Использование алгебраических моделей программ для обнаружения метаморфного вредоносного кода

Р. И. ПОДЛОВЧЕНКО
Московский государственный университет им. М. В. Ломоносова
e-mail: rip@parallel.ru

Н. Н. КУЗЮРИН
Институт системного программирования РАН

В. С. ЩЕРБИНА
Московский государственный университет им. М. В. Ломоносова

В. А. ЗАХАРОВ
Московский государственный университет им. М. В. Ломоносова

УДК 519.95

Ключевые слова: автомат, программа, обнаружение вируса, проверка эквивалентности программ.

Аннотация

Полиморфные и метаморфные вирусы — это наиболее сложные вредоносные программы, сталкиваясь с которыми, антивирусные сканеры могут испытывать значительные затруднения. Каждый раз, когда подобные вирусы заражают новые приложения или воспроизводят себя, они полностью изменяют свой код, чтобы избежать обнаружения; эта техника называется «обфускация». Указанное свойство создаёт серьёзную проблему для антивирусного программного обеспечения, которое полагается на классические методы обнаружения вредоносного кода. Это связано с тем, что в коде полиморфных и метаморфных вирусов нет постоянной последовательности инструкций, которую антивирус мог быть рассматривать в качестве идентификатора для поиска. В конечном итоге единственной характеристикой, которая остаётся неизменной для всех поколений одного и того же вируса, является их поведение (семантика). По-видимому, единственный способ достоверно определить наличие метаморфного вредоносного кода — это поиск по шаблону, который имеет ту же семантику (эквивалентное поведение), что и некоторая репрезентативная выборка вируса. Таким образом, обнаружение метаморфного вредоносного кода тесно связано с проблемой проверки эквивалентности для программ. В данной работе рассматривается новая теоретико-автоматная модель, которая может послужить основой для разработки антивирусов. Представленный подход основан на технике проверки эквивалентности в алгебраических моделях последовательных программ. Алгебраическая модель программ — это абстрактная модель вычислений, где программы рассматриваются как конечный автомат, работающий со структурами Крипке. Модели такого типа позволяют рассматривать только те характеристики операторов программы, которые широко используются для обфускирующих преобразований. В настоящей работе приводится обзор (включая последние

Фундаментальная и прикладная математика, 2009, том 15, № 5, с. 181—198.

результаты) методов решения проблемы проверки эквивалентности для различных алгебраических моделей программ и оценка устойчивости некоторых обфускирующих преобразований, которые обычно используются метаморфными вирусами.

Abstract

R. I. Podlovchenko, N. N. Kuzyurin, V. S. Shcherbina, V. A. Zakharov, Using algebraic models of programs for detecting metamorphic malwares, Fundamentalnaya i prikladnaya matematika, vol. 15 (2009), no. 5, pp. 181—198.

Polymorphic and metamorphic viruses are the most sophisticated malicious programs that give a lot of trouble to virus scanners. Each time when these viruses infect new executables or replicate themselves, they completely modify (obfuscate) their signature to avoid being detected. This contrivance poses a serious threat to antivirus software which relies on classical virus-detection techniques: such viruses do not have any stable specific sequence of instructions to be looked for. In the ultimate case, the only characteristic which remains invariable for all generations of the same virus is their functionality (semantics). To all appearance, the only way to detect for sure a metamorphic malicious code is to look for a pattern which has the same semantics as (i.e., equivalent to) some representative sample of the virus. Thus, metamorphic virus detection is closely related to the equivalence-checking problem for programs. In this paper, we outline some new automata-theoretic framework for the designing of virus detectors. Our approach is based on the equivalence-checking techniques in algebraic models of sequential programs. An algebraic model of programs is an abstract model of computation, where programs are viewed as finite automata operating on Kripke structures. Models of this kind make it possible to focus on those properties of program instructions that are widely used in obfuscating transformations. We give a survey (including the latest results) on the complexity of equivalence-checking problem in various algebraic models of programs and estimate thus a resilience of some obfuscating transformation commonly employed by metamorphic viruses.

1. Введение

Одно из наиболее важных направлений в компьютерной безопасности — это разработка антивирусного программного обеспечения. В соответствии с определением, приведённым в [31], *компьютерный вирус* — это самовоспроизводящаяся компьютерная программа, которая распространяется путём присоединения копий своего кода к программам и/или документам. Однако очень часто понятие «компьютерный вирус» относят и ко многим другим видам вредоносных программ, таким как черви, троянские кони, шпионское программное обеспечение и т. д. Первые антивирусные программы возникли в 1982 году почти одновременно с появлением компьютерных вирусов [36]. С тех пор мы являемся жертвами, а иногда и участниками непрекращающейся «гонки вооружений», в которую вовлечены создатели вредоносных программ и разработчики антивирусов. Антивирусные программы пытаются идентифицировать вирусы, мешать их работе и удалять вредоносный код. Несомненно, решающим моментом их работы является обнаружение вируса, и для того чтобы выполнить эту задачу, может быть использовано множество методов [30].

Техника обнаружения подозрительного поведения заключается в отслеживании поведения всех программ. Если одна из программ демонстрирует подозрительное поведение (например, пытается записать данные в выполняемый файл), антивирусное программное обеспечение может выявить подобные аномалии и оповестить пользователя. К сожалению, современные программы, не являющиеся вредоносными, также обладают многими свойствами, типичными для вирусов, и граница между нормальным и опасным поведением сильно размывается. Поэтому техника обнаружения подозрительного поведения имеет недостаток, заключающийся в большом количестве ложноположительных срабатываний, и большинство современных средств антивирусной защиты используют эту технику все реже и реже.

Метод песочницы заключается в эмуляции операционной системы, запуске выполняемого кода в этой виртуальной среде и анализе получившейся «песочницы»на наличие каких-либо изменений, которые позволят идентифицировать вирус. Эта техника требует значительных ресурсов и обычно применяется только в рамках сканирований, запускаемых по запросу пользователя, или в сочетании с техникой обнаружения подозрительного поведения.

Наиболее эффективная техника обнаружения вирусов основана на «словаре» вирусов. Антивирусная программа пытается найти вредоносный код внутри обычных программ путём их сканирования на предмет обнаружения так называемых *вирусных сигнатур*. Сигнатура — это характерная последовательность байт, которая служит частью определённого вируса или семейства вирусов. Если антивирусный сканер обнаруживает подобный шаблон в файле, он оповещает пользователя, что файл заражён. При поиске вирусных сигнатур антивирусный сканер обращается к базе сигнатур вирусов, которые уже известны авторам антивирусного программного обеспечения. Таким образом, проблема обнаружения вирусов сводится к хорошо известной проблеме соответствия шаблону, которая может быть решена с помощью многочисленных эффективных алгоритмов.

Сигнатурный метод эффективен только в том случае, если вирусная сигнатура остаётся неизменной для всех поколений одного и того же вируса. Поэтому для того чтобы избежать обнаружения, некоторые вирусы пытаются скрыть свои сигнатуры. Полиморфный код стал первой техникой, которая создала серьёзные проблемы антивирусным сканерам. *Полиморфные вирусы* (zombie-6.b, f0sf0r0, Hare) используют множество способов, чтобы предотвратить обнаружение сигнатуры. Во-первых, код вируса шифруется на различных ключах для каждой копии и только небольшая часть программы остаётся в открытом виде, чтобы расшифровать код перед запуском вируса. Во-вторых, когда полиморфный вирус воспроизводит себя, он не только зашифровывает своё тело с заново сгенерированным ключом, но и применяет *обфускирующие преобразования* [20, 21], чтобы изменить открытую часть кода, используемую для расшифрования. У профессионально написанного полиморфного вируса нет таких частей кода, которые оставались бы неизменными при каждом заражении, что делает невозможным обнаружение вируса непосредственно по его сигнатуре. Для обнаружения полиморфных вирусов антивирусное программное обеспечение использует эври-

стический анализ и эмуляцию программ в песочнице, чтобы выявить вирус при расшифровке тела вируса в основной памяти [32, 33].

Метаморфные вирусы (Sobig, Beagle, Smile) пытаются обойти эвристический метод обнаружения с помощью использования более сложных обфускирующих преобразований и полного переписывания тела вируса при каждом новом заражении. При использовании разработчиками вирусов более сложных обфускирующих преобразований в метаморфных вирусах эвристические методы обнаружения обречены на неудачу [38]. Кроме того, метаморфные вирусы пытаются «спутать» свой код с кодом заражаемой программы, что делает невозможным обнаружение вируса традиционными эвристическими методами, так как вредоносный код перемешан с кодом программы [29].

Таким образом, полиморфные и метаморфные вирусы бросают серьёзный вызов антивирусным сканерам. Ключевое свойство, которое делает эти вирусы особенно устойчивыми против обычных методов обнаружения вредоносного кода, — это использование обфускирующих преобразований. Обфускирующее преобразование $\mathcal{O}$ — это произвольное сохраняющее семантику преобразование, которое приводит программу π к форме $\mathcal{O}(\pi)$, которая гораздо менее понятна, чем исходная программа π. Изначально обфускация предназначалась для того, чтобы обеспечить защиту интеллектуальной собственности для программного обеспечения [20, 21]. Однако последующие исследования [14] показали, что обфускация может использоваться и в других прикладных областях, например при разработке криптосистем с открытым ключом, защите «водяных знаков» и мобильных агентов, а также (последнее, но не менее важное) при разработке вирусов-невидимок. В [14] было доказано, что обфускация, реализующая защиту типа «чёрный ящик» для всех программ, невозможна. Однако она может быть реализована для некоторых специальных семейств программ [39]. Тем не менее даже слабые обфускирующие преобразования [16, 28] могут затруднить работу алгоритмов анализа программ. Эксперименты [18, 19] показали, что три коммерческих антивирусных сканера, широко используемых на практике, можно обойти с помощью очень простых обфускирующих преобразований (путём перестановки кода и вставки команды `nop`). Это означает, что для дальнейших улучшений антивирусных сканеров, призванных сделать их эффективными механизмами защиты против вирусов-невидимок, требуются фундаментальные исследования возможностей обфускирующих и деобфускирующих методов.

Следует отметить, что обфускирующие преобразования изменяют синтаксис программы, но сохраняют её семантику. Можно предположить, что в общем случае единственной характеристикой, которая остаётся неизменной для всех поколений одного и того же метаморфного вируса, является семантика. Следовательно, семантика вируса является его подлинной «подписью», она остаётся постоянной для всех преобразований вирусного кода. Значит, для того чтобы проверить, заражена ли программа Π метаморфным вирусом π, важно выяснить, содержит ли Π фрагмент кода, который имеет ту же семантику (т. е. эквивалентное поведение), что и π. К сожалению, как было показано в [15],

проблема обнаружения вируса в общем случае неразрешима. Но если предположить, что преобразование $\mathcal{O}$, используемое для обфускации вируса π, также известно разработчикам антивируса, ситуация решительно изменится. Дело в том, что на практике обфускирующее преобразование $\mathcal{O}$ сохраняет отношение эквивалентности программ $\sim_{\mathcal{O}}$, которое гораздо сильнее функциональной эквивалентности. Следовательно, вполне возможно установить эквивалентность $\sim_{\mathcal{O}}$ в рамках некоторой достаточно простой модели вычислений (модели программ) и выработать технику проверки эквивалентности $\mathrm{EqCheck}_{\mathcal{O}}$ внутри этой модели. Алгоритм проверки эквивалентности $\mathrm{EqCheck}_{\mathcal{O}}$ может быть использован как основа для разработки антивирусной программ, которая смогла бы осуществлять проверку соответствия сигнатуре по модулю $\sim_{\mathcal{O}}$.

Этот подход к определению метаморфных вирусов был предложен М. Кристодореску и С. Джха [17]. Они описали принцип обнаружения шаблона вредоносного кода в выполняемых файлах, который может справиться с некоторыми простыми обфускирующими преобразованиями, такими как изменения в потоке управления, перераспределение регистров и добавление неисполняемых фрагментов (так называемого «мёртвого» кода). Экспериментальные результаты, которые демонстрируют эффективность их опытного образца, выглядят весьма обнадёживающе. Однако в то же время авторы работ [18, 19] отмечают, что более сложные обфускирующие преобразования, в частности перестановка кода, требуют дальнейших исследований. Некоторые из этих преобразований сильно зависят от алгебраических свойств типовых операторов программы. Большинство таких свойств может быть определено в терминах теории алгебраических моделей последовательных программ, разработанных в серии работ [4, 6, 9, 10, 27, 42].

Алгебраические модели программ аппроксимируют операционную семантику реальных императивных программ и таким образом способствуют развитию алгоритмов проверки эквивалентности и преобразований оптимизации для программ. Пусть даны два множества: множество базовых операторов программы $\mathcal{A}$ и множество базовых предикатов $\mathcal{P}$. Программа π в рамках алгебраической модели рассматривается как детерминированный конечный автомат с входным алфавитом $\mathcal{A}$ и выходным алфавитом $\rho(\mathcal{P})$ всех возможных значений базовых предикатов. Такой автомат $\mathcal{A}$ функционирует на множестве $\mathcal{M}$ помеченных структур Крипке, которые представляют семантику программных операторов и предикатов. Модели такого типа называются алгебраическими, так как в большинстве практических случаев множество $\mathcal{M}$ соответствует некоторой полугруппе, которая включает наиболее важные алгебраические свойства операционной семантики реальных программ. Основное преимущество этих моделей — их масштабируемость: выбирая множество алгебраических и логических свойств операторов и предикатов, можно построить подходящую модель программ, для которых семантика обладает именно заданным набором свойств. Это очень удобно для разработки сканеров для поиска метаморфных вирусов. При заданном обфускирующем преобразовании $\mathcal{O}$, используемом в метаморфном механизме вируса, можно:

- выбрать алгебраическую модель $\mathcal{M}$, такую что $\pi \sim_{\mathcal{M}} \mathcal{O}(\pi)$ выполняется для каждой программы π (в этом случае говорят, что обфускация $\mathcal{O}$ выразима в модели $\mathcal{M}$);
- оценить сложность решения задачи проверки эквивалентности для программ в модели $\mathcal{M}$; это даёт приблизительную оценку сложности проблемы обнаружения вируса;
- разработать (если удастся) алгоритм проверки эквивалентности для модели $\mathcal{M}$ и использовать его как средство сканирования для метаморфных вирусов, использующих данную обфускацию $\mathcal{O}$.

Само понятие алгебраической модели программ было введено в [27]. С тех пор проблема проверки эквивалентности для программ в алгебраических моделях была широко изучена и были разработаны некоторые универсальные подходы к её решению, которые сочетают теоретико-автоматные и теоретико-групповые методы. Оказывается (см. [13, 26, 40, 42]), что для многих алгебраических моделей программ проблема проверки эквивалентности разрешима за полиномиальное время. Поэтому можно ожидать, что некоторые метаморфные вирусы можно обнаруживать с помощью усовершенствованной техники поиска сигнатур. Аналогичный подход, основанный не на алгебраических моделях программ, а на понятии абстрактной интерпретации, был представлен в [22—24]. Данный метод используется для оценки устойчивости программных водяных знаков.

С другой стороны, некоторые более поздние результаты показывают, что проблема проверки эквивалентности в некоторых естественных алгебраических моделях трудноразрешима. Это означает, что обфускирующие преобразования в рамках этих моделей могут обладать довольно сильной устойчивостью по отношению к поисковому механизму, основанному на проверке эквивалентности. Надеемся, что дальнейшие исследования прольют свет на этот эффект.

Оставшаяся часть данной работы организована следующим образом. В разделе 2 вводятся формальные понятия алгебраических моделей программ. В разделах 3 и 4 приводится обзор результатов, касающихся сложности проблемы проверки эквивалентности для некоторых алгебраических моделей. Наконец, в разделе 5 обсуждается вопрос влияния этих результатов на проблему обнаружения метаморфных вирусов.

2. Алгебраические модели программ

Для алгебраических моделей программ рассматриваются последовательные компьютерные программы на логическом уровне абстракции. В данном разделе определяется синтаксис и семантика пропозициональных последовательных программ.

Пусть $\mathcal{A} = \{a_1, \ldots, a_r\}$ и $\mathcal{P} = \{c_1, \ldots, c_k\}$ — два конечных алфавита. Обозначим через r и k мощности алфавитов $\mathcal{A}$ и $\mathcal{P}$ соответственно. Подчеркнём, что эти параметры фиксированы для каждой рассматриваемой алгебраической модели программ. Элементы алфавита $\mathcal{A}$ называются *базовыми операторами*.

Как интуитивно понятно, каждый базовый оператор соответствует некоторому оператору присваивания в императивной программе. Но гораздо лучше он может соответствовать вызову библиотечной функции, базовому элементу программы или любому другому фрагменту программы, который обязательно заканчивает свою работу.

Элементы множества $\mathcal{P}$ называются *базовыми предикатами*. Они соответствуют элементарным встроенным соотношением между данными программы. Каждый базовый предикат может принимать значение *ложь* или *истина*. Множество всех подмножеств $\mathcal{P}$ обозначим через $\rho(\mathcal{P})$; элементы из $\rho(\mathcal{P})$ соответствуют всевозможным значениям базовых предикатов.

Определение 1. *Пропозициональная последовательная программа* — это конечная переходная система $\pi = \langle V, \textsf{вход}, \textsf{выход}, T, B\rangle$, где

- V — непустое множество *шагов программы*;
- **вход** — *точка входа* программы;
- **выход** — *точка выхода* программы;
- $T\colon (V - \{\textsf{выход}\}) \times \rho(\mathcal{P}) \to V$ — (общая) *функция переходов*;
- $B\colon (V - \{\textsf{выход}\}) \to \mathcal{A}$ — (общая) *функция привязки*.

Функция переходов соответствует потоку управления программы, тогда как функция привязки определяет соответствие для каждого шага некоторого базового оператора. Можно также рассматривать пропозициональную последовательную программу как детерминированный конечный автомат, функционирующий на входном алфавите $\rho(\mathcal{P})$ и выходном алфавите $\mathcal{A}$. *Размером* $|\pi|$ программы π называется мощность множества V (предполагается, что мощности $\mathcal{A}$ и $\mathcal{P}$ фиксированы).

Семантика пропозициональной последовательной программы определяется с помощью полугрупповых структур Крипке.

Определение 2. *Полугрупповая структура Крипке* — это тройка $M = \langle S, \circ, \xi\rangle$, где

- $(S, \circ)$ — *полугруппа*, порождённая множеством $\mathcal{A}$ базовых операторов;
- $\xi\colon S \to \rho(\mathcal{P})$ — это (итоговая) *функция разметки*.

Полугруппа $(S, \circ)$ позволяет интерпретировать базовые операторы. Элементы S могут рассматриваться как состояния данных, а нейтральный элемент (единица) ε соответствует начальному состоянию данных. Когда оператор a, $a \in \mathcal{A}$, применяется к состоянию данных s, $s \in S$, результат применения a — это состояние данных $s' = s \circ a$. Функция разметки ξ используется для интерпретации базовых предикатов: при заданном состоянии данных s функция разметки $\xi(s)$ возвращает множество всех базовых предикатов, которые принимают значение *истина* на s.

Пусть π — это пропозициональная последовательная программа, а M — полугрупповая структура Крипке. *Вычисление* программы π на M — это последовательность пар (конечная или бесконечная)

$$r(\pi, M) = (v_0, s_0), (v_2, s_2), \ldots, (v_i, s_i), (v_{i+1}, s_{i+1}), \ldots,$$

такая что

1) $v_0 = \textbf{вход}$, $s_0 = \varepsilon$ — начальное состояние данных в M;
2) $v_{i+1} = T\big(v_i, \xi(s_i)\big)$, $s_{i+1} = s_i \circ B(v_i)$ выполнено для каждого i, $i \geqslant 1$;
3) последовательность $r(\pi, M)$ либо бесконечная (тогда говорят, что вычисление *зацикливается* и не даёт результатов), либо заканчивается парой (v_n, s_n), такой что $v_n = \textbf{выход}$ (тогда говорят, что вычисление *завершается* с результатом s_n).

Обозначим через $[r(\pi, M)]$ результат вычисления $r(\pi, M)$, предполагая, что результат не определён, когда $r(\pi, M)$ зацикливается.

Алгебраической моделью программ $\mathcal{M}$ назовём множество всех пропозициональных последовательных программ над фиксированными алфавитами $\mathcal{A}$, $\mathcal{P}$, семантика которых определяется множеством $\mathcal{M}$ полугрупповых структур Крипке.

Определение 3. Пусть дана алгебраическая модель программ $\mathcal{M}$. Программы π_1 и π_2 называются *эквивалентными в* $\mathcal{M}$ (обозначение: $\pi_1 \sim_{\mathcal{M}} \pi_2$), если $[r(\pi_1, M)] = [r(\pi_2, M)]$ выполнено для каждой полугрупповой структуры Крипке M из $\mathcal{M}$.

Проблема проверки эквивалентности в алгебраической модели программ $\mathcal{M}$ — это проблема проверки соотношения $\pi_1 \sim_{\mathcal{M}} \pi_2$ для данной произвольной пары пропозициональных последовательных программ π_1 и π_2.

Как видно из определений выше, алгебраическая модель программ — это всего лишь абстрактная модель вычислений, где программы рассматриваются как детерминированные конечные автоматы, функционирующие на структурах Крипке. В связи с этим проблема проверки эквивалентности для алгебраических моделей программ — это не что иное, как обобщение хорошо известной задачи проверки эквивалентности детерминированных конечных автоматов.

Будем говорить, что алгебраическая модель $\mathcal{M}$ *аппроксимирует* отношение эквивалентности $\equiv$ на множестве пропозициональных последовательных программ, если из $\pi_1 \sim_{\mathcal{M}} \pi_2$ следует $\pi_1 \equiv \pi_2$ для каждой пары программ π_1 и π_2. Отношение аппроксимации создаёт решётку на множестве всех алгебраических моделей программ [4, 9]. Следовательно, при заданном эквивалентном (обфускирующем) преобразовании программ $\mathcal{O}$ можно рассматривать алгебраическую модель $\mathcal{M}_{\mathcal{O}}$, которая служит лучшей аппроксимацией для отношения эквивалентности $\equiv_{\mathcal{O}}$, индуцированного преобразованием $\mathcal{O}$, а затем решать задачу проверки эквивалентности для $\mathcal{M}_{\mathcal{O}}$. Если задача неразрешима (или труднаразрешима), можно выбрать подходящую аппроксимацию $\mathcal{M}'$ для модели $\mathcal{M}_{\mathcal{O}}$ и разработать эффективный алгоритм проверки эквивалентности для $\mathcal{M}'$. Этот алгоритм может быть использован деобфускирующим антивирусным сканером, чтобы определить метаморфный вредоносный код, для обфускации которого используется преобразование $\mathcal{O}$.

В следующем разделе детально рассматриваются несколько алгебраических моделей, которые могут быть использованы для раскрытия наиболее распространённых обфускаций.

3. Задача проверки эквивалентности для основных алгебраических моделей программ

Ограничимся рассмотрением алгебраических моделей программ $\mathcal{M}$, для которых структуры M, $M \in \mathcal{M}$, основаны на одной полугруппе $(S, \circ)$. Каждая модель $\mathcal{M}$ такого типа полностью определяется парой (S, L), где $L = \{\xi\colon \langle S, \circ, \xi\rangle \in \mathcal{M}\}$ — множество допустимых функций разметки. Если, кроме того, каждая функция разметки ξ, определённая на S, допустима (т. е. $\langle S, \circ, \xi\rangle \in \mathcal{M}$ выполнено для любой функции ξ), то такую модель $\mathcal{M}$ будем записывать как (S, L_S).

Далее рассмотрим несколько семейств алгебраических моделей программ, основанных на простых полугруппах. Для каждой модели $\mathcal{M}$ выясним, какой тип обфускации выразим внутри этой модели, и оценим сложность решения задачи проверки эквивалентности для $\mathcal{M}$. Можно рассматривать эти оценки как некие качественные характеристики устойчивости обфускирующих преобразований, которые выразимы в рамках $\mathcal{M}$.

3.1. Свободные полугруппы. Максимальная модель $\mathcal{M}_0 = (S_0, L_{S_0})$

Максимальная модель $\mathcal{M}_0$ образована с помощью свободной полугруппы S_0, порождённой множеством базовых операторов $\mathcal{A}$. Эта модель называется максимальной, так как она аппроксимирует любую другую алгебраическую модель программ. Обфускирующие преобразования в рамках $\mathcal{M}_0$ могут изменить структуру потока управления программы, однако они не могут повлиять на порядок выполнения операторов. Понятие максимальной модели ввёл Янов [27] в 1958 году, и вскоре эта модель получила название схем Янова. В [27] было доказано, что задача проверки эквивалентности для максимальной алгебраической модели программ разрешима. Фактически, это первый положительный результат, который продемонстрировал применимость формальных методов к анализу программ. Позже в [35] было показано, что задача проверки эквивалентности для $\mathcal{M}_0$ и задача проверки эквивалентности для детерминированных конечных автоматов сводятся друг к другу. Принимая во внимание большинство эффективных алгоритмов проверки выполнимости для конечных автоматов, представленных в [26], получим теорему, приведённую ниже.

Теорема 1. *Задача проверки эквивалентности программ «$\pi_1 \sim_{\mathcal{M}_0} \pi_2$?» в максимальной модели $\mathcal{M}_0$ разрешима за время $O(n \log n)$, где n — суммарный размер программ π_1 и π_2.*

Существует множество способов усовершенствования максимальной модели. Например, можно предположить, что полугруппа S_0 содержит несколько

единичных элементов. Каждый такой единичный элемент соответствует несущественным операторам, выполнение которых не меняет состояния данных (например, оператор `nop` в множестве операторов для архитектуры IA-32). Как нетрудно проверить, сложность задачи проверки эквивалентности программ для этой алгебраической модели та же, что и для $\mathcal{M}_0$. Этот факт объясняет эффективность статического анализатора для обнаружения шаблонов вредоносного кода в приложениях [17, 19], разработанного Кристодореску с соавторами: он раскрывает в точности те обфускирующие преобразования, которые выразимы в улучшенном варианте максимальной алгебраической модели программ.

Другие усовершенствования модели $\mathcal{M}_0$ также представляют некоторый интерес относительно обфускации программ. Обозначим через $\mathrm{mod}(a)$ множество переменных программы, значения которых изменяются оператором a, через $\mathrm{used}(a)$ ($\mathrm{used}(p)$) — множество переменных программы, значения которых используются оператором a (предикатом p). Ясно, что если $\mathrm{mod}(a) \cap \mathrm{used}(p) = \varnothing$, то значение истинности предиката p до и после выполнения a не меняется. В этом случае говорится, что предикат p является *инвариантом* по отношению к оператору a. Например, предикат `y == 0` — инвариант по отношению к оператору `x++`. Обычно с помощью очень простого синтаксического анализа можно проверить, является ли некоторый предикат p инвариантом по отношению к оператору a. Алгебраические модели программ дают возможность представить этот эффект. Пусть $Z\colon \mathcal{A} \to 2^{\mathcal{P}}$ — отображение (инвариантное распределение), которое ставит в соответствие каждому базовому оператору a множество базовых предикатов, которые инварианты относительно a. Скажем, что функция разметки ξ *сохраняет* инвариантное распределение Z, если условие $\big(p \in \xi(s) \iff p \in \xi(s \circ a)\big)$ выполнено для каждого состояния данных s, $s \in S_0$, базового оператора a, $a \in \mathcal{A}$, и базового предиката $p \in Z(a)$. Обозначим через L_Z множество всех функций разметки, сохраняющих инвариантное распределение Z на свободной полугруппе S_0. Как было показано в [27], задача проверки эквивалентности программ для любой модели $M_{0,Z} = (S_0, L_Z)$ имеет ту же сложность, что задача проверки эквивалентности программ для максимальной модели.

Другой эффект, который может быть принят во внимание при усовершенствовании максимальной модели, — это монотонность предикатов по отношению к операторам программы. Например, если предикат `x > 0` принимает значение *истина* на некотором шаге вычисления программы, то он сохраняет это значение после выполнения оператора `x++`. Скажем, что функция разметки ξ *монотонна* для предиката p, если условие $\big(p \in \xi(s) \Longrightarrow p \in \xi(s \circ a)\big)$ выполнено для каждого состояния данных s, $s \in S_0$, и базового оператора a, $a \in \mathcal{A}$. Таким же образом, как было описано выше, можно рассмотреть семейство алгебраических моделей $\mathcal{M}_{0m} = (S_0, L_m)$, для которых функции разметки монотонны для некоторых базовых предикатов. В [3] было показано, что задача проверки эквивалентности для любой монотонной модификации максимальной алгебраической модели разрешима за время $O(n \log n)$. В [3] также рассмотрены некоторые другие улучшения $\mathcal{M}_0$.

Необходимо отметить, что оба свойства, рассмотренные выше, хорошо подходят для создания *«непроницаемых» предикатов* [20, 21]. «Непроницаемые» предикаты предназначены для затруднения понимания программы; они широко используются в обфускации программ для вставки неисполняемых фрагментов, изменения потока управления и потока данных и т. д. «Непроницаемые» предикаты также могут быть использованы метаморфным вирусом для «спутывания» своего кода с кодом заражённого приложения, а также для имитации выполнения вирусом служебных функций, характерных для разрешённых приложений. Эти возможности обфускирующих преобразований должны быть приняты во внимание при разработке современных антивирусных сканеров.

3.2. Коммутативные полугруппы. Модели $\mathcal{M}_1 = (S_1, L_{S_1})$ с коммутативными операторами

Семантика некоторых операторов такова, что результат их выполнения не зависит от порядка, в котором эти операторы выполняются. Для того чтобы удостовериться, что операторы a и b можно поменять местами, достаточно проверить с помощью простого синтаксического анализатора, что $\mathrm{used}(a) \cap \mathrm{mod}(b) = \mathrm{used}(b) \cap \mathrm{mod}(a) = \mathrm{mod}(a) \cap \mathrm{mod}(b) = \varnothing$. Это выполнено, например, для операторов `x++` и `z+=&y`. Для того чтобы использовать это, необходимо рассмотреть (частично) коммутативные полугруппы S_1, порождённые множеством определяющих соотношений $a \circ b = b \circ a$. Алгебраическая модель программ $\mathcal{M}_1 = (S_1, L_{S_1})$, основанная на (частично) коммутативных полугруппах S_1, называется *коммутативной моделью*.

Перестановка кода, основанная на коммутативности операторов, — это одно из самых простых и эффективных обфускирующих преобразований, которое может «сбить со следа» все распространённые антивирусные сканеры, основанные на проверке соответствия шаблону [17]. Данная обфускация перемешивает операторы так, что их порядок для различных репликаций вируса отличается от порядка операторов, указанного в сигнатуре, используемой антивирусом. Тем не менее в [13, 40, 41] приведён достаточно действенный поход к разработке эффективных алгоритмов проверки эквивалентности для пропозициональных последовательных программ, который может справиться с этой проблемой.

Теорема 2. *Задача проверки эквивалентности* «$\pi_1 \sim_{\mathcal{M}_1} \pi_2$?» *в модели пропозициональных последовательных программ с коммутативными операторами* $\mathcal{M}_1$ *разрешима за время* $O(n^3 \log n)$, *где* n — *суммарный размер программ* π_1 *и* π_2.

3.3. Модели программ $\mathcal{M}_2 = (S_2, L_{S_2})$ с подавляемыми операторами

Будем говорить, что базовый оператор a *подавляет* оператор b, если результат выполнения b утрачивается каждый раз, когда вслед за ним выполняется оператор a. Так, например, оператор `x=y++` подавляет оператор `x++`. Чтобы убедиться, что оператор a подавляет оператор b, достаточно проверить, что

$\mathrm{mod}(b) \subseteq \mathrm{mod}(a)$ и $\mathrm{mod}(b) \cap \mathrm{used}(a) = \varnothing$. Вставка и удаление подавляемых операторов — это ещё один вид обфускирующих преобразований, который позволяет вирусу незаметно изменять длину своего кода (сигнатуру). Так как добавленные операторы выполняют полезные действия, это может ввести в заблуждение даже антивирусные средства, использующие подход эмуляции программ в песочнице.

Эффект подавления одних операторов другими может быть хорошо отражён в семействе алгебраических моделей $\mathcal{M}_2 = (S_2, L_{S_2})$, в котором полугруппа S_2 полностью определяется тождествами вида $b \circ a = a$. На основе техники, приведённой в [40], нами была доказана следующая теорема.

Теорема 3. *Задача проверки эквивалентности «*$\pi_1 \sim_{\mathcal{M}_2} \pi_2$*?» в модели* $\mathcal{M}_2$ *пропозициональных последовательных программ с подавляемыми операторами разрешима за время* $O(n^2 \log n)$*, где* n *— суммарный размер программ* π_1 *and* π_2*.*

3.4. Полугруппы с правыми нулями. Модели $\mathcal{M}_3 = (S_3, L_{S_3})$ с переключением режима

Вычисление программы *с переключением режима* разбивается на два этапа. На первом этапе программа выбирает подходящий режим вычислений. До того как сделать окончательный выбор, программа может попробовать несколько режимов. Каждый раз, когда следующий режим включается в тестирование, программа приводит данные к некоторому заранее определённому состоянию. На втором этапе, когда окончательный режим зафиксирован, определяется конечный результат вычисления. В реальных программах переключение режима может заключаться в перезапуске операторов или в присваивании констант, как, например, `x=(y=0)+(z=1)`. Характерное свойство оператора переключения режима a состоит в том, что для него выполнено $\mathrm{used}(a) = \varnothing$ и $\mathrm{mod}(a)$ включает все переменные программы. Так как каждый вызов оператора переключения режима всегда даёт один и тот же результат, дальнейшее поведение программы хорошо предсказуемо. Поэтому операторы переключения режима используются как подходящее средство для порождения «непроницаемых» предикатов, результат которых становится известным в ходе обфускации, но обфускацию сложно отследить [20].

Эффект операторов переключения режима отражается в алгебраических моделях программ $\mathcal{M}_3 = (S_3, L_{S_3})$, где S_3 — это свободная полугруппа с несколькими правыми нулями, т. е. элементами e, которые удовлетворяют уравнению $a \circ e = e$ для каждого базового оператора a. В [5] было доказано, что задача проверки эквивалентности для программ с операторами переключения режима разрешима. Более эффективный алгоритм проверки эквивалентности для $\mathcal{M}_3$ был представлен в [34].

Теорема 4. *Задача проверки эквивалентности «*$\pi_1 \sim_{\mathcal{M}_3} \pi_2$*?» в модели пропозициональных последовательных программ с операторами переключения режима* $\mathcal{M}_3$ PSPACE*-полна.*

3.5. Свободные группы. Модели $\mathcal{M}_4 = (S_4, L_{S_4})$ с обратимыми операторами

Два оператора a и b называются (взаимно) *обратимыми*, если их последовательное выполнение оставляет данные в исходном состоянии. Операторы `x++` и `x--` представляют собой наиболее простой пример пары взаимно обратимых операторов. Обратимые операторы, как и присвоение констант (или операторы переключения режима), могут быть использованы для перевода данных в некоторое заранее определённое состояние и для генерации таким образом «непроницаемых» предикатов. Однако, так как состояние данных, которое получается после выполнения последовательности взаимно обратимых операторов, не фиксировано (в отличие от операторов переключения режима), поверхностный статический анализ не может справиться с этим эффектом. Поэтому необходимо более глубокое изучение семантических особенностей поведения программ.

Поведение программ с обратимыми операторами может быть проанализировано с помощью алгебраических моделей программ $\mathcal{M}_4 = (S_4, L_{S_4})$, где S_4 — это свободная группа. В [6, 7] было показано, что проблема проверки эквивалентности алгебраических моделей программ с обратимыми операторами разрешима. Недавно нами была оценена сложность задачи проверки эквивалентности для $\mathcal{M}_4$.

Теорема 5. *Задача проверки эквивалентности* «$\pi_1 \sim_{\mathcal{M}_4} \pi_2$?» *в модели* $\mathcal{M}_4$ *пропозициональных последовательных программ с обратимыми операторами* PSPACE*-полна.*

4. Проблема проверки эквивалентности для комбинированных алгебраических моделей программ

Зачастую для анализа поведения программ необходимо одновременно учитывать несколько различных семантических свойств операторов и предикатов. Это приводит к необходимости рассматривать более сложные алгебраические модели программ. Некоторые из них получаются из комбинации некоторых базовых моделей, описанных выше. Оказывается, однако, что, соединив вместе некоторые довольно простые алгебраические модели, часто можно получить комбинированную модель, гораздо более сложную для анализа. Ниже мы рассмотрим несколько примеров, подтверждающих это наблюдение.

4.1. Алгебраические модели с коммутативными операторами и монотонными предикатами

Как было показано в разделе 3.1, задача проверки эквивалентности для алгебраических моделей программ с монотонными предикатами разрешима за время

$O(n \log n)$. В разделе 3.2 мы также показали, что та же проблема для алгебраических моделей с коммутативными операторами разрешима за время $O(n^3 \log n)$. В обоих случаях сложность решения задачи проверки эквивалентности не зависит существенно от мощности множеств $\mathcal{A}$ и $\mathcal{P}$ базовых операторов и предикатов. Но если рассмотреть комбинированную модель $\mathcal{M}_{0m} + \mathcal{M}_1$, которая включает коммутативные операторы и монотонные предикаты, то наилучший известный алгоритм проверки эквивалентности [43] для этой модели не так эффективен, как алгоритмы для $\mathcal{M}_{0m}$ и $\mathcal{M}_1$.

Теорема 6. *Задача проверки эквивалентности «$\pi_1 \sim_{\mathcal{M}_0} \pi_2$?» в комбинированной модели $\mathcal{M}_{0m}+\mathcal{M}_1$ разрешима за время $n^{O(rk)}$, где n — суммарный размер программ π_1 и π_2, а r и k — мощности алфавитов $\mathcal{A}$ и $\mathcal{P}$.*

Эквивалентная проблема преобразования для этого класса алгебраических моделей была рассмотрена в [12]. Проблема проверки эквивалентности гораздо более сложна для алгебраических моделей, у которых функции разметки подвержены инвариантным распределениям Z. Пусть $M_{0Z} + M_1 = (S_1, L_{(S_1,Z)})$ — алгебраическая модель, где S_1 — свободная коммутативная полугруппа, а Z — инвариантное распределение, такое что условие $p \notin Z(a) \Longrightarrow p \in Z(b)$ выполнено для каждой пары базовых операторов a, b, $a \neq b$, и предиката p. Тогда задача проверки эквивалентности для $M_{0Z} + M_1$ и задача проверки эквивалентности для детерминированных многоленточных автоматов сводятся друг к другу. В течение более чем 30 лет не было известно, разрешима ли вторая проблема. Но в 1991 году Харжу и Кархумаки [25] представили процедуру, которая решает эту проблему за экспоненциальное время. Тем не менее, насколько нам известно, сложность задачи проверки эквивалентности для детерминированных многоленточных автоматов до сих пор не определена.

4.2. Алгебраические модели программ с коммутативными операторами и операторами переключения режима

Соединяя вместе алгебраические модели программ, представленные в разделах 3.2 и 3.4, мы получим алгебраическую модель $\mathcal{M}_1 + \mathcal{M}_3$, для которой структуры Крипке основаны на свободной коммутативной полугруппе, дополненной правыми нулями. Задача проверки эквивалентности для этой модели была изучена А. Б. Годлевским. В работе [1] он представил алгоритм проверки эквивалентности для $\mathcal{M}_1 + \mathcal{M}_3$, который имеет сверхэкспоненциальную временну́ю сложность. Создание более эффективных процедур проверки эквивалентности и получение более точной оценки сложности решения задачи проверки эквивалентности требуют проведения дальнейших исследований.

Можно также рассматривать другую комбинацию $\mathcal{M}_1$ и $\mathcal{M}_3$, которая приводит к алгебраической модели со структурами Крипке, основанными на прямом произведении свободных коммутативных полугрупп, каждая из которых дополнена правыми нулями. Такая модель соответствует случаю, когда в программе

используются операторы `x++`, `y++`, `x=0` и `y=0`. В [2, 8] было показано, что задача проверки эквивалентности для таких алгебраических моделей неразрешима.

4.3. Алгебраические модели программ с коммутативными и обратимыми операторами

Если совместить алгебраические модели, представленные в разделах 3.2 и 3.5, мы получим модель со структурами Крипке, основанными на абелевых группах. Эта модель подходит для программ, которые содержат пары обратимых операторов `x++`, `x--`, `y++`, `y--` и т. д., каждый из которых работает со своей переменной. В [6] было показано, что задача проверки эквивалентности для алгебраических моделей $\mathcal{M}_1 + \mathcal{M}_4$ неразрешима.

4.4. Алгебраические модели программ с подавляемыми и коммутативными операторами

Тем не менее в некоторых случаях сложность решения задачи проверки эквивалентности для комбинированной модели не превышает сложности решения задачи проверки эквивалентности для её компонентов. Так, например, можно рассмотреть алгебраическую модель $\mathcal{M}_1 + \mathcal{M}_3$ с подавляемыми и коммутативными операторами, которая получена путём соединения моделей, приведённых в разделах 3.2 и 3.3.

Теорема 7. *Задача проверки эквивалентности «$\pi_1 \sim_{\mathcal{M}_2} \pi_2$?» в комбинированной модели $\mathcal{M}_1 + \mathcal{M}_3$ пропозициональных последовательных программ с подавляемыми и коммутативными операторами разрешима за время $O(n^4 \log n)$, где n — суммарный размер программ π_1 и π_2.*

5. Заключение

Теоремы 1—3 и 7 показывают, что в некоторых случаях алгоритмы проверки эквивалентности для алгебраических моделей программ очень эффективны (они имеют полиномиальную сложность, и степень полинома невысока). Это означает, что обфускирующие преобразования, которые выразимы в таких моделях, далеко не безопасны. Если метаморфный вирус использует только такие преобразования для обфускации, то его можно обнаружить с помощью современных методов проверки соответствия шаблону, основанных на универсальных и эффективных процедурах проверки эквивалентности.

В то же время задача проверки эквивалентности для некоторых алгебраических моделей может быть очень сложной и даже неразрешимой (см. теоремы 4—6). Метаморфные вирусы, использующие обфускирующие преобразования, выразимые в этих моделях, могут быть весьма устойчивыми. В таких случаях

универсальные методы проверки эквивалентности оказываются неэффективными. К счастью, это также означает, что применение подобных обфускирующих преобразований является сложной задачей, которая требует глубокого анализа программного обеспечения и поглощает значительное количество вычислительных ресурсов. Так как компьютерный вирус, как правило, имеет компактный код, вряд ли возможно, что для него будут применяться очень сложные обфускации.

Поэтому, на наш взгляд, наиболее подходящей для метаморфных вирусов является следующая стратегия обфускации. Предположим, что $\mathcal{M} = \mathcal{M}_{i_1} + \mathcal{M}_{i_2} + \ldots + \mathcal{M}_{i_k}$ — сложная алгебраическая модель, которая составлена из нескольких простых моделей $\mathcal{M}_{i_1}, \mathcal{M}_{i_2}, \ldots, \mathcal{M}_{i_k}$. Также допустим, что каждая модель $\mathcal{M}_{i_j}$ допускает конструирование простых и эффективных механизмов для того, чтобы создать достаточное количество обфускирующих преобразований. Тогда для того чтобы преобразовать свой код, вирус π может применять случайно выбранные серии обфускирующих преобразований $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_N$, которые выразимы в моделях $\mathcal{M}_{i_1}, \mathcal{M}_{i_2}, \ldots, \mathcal{M}_{i_k}$. Порядок, в котором применяются эти преобразования, служит «секретным ключом» обфускации $\mathcal{O}$, полученной таким образом. Если задача проверки эквивалентности для $\mathcal{M}$ неразрешима, то применение подхода проверки эквивалентности напрямую для деобфускации $\mathcal{O}(\pi)$ может занять много времени. Этой сложности можно избежать, если учесть строение $\mathcal{M}$ и использовать деобфускирующие процедуры для каждой простой модели, входящей в состав $\mathcal{M}$. Однако определить правильный порядок, в котором применялись эти процедуры в $\mathcal{O}(\pi)$, без знания «секретного ключа» обфускации тоже сложно. Представленные соображения соответствуют результатам, полученным в работе [37]. Возможность применения эквивалентных преобразований, полученных на основе алгебраических моделей, для обфускации программ изучена в [11].

На наш взгляд, дальнейшее исследование рассмотренной темы может быть очень полезным для понимания возможностей метаморфных вирусов и для разработки нового антивирусного программного обеспечения.

Литература

[1] Годлевский А. Б. Некоторые специальные случаи проблемы остановки и функциональной эквивалентности автоматов // Кибернетика. — 1973. — № 4. — С. 90—97.

[2] Годлевский А. Б. Об одном разрешимом случае специальной проблемы функциональной эквивалентности дискретных преобразователей // Кибернетика. — 1974. — № 3. — С. 32—36.

[3] Захаров В. А. Автоматные модели программ // ДАН СССР. — 1989. — Т. 309, № 1. — С. 24—27.

[4] Захаров В. А. Об одном критерии сравнимости операторных формальных моделей программ // Программирование. — 1993. — № 4. — С. 12—25.

[5] Летичевский А. А. Эквивалентность автоматов относительно свободной полугруппы с правым нулём // Докл. АН УССР. — 1968. — Т. 182, № 5. — С. 248—252.

[6] Летичевский А. А. Эквивалентность автоматов относительно полугрупп // Теор. кибернетика. — 1970. — Вып. 6. — С. 3—71.

[7] Летичевский А. А., Смикун Л. Б. О классах групп с разрешимой проблемой эквивалентности автоматов // ДАН СССР. — 1976. — Т. 17, № 2. — С. 341—344.

[8] Петросян Г. Н. Об одном базисе операторов и предикатов с неразрешимой проблемой пустоты // Кибернетика. — 1974. — № 5. — С. 23—28.

[9] Подловченко Р. И. Иерархия моделей программ // Программирование. — 1981. — № 2. — С. 3—14.

[10] Подловченко Р. И. Полугрупповые модели программ // Программирование. — 1981. — № 4. — С. 3—13.

[11] Подловченко Р. И. Эквивалентные преобразования схем программ для «запутывания» самих программ // Программирование. — 2002. — Т. 28, № 2. — С. 106—116.

[12] Подловченко Р. И. О схемах программ с перестановочными и монотонными операторами // Программирование. — 2003. — Т. 29, № 5. — С. 270—276.

[13] Подловченко Р. И., Захаров В. А. Полиномиальный по сложности алгоритм, распознающий коммутативную эквивалентность схем программ // Докл. РАН. — 1998. — Т. 362, № 6. — С. 744—747.

[14] Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vedhan S., Yang K. On the (im)possibility of obfuscating programs // CRYPTO'01 — Advances in Cryptology. — Springer, 2001. — (Lect. Notes Comput. Sci.; Vol. 2139). — P. 1—18.

[15] Chess D., White S. An undetectable computer virus // Proc. of the 2000 Virus Bulletin Conference. — 2000.

[16] Chow S., Gu Y., Johnson H., Zakharov V. A. An approach to the obfuscation of control-flow of sequential computer programs // 6th Inf. Security Conf. — Springer, 2001. — (Lect. Notes Comput. Sci.; Vol. 2200). — P. 144—155.

[17] Christodorescu M., Jha S. Static analysis of executables to detect malicious patterns // Proc. of the 12th USENIX Security Symposium (Security'03). — 2003. — P. 169—186.

[18] Christodorescu M., Jha S. Testing malware detectors // Proc. of the Int. Symp. on Software Testing and Analysis (ISSTA 2004). — 2004.

[19] Christodorescu M., Jha S., Seshia S. A., Song D., Bryant R. E. Semantics-aware malware detection // Proc. of the 2005 IEEE Symp. on Security and Privacy (Oakland, 2005). — 2005. — P. 32—46

[20] Collberg C., Thomborson C., Low D. A taxonomy of obfuscating transformations: Tech. Report, No. 148. — Univ. of Auckland, 1997.

[21] Collberg C., Thomborson C., Low D. Manufacturing cheap, resilient and stealthy opaque constructs // Symp. on Principles of Programming Languages. — 1998. — P. 184—196.

[22] Cousot P., Cousot R. An abstract interpretation-based framework for software watermarking // Conf. Record of the 31st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Programming Languages. — 2004. — P. 173—185.

[23] Dalla Preda M., Giacobazzi R. Semantic-based code obfuscation by abstract interpretation // Automata, Languages and Programming. — Springer, 2005. — (Lect. Notes Comput. Sci.; Vol. 3580). — P. 1325—1336.

[24] Dalla Preda M., Giacobazzi R., Madou M., de Bosschere B. Opaque predicate detection by means of abstract interpretations // Proc. of 11th Int. Conf. on Algebraic Methodology and Software Technology (AMAST06). — Springer, 2006. — (Lect. Notes Comput. Sci.; Vol. 4019). — P. 81—95.

[25] Harju T., Karhumaki J. The equivalence of multi-tape finite automata // Theor. Comput. Sci. — 1991. — Vol. 78. — P. 347—355.

[26] Hopcroft J. E., Karp R. M. A linear algorithm for testing equivalence of finite automata: Technical Report TR 71-114. — Cornell Univ., Comput. Sci. Dep., 1971.

[27] Ianov Iu. I. On the equivalence and transformation of program schemes // Comm. ACM. — 1958. — Vol. 1, no. 10. — P. 8—12.

[28] Ivanov K. S., Zakharov V. A. Program obfuscation as obstruction of program static analysis // Proc. of the Inst. for System Programming. — 2004. — Vol. 6. — P. 141—161.

[29] Kaspersky E. Virus List Encyclopedia. — Kaspersky Lab., 2002.

[30] Marx A. A guideline to anti-malware-software testing // Proc. of the 9th Ann. European Institute for Computer Antivirus Research Conference (EICAR'00). — 2000.

[31] McGrow G., Morriset G. Attacking malicious code: report to the Infosec research council // IEEE Software. — 2000. — Vol. 17, no. 5. — P. 33—41.

[32] Nachenberg C. Understanding and managing polymorphic viruses // The Symantec Enterprise Papers. — 1996. — Vol. 30.

[33] Nachenberg C. Computer virus-antivirus coevolution // Comm. ACM. — 1997. — Vol. 40, no. 1. — P. 46—51.

[34] Podlovchenko R. I., Rusakov D., Zakharov V. On the equivalence problem for programs with mode switching // Conf. on the Implementation and Application of Automata. — Springer, 2006. — (Lect. Notes Comput. Sci.; Vol. 3845). — P. 351—352.

[35] Rutledge J. D. On Ianov's program schemata // J. ACM. — 1964. — Vol. 11, no. 1. — P. 1—9.

[36] Sampson T. A virus in info-space: the open network and its enemies // J. Media Culture. — 2004. — Vol. 7, no. 3.

[37] Spinellis D. Reliable identification of bounded-length viruses is NP-complete // IEEE Trans. Inform. Theory. — 2003. — Vol. 49, no. 1. — P. 280—284.

[38] Szor P., Ferrie P. Hunting for metamorphic // Proc. of the 2001 Virus Bulletin Conference. — 2001. — P. 123—144.

[39] Varnovsky N. P., Zakharov V. A. On the possibility of probably secure obfuscating programs // 5th Conf. «Perspectives of System Informatics». — Springer, 2004. — (Lect. Notes Comput. Sci.; Vol. 2890). — P. 91—102.

[40] Zakharov V. A. An efficient and unified approach to the decidability of equivalence of propositional program schemes // Automata, Languages, and Programming (Proc. of ICALP'98, Aalborg, Denmark, July 13—17, 1998). — Springer, 1998. — (Lect. Notes Comput. Sci.; Vol. 1443). — P. 247—258.

[41] Zakharov V. A. On the decidability of the equivalence problem for monadic recursive programs // Theor. Inform. Appl. — 2000. — Vol. 34, no. 2. — P. 157—171.

[42] Zakharov V. A. The equivalence problem for computational models: decidable and undecidable cases // Machines, Computations, and Universality. — Springer, 2001. — (Lect. Notes Comput. Sci.; Vol. 2055). — P. 133—153.

[43] Zakharov V. A., Zakharyaschev I. M. On the equivalence checking problem for a model of programs related with multi-tape automata // Conf. on Implementation and Application of Automata. — Springer, 2005. — (Lect. Notes Comput. Sci.; Vol. 3317). — P. 293—305.