J. Binary Analysis with Architecture and Code Section Detection using Supervised Machine Learning // Proceedings of the IEEE Security and Privacy Workshops (San Francisco, USA, 21–21 May 2020). IEEE, 2020. PP. 152–156. DOI:10.1109/SPW50608.2020.00041

23. Гусенко М.Ю. Создание обобщенной нотации программного интерфейса процессоров x86 для автоматизированного построения дизассемблера // Программные системы и вычислительные методы. 2024. № 2. С. 119–146. DOI:10.7256/2454-0714.2024.2.70951. EDN:EJJSYT

24. Долидзе А.Н. Обзор специфических функций языка FBD на примере программируемых реле LOGO! // Инженерный вестник Дона. 2022. № 11(95). С. 1–10. EDN:UZGJVM

25. Ding S.H.H., Fung B.C.M., Charland P. Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization // Proceedings of the IEEE Symposium on Security and Privacy (San Francisco, USA, 19–23 May 2019). IEEE, 2019. PP. 472–489. DOI:10.1109/SP.2019.00003

26. Смольянинова М.О., Сидорова О.А. Об основных парадигмах современного программирования // Оригинальные исследования. 2023. Т. 13. № 7. С. 109–113. EDN:GJRBFF

References

1. Getman A.I., Goryunov M.N., Matskevich A.G., Rybolovlev D.A. A Comparison of a Machine Learning-Based Intrusion Detection System and Signature-Based Systems. *Proceedings of the Institute for System Programming of the RAS*. 2022;34(5): 111–126. (in Russ.) DOI:10.15514/ISPRAS-2022-34(5)-7. EDN:LDJOUO

2. Izrailov K.E., Gololobov N.V., Kraskin G.A. Method of Malware Analysis Based on Fuzzy Hash. *Informatization and communication*. 2019;2:36–44. (in Russ.) EDN:DUIUJM

3. Izrailov K. Genetic Decompilation Concept of the Telecommunication Devices Machine Code. *Proceedings of Telecommunication Universities*. 2021;7(4):95–109. (in Russ.) DOI:10.31854/1813-324X-2021-7-4-95-109. EDN:AIOFPM

4. 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

5. 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

6. Skobtsov Yu.A. From Genetic Algorithms to Metaheuristics. *Informatika i kibernetika*. 2021;1-2(23-24):101–107. EDN:ILCTUW

7. Totukhov K.E., Romanov A.Yu., Lukyanov V.I. Investigation of the Effectiveness of Genetic Algorithms with Various Methods of Crossing and Selection. *Scientific Works of the Kuban State Technological University*. 2022;6:98–109. (in Russ.) EDN:DPRWIJ

8. Emelyanov A.A. The Reflexive Recognizing Grammar. *Bulletin of VSAWT*. 2016;46:23–32. (in Russ.) EDN:VPBBLP
9. Izrailov K.E., Buinevich M.V. Signature Search for Vulnerabilities in Machine Code Based on Genetic Decompilation. *Zaŝita informacii. Inside*. 2025;2(122):2–11. (in Russ.)
10. Mikulik I.I., Utkin L.V., Golubeva I.E. Development and Research of Methods for Local Interpretation of the Siaman Neural Network Based on Explanatory Intelligence. *Mathematical methods in technics and technologies - MMTT*. 2020;10:88–91. (in Russ.) EDN:SBFYYB
11. Silenko D.I., Lebedev I.G. Global Optimization Algorithm That Uses Decision Trees To Find Local Extrema. *Problems of Informatics*. 2023;2(59):21–33. (in Russ.) DOI:10.24412/2073-0667-2023-2-21-33. EDN:MLGKOX
12. Pyrnova O.A., Nikonorov D.P., Sharifullina A.Yu. Development of a Static Program Code Analyzer. *Nauchno-tekhnicheskii vestnik Povolzhia*. 2023;12:522–525. (in Russ.) EDN:AVFOIE
13. Izrailov K.E. Constant Values Distribution Investigation in the C Programs Source Code. *Proceedings of Telecommunication Universities*. 2024;10(5):119–129. (in Russ.) DOI:10.31854/1813-324X-2024-10-5-118-128. EDN:KARAVM
14. Hu W., Chen T., Zhang N., Ma J. Adjust ELF Format for Multi-core Architecture. *Proceedings of the International Conference on Electronic Computer Technology, 20–22 February 2009, Macau, China*. IEEE; 2009. p.388–391. DOI:10.1109/ICECT.2009.73
15. Tsygankov V.A., Shabalina O.A., Kataev A.V. Investigation of the Impact of Population Size on the Performance of a Genetic Algorithm. *Izvestiya SFedU. Engineering Sciences*. 2024;3(239):168–176. (in Russ.) DOI:10.18522/2311-3103-2024-3-168-176. EDN:IAFWKU
16. Izrailov K.E. Predicting the Size of the Source Code of a Binary Program in the Interests of Its Intellectual Reverse Engineering. *Voprosy kiberbezopasnosti*. 2024;4(62):13–25. (in Russ.) DOI:10.21681/2311-3456-2024-4-13-25. EDN:NRFCND
17. Izrailov K.E., Buinevich M.V. Author's metric for assessing proximity of programs: application for vulnerability search using genetic de-evolution. *Software & Systems*. 2025;38(1):197–206. (in Russ.) DOI:10.15827/0236-235X.149.197-206
18. Pikalov M.V., Pismerov A.M. Genetic Algorithm Parameter Tuning Using Exploratory Landscape Analysis and Machine Learning. *Izvestiya SFedU. Engineering Sciences*. 2024;2(238):221–228. (in Russ.) DOI:10.18522/2311-3103-2024-2-221-228. EDN:EFIXDB
19. Pan Z., Yan Y., Yu L., Wang T. Identification of binary file compilation information. *Proceedings of the IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference, 16–18 December 2022, Chongqing, China.* IEEE; 2022. p.1141–1150. DOI:10.1109/IMCEC55388.2022.10019958
20. Kotenko I., Izrailov K., Buinevich M. Analytical Modeling for Identification of the Machine Code Architecture of Cyber-physical Devices in Smart Homes. *Sensors*. 2022;22(3):1017. DOI:10.3390/s22031017. EDN:WPXNDJ
21. Kotenko I., Izrailov K., Buinevich M. The Method and Software Tool for Identification of the Machine Code Architecture in Cyberphysical Devices. *Journal of Sensor and Actuator Networks*. 2023;12(1):11. DOI:10.3390/jsan12010011. EDN:POQUEB
22. Beckman B., Haile J. Binary Analysis with Architecture and Code Section Detection using Supervised Machine Learning. *Proceedings of the IEEE Security and Privacy Workshops, 21–21 May 2020, San Francisco, USA*. IEEE; 2020. PP. 152–156. DOI:10.1109/SPW50608.2020.00041
23. Gusenko M.Yu. Creating a Common Notation of the X86 Processor Software Interface for Automated Disassembler Construction. *Software systems and computational methods*. 2024;2:119–146. (in Russ.) DOI:10.7256/2454-0714.2024.2.70951. EDN:EJJSYT
24. Dolidze A.N. Overview of the Specific Functions of the FBD Language on the Example of Programmable Relays Logo! *Engineering journal of Don*. 2022;11(95):1–10. (in Russ.) EDN:UZGJVM
25. Ding S.H.H., Fung B.C.M., Charland P. Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization. *Proceedings of the IEEE Symposium on Security and Privacy, 19–23 May 2019, San Francisco, USA*. IEEE; 2019. p.472–489. DOI:10.1109/SP.2019.00003
26. Smolyaninova M.O., Sidorova O.A. About the Main Paradigms of Modern Programming. *Originalnye issledovaniia*. 2023;13(7):109–113. (in Russ.) EDN:GJRBFF

Статья поступила в редакцию 20.12.2024; одобрена после рецензирования 22.01.2025; принята к публикации 05.02.2025.
The article was submitted 20.12.2024; approved after reviewing 22.01.2025; accepted for publication 05.02.2025.

Информация об авторе:

ИЗРАИЛОВ Константин Евгеньевич
кандидат технических наук, доцент, старший научный сотрудник лаборатории проблем компьютерной безопасности Санкт-Петербургского Федерального исследовательского центра Российской академии наук
https://orcid.org/0000-0002-9412-5693

Автор сообщает об отсутствии конфликтов интересов.
The author declares no conflicts of interests.