ИСП Институт Системного Программирования
Российской Академии наук

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

Труды Института Системного Программирования РАН

Proceedings of the Institute for System Programming of the RAS

Том 26, выпуск 3

Volume 26, issue 3

Москва 2014

ИСП **Российская Академия наук**
Институт Системного Программирования

Труды Института Системного Программирования

Том 26
выпуск 3

Под редакцией
академика РАН В.П. Иванникова

Москва 2014

УДК004.45

Труды Института системного программирования: Том 26, выпуск 3.
/Под ред. Академика РАН В.П. Иванникова/ – М.: ИСП РАН, 2014.

В этом выпуске Трудов Института системного программирования РАН публикуются статьи, написанные сотрудниками ИСП РАН и других организаций. В этих статьях описываются результаты исследований, выполненных в 2014 г.

ISSN 2079-8156 (Print)

© Институт Системного Программирования РАН, 2014

Содержание

Современные модели и методы теории расписаний

А. С. Аничкин
<anton.anichkin@ispras.ru>
В. А. Семенов
<sem@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. Статья посвящена важной проблеме систематизации и концептуализации теории расписаний, которая находит применение в таких предметных областях как управление производством, организация транспортных потоков, планирование проектов, управление ресурсами в вычислительных системах. Однако разнообразие математических моделей и методов составления расписаний обычно ставит перед прикладными математиками и программистами проблему построения быстрого алгоритма, а также его эффективной программой реализации с учетом особенностей решаемой задачи. Использование типовых решателей в составе математических библиотек общего назначения для подобных целей крайне ограниченно. Более перспективным для реализации программных приложений близкой функциональности представляется использование объектно-ориентированных каркасов. В статье предпринимается попытка систематизировать и обобщить модели и методы теории расписания с целью построения подобного каркаса. Главное внимание при этом уделяется задачам ресурсного планирования проектов, которые, с одной стороны, находят широкое практическое применение, а с другой стороны, — обобщают математические постановки, возникающие в смежных предметных областях и дисциплинах.

Ключевые слова: теория расписаний, календарно-сетевое планирование, программная инженерия, объектно-ориентированное программирование

1. Введение

Статья посвящена актуальной теме систематизации и концептуализации теории расписаний. Теория расписаний находит применение в таких предметных областях как управление производством, организация транспортных потоков, планирование проектов, управление ресурсами в вычислительных системах. Однако разнообразие математических моделей и методов составления расписаний обычно ставит перед прикладными математиками и программистами неизбежную проблему построения быстрых алгоритмов и их эффективной программой реализации с учетом особенностей

решаемой задачи. Использование типовых решателей в составе математических библиотек общего назначения для подобных целей крайне ограниченно и неэффективно. Более перспективным для разработки приложений близкой функциональности представляется использование объектно-ориентированных каркасов (object-oriented framework). Организация каркаса в виде системы абстрактных и конкретных классов вместе с предусмотренными механизмами их взаимодействия обеспечивает надлежащую степень общности и гибкости инструментальной среды, необходимую для разработки специализированных приложений. Построению каркаса, как и любого программного приложения, базируемого на объектной парадигме, предшествует этап анализа и моделирования предметной области.

В настоящей статье предпринимается попытка систематизировать модели и методы теории расписания с целью проведения их объектного анализа и построения универсального каркаса для реализации программных приложений. Статья представлена в виде обзора работ в области теории расписаний. При этом главное внимание уделяется задачам и методам ресурсного планирования проектов (Resource Constrained Project Scheduling Problem или, сокращенно, RCPSP), которые, с одной стороны, находят широкое практическое применение, а с другой стороны, — обобщают математические постановки, возникающие в смежных предметных областях и дисциплинах.

Задачи теории расписаний обычно формулируются как задачи оптимизации обслуживания конечного множества требований в системе, содержащей ограниченное число машин. Для каждого требования указывается время обработки на каждой машине, порядок обслуживания и сроки выполнения. Традиционно задачи теории расписания делят на четыре основных класса:

- постановка «открытая линия» (open shop) предполагает многостадийное выполнение каждого требования на заданном подмножестве машин в произвольном порядке;
- постановка «рабочий цех» (job shop) устанавливает для каждого требования строгий порядок выполнения на заданном подмножестве машин;
- постановка «потоковая линия» (flow shop) фиксирует порядок использования машин и предполагает последовательное многостадийное выполнение каждого требования на каждой машине в установленном порядке; решением задачи является последовательность требований, при которой минимизируется общее время обслуживания;
- постановка с директивными сроками (release dates) предполагает задание для каждого требования времени обслуживания, а также директивных сроков его поступления и окончания. В отличие от многостадийных постановок, требования в данном классе задач могут

выполняться на одной машине. Поэтому обычно допускаются прерывания и произвольный порядок обслуживания требований. При наличии нескольких расписаний, удовлетворяющих предписанным директивным срокам, в качестве решения выбирается расписание с минимальным общим количеством прерываний.

Большинство практически содержательных задач составления расписаний допускают произвольные комбинации ограничений и дисциплин обслуживания и являются NP-полными задачами. Лишь немногие постановки с частными условиями могут быть решены за полиномиальное время. Например, расписание с прерываниями, удовлетворяющее директивным срокам при произвольном числе машин, может быть построено за $O(n^3)$ [[1]]. Для задач в постановке «потоковая линия» расписание для двух машин с минимальным общим временем обслуживания может быть составлено за $O(n \log(n))$ [[1]]. Тем самым, вычислительная сложность задач может существенно варьироваться в зависимости от конкретных условий. Развернутые обзоры задач теории расписаний с анализом их вычислительной сложности можно найти в ряде источников [[2], [3]].

Примечательно, что постановки «открытая линия», «рабочий цех» и «потоковая линия» являются частными случаями задач ресурсного планирования проектов [[4]]. Более того, известные задачи об упаковке в контейнеры (линейная, двумерная упаковка) [[5]], разнообразные постановки «О рюкзаке» (упаковки по стоимости и весу) [[6]], классическая задача «О коммивояжере», а также задачи составления расписаний в учебных заведениях могут формулироваться и решаться как задачи ресурсного планирования RCPSP [[7], [8], [9], [10]]. Тем самым, обсуждаемый класс задач приобретает особое значение в контексте проводимой систематизации и концептуализации теории расписаний, а также в связи с построением универсального объектно-ориентированного каркаса для разработки программных приложений.

Поскольку в теории расписаний нет единой принятой терминологии, в дальнейшем мы будем применять понятия ресурсного планирования. Вместо «требование», «активность», «процесс» или «операция» будем употреблять термин «работа». Вместо «машина» и/или «станок» будем использовать понятие «обобщенный ресурс», который охватывает как возобновляемые ресурсы (например, комплект оборудования или штат сотрудников), так и не возобновляемые ресурсы (связанные, например, с материальными и финансовыми затратами). Классическая RCPSP-задача ставится как задача минимизации общего времени выполнения всего проекта при соблюдении временных отношений между работами и не нарушении условий доступности ресурсов. Математическая формализация классической задачи была проведена Притскером [[11]], а её NP-полнота доказана Блазевицем [[12]].

Для решения классической RCPSP-задачи были разработаны точные и приближенные методы [[13], [14], [15]]. Первую группу составляют метод прямого перебора, метод «ветвей и границ» [[16]], методы линейного

программирования [[17]], метод динамического программирования [[18]] и метод декомпозиции [[19]]. Методы обеспечивают поиск оптимального расписания, но в силу высокой вычислительной сложности применимы лишь к небольшим проектам. Приближенные методы, такие как метод Монте-Карло [[20]], метод частичного перебора [[21]], метод направленного перебора [[21]], упрощенный метод «ветвей и границ» [[16]], а также современные методы последовательного и параллельного составления расписаний на основе эвристических правил [[22]] позволяют генерировать эффективные расписания для масштабных проектов за разумное время [[15], [23]].

С релаксацией ресурсных ограничений вычислительная сложность RCPSP-задачи может быть существенно понижена. Так, при отсутствии ресурсных ограничений задача вовсе сводится к задаче поиска наидлиннейшего пути в плане, которая решается методом критических путей (Critical Path Method или, сокращенно, CPM) за линейное время $O(n)$. Кроме упомянутой классической задачи опубликовано значительное число работ, посвященных частным и обобщенным постановкам ресурсного планирования. Главным образом они отличаются целевыми функциями, способами исполнения работ, типами временных или иных ограничений, а также моделями ресурсов. Немногие из этих задач получили практическое распространение и лишь несколько методов составления расписаний для RCPSP-задач реализованы в составе современных программных систем управления проектами. Причина, видимо, заключается в сложности поддержки многовариантных постановок и трудности обобщенной программной реализации методов планирования, рассчитанной на широкие классы задач.

В разделе 2 мы рассматриваем классическую постановку RCPSP-задач, а также уточняем роль нотации Грэхэма [[24]] и правил Брюкера [[25]] в систематизации и классификации задач теории расписания. Раздел 3 посвящен анализу моделей ресурсов, применяемых в задачах планирования проектов. Выделение классов возобновляемых, не возобновляемых, ограничено-возобновляемых, частично возобновляемых, эксклюзивных, логистических, непрерывно разделяемых ресурсов и ресурсов с переменной доступностью позволяет охватить наиболее содержательные случаи. В разделе 4 детально обсуждаются особенности моделей исполнения работ. Особое внимание уделяется работам с прерываниями, альтернативным режимам исполнения работ, профилям использования ресурсов, учету накладных расходов, а также обеспечению компромиссов. Важные факторы календарно-сетевого планирования, включая основные виды временных ограничений, рассматриваются в разделе 5. Проводимый анализ охватывает отношения предшествования с минимальными и максимальными лагами, явные временные ограничения, ограничения рабочего времени и логические зависимости между работами. Наконец, в заключительном 6 разделе обсуждается выбор целевой функции, необходимой для корректной постановки соответствующей оптимизационной задачи Минимизация

временных показателей проекта, устойчивость к задержкам, обеспечение консервативности расписания, минимизация затрат на возобновляемые и не возобновляемые ресурсы, минимизация общей стоимости проекта, максимизация чистой приведенной стоимости могут применяться в качестве критериев поиска оптимального расписания.

В заключении мы подводим итог проведенному анализу современных моделей и методов теории расписаний, а также выделяем основные категории объектов, которые могут быть положены в основу универсального объектно-ориентированного каркаса для разработки программных приложений.

2. Классическая постановка RCPSP-задачи

Классическую постановку RCPSP-задачи составления расписания можно формализовать следующим образом. Исходными данными является проект, состоящий из J работ. Каждая работа имеет свой номер (индекс) $j = 1,2,\dots,J$. Кроме того каждая работа j характеризуется временем своего выполнения (продолжительностью) p_j, которое может быть нулевым ($p_j \geq 0$). На работу могут накладываться технологические ограничения, связанные с невозможностью начать её выполнение ранее, чем завершатся одна или более связанные с ней предыдущие работы, именуемые предшественниками. Данная работа по отношению к своим предшественникам является последователем. P_j является множеством, содержащем всех предшественников работы j. Взаимозависимости между работами не должны иметь циклический характер.

Кроме работ в проекте может присутствовать K возобновляемых ресурсов. Максимально доступное количество каждого ресурса $k = 1,2,\dots,K$ ограничено константой R_k. Каждая работа j может требовать для своего выполнения некоторое количество r_{jk} ресурса k. Причём r_{jk} не может превышать предел доступности R_k ресурса k. Потребляемых работой ресурсов может быть несколько. Потребление ресурса работой означает, что с началом выполнения работы j ресурс k в количестве r_{jk} считается занятым, то есть не доступным для выполнения других работ, а с окончанием выполнения работы данное количество ресурса высвобождается. Таким образом потребление ресурса носит равномерных характер в течение всего времени выполнения работы. Работы с нулевой продолжительностью не требуют для своего выполнения ресурсов, так как время захвата ресурсов совпадает с временем их высвобождения.

Работа не может быть прервана, то есть если работа была начата, то она не может приостановиться и временно высвободить все используемые ею ресурсы. Для упрощения вычисления временных рамок выполнения всего проекта вводится две дополнительные фиктивные работы с нулевой продолжительностью ($j = 0$ и $j = J + 1$), соответствующие началу и завершению всего проекта. Первая работа ($j = 0$) становиться предшественником для всех работ, ранее не имевших предшественников.

Последняя ($j = J + 1$) – последователем для всех работ, ранее не имевших последователей.

Все данные считаются известными и детерминированными. Все параметры и константы являются целочисленными и неотрицательными. Неизвестными считаются время S_j и время C_j начала и завершения работы $j = 0,1, \dots, J, J + 1$ соответственно, причём $C_j = S_j + p_j$.

Классическая RCPSP задача построения расписания сводиться к поиску (вычислению) всех значений S_j и/или C_j таких, чтобы время выполнение всего проекта $C_{MAX} = C_{J+1}$ было минимальным из всех возможных. В терминах общепринятой нотации Грэхема [[24]] данная задача обозначается как $PS|prec|C_{MAX}$.

Нотация Грэхема для обозначения классов задач теории расписания представляет собой комбинацию трёх характеристик $\alpha|\beta|\gamma$. Первая характеристика α может задаваться только одним значением, однозначно определяющим модель ресурсов. Вторая характеристика β описывает используемую модель исполнения работ. Данная характеристика может быть представлена одним или несколькими значениями или отсутствовать вовсе. Третья характеристика γ определяет целевую функцию, минимизация которой и является задачей составления оптимального расписания. Сама целевая функция может быть как простой, так и составной. Задание данной характеристики является обязательным, поскольку именно она определяет стратегию поиска решения и качество полученного результата.

3. Модели ресурсов

3.1. Не возобновляемые и ограничено-возобновляемые ресурсы

В классической постановке RCPSP-задачи рассматриваются только, так называемые, возобновляемые ресурсы, которые доступны в любой момент времени в фиксированном количестве. Однако часто рассматривают три вида ресурсов: возобновляемые (renewable), не возобновляемые (nonrenewable) и ограничено-возобновляемые (doubly constrained). Такая классификация впервые была предложена в [[26], [27]].

Доступность возобновляемых ресурсов, таких как рабочие или машины, определяется в каждый момент времени. Ограничения, связанные с не возобновляемыми ресурсами, например, с бюджетным планом, распространяются на весь проект. Классическая RCPSP-задача предусматривает задание только возобновляемых ресурсов. Не возобновляемые ресурсы могут учитываться, например, в рамках мультимодальной постановки (см. раздел 0). В классической постановке сумма потраченных не возобновляемых ресурсов всегда будет одинаковой вне зависимости от построенного расписания.

Как правило, при планировании учитываются как возобновляемые, так и не возобновляемые ресурсы. В терминах $\alpha|\beta|\gamma$ нотации данная ресурсная модель обозначается как $\alpha = MPS; R; N$. Это обозначение несколько отличается от оригинального, приведённого Брюкером в публикации [[25]]. Однако данное обозначение удачно отражает комбинированный характер ресурсной модели.

Ограниченно-возобновляемые ресурсы сочетают в себе особенности как возобновляемых, так и не возобновляемых ресурсов. Ограничения, заданные для них, содержательны как для определенных моментов времени, так и на протяжении всего периода выполнения. К ресурсам данного вида следует отнести рабочего, который не может быть задействован в более чем в пяти работах за весь период выполнения проекта. Однако, такая двойственность позволяет интерпретировать ограничено-возобновляемые ресурсы как простую комбинацию возобновляемых и не возобновляемых ресурсов, избегая выделение самостоятельного вида.

3.2. Частично возобновляемые ресурсы

Понятие частично возобновляемого ресурса (partially renewable resource) впервые было введено в [[28]]. Суть данной ресурсной модели заключается в следующем. Для каждого частично возобновляемого ресурса k имеется множество Π_k, представленное временными подмножествами Q_{ki}. Считается, что ресурс k доступен в количестве $R_k^{\Pi}(Q_{ki})$ в течение времени, определённого подмножеством $Q_{ki} \subseteq \{1, \dots, T\}$, где $Q_{ki} \in \Pi_k$ и T – условное время завершения проекта. Потребление ресурса k работой j по-прежнему представляет собой константное значение r_{jk}. В более общем случае потребление частично возобновляемого ресурса определяется для каждого подмножества Q_{ki}. Другими словами, данный вид ресурсов позволяет определить несколько альтернативный сценариев их потребления и доступности. Примером является обычный рабочий, который может работать либо каждый день с понедельника по пятницу, либо только в выходные, но не все семь дней. Π_k в этом случае будет состоять из двух подмножеств: с понедельника до пятницы (Q_{k1}) и с субботы до воскресения (Q_{k2}). При этом $R_k^{\Pi}(Q_{k1}) = R_k^{\Pi}(Q_{k2}) = 1$.

Авторы публикации [[29]] рассмотрели данную ресурсную модель в рамках мультимодальной постановки RCPSP-задачи (см. раздел 0). В публикации [[30]] авторами был рассмотрен частный случай данной ресурсной модели, когда подмножества Q_{ki} представляют собой временные интервалы $[t_1, t_2]$. Такой подход представляется более естественным, поскольку ресурсы обычно доступны на некотором промежутке времени, пусть даже и очень коротком.

Для обозначения ресурсного планирования с частично возобновляемыми ресурсами обычно использую $\alpha = PS; PR$.

3.3. Логистические ресурсы

Понятие логистического (или кумулятивного) ресурса (cumulative resource) было введено в [[31]], в где оно рассматривалось в сочетании с минимальными и максимальными временными лагами (см. разделы 0 и 0) в рамках RCPSP-задачи. Данный вид ресурсов позволяет учитывать процессы производства, связанные с временным размещением промежуточных результатов деятельности на некоторой площади, в резервуаре или таре, которые в этом случае следует рассматривать в качестве самостоятельных логистических ресурсов. Следует отметить, что логистические ресурсы могут определяться как в контексте возобновляемых, так и не возобновляемых ресурсов.

Логистический ресурс характеризуется своей номинальной ёмкостью R_k и некоторым её запасом $\overline{R_k}$. Каждая работа j может производить или потреблять (перерабатывать) продукцию в объёме r_{jk} ($r_{jk} > 0$ в случае производства и $r_{jk} < 0$ в случае потребления). Использование подобной модели обозначается как $\alpha = PS; Cu$.

В публикации [[32]] логистические ресурсы рассматривались в рамках мультимодальной постановки RCPSP-задачи (см. раздел 0) применительно к процессам разработки и тестирования в автомобильной промышленности. Тестовый автомобиль моделировался в качестве логистического ресурса, который сначала производился (занимал некоторый объём), потом использовался (занимаемый объём не изменялся), а затем уничтожался в краш-тесте (объём освобождался). В публикациях [[33], [34]] данные ресурсы применялись для планирования серийного производства в обрабатывающей промышленности.

3.4. Непрерывно разделяемые ресурсы

В классической постановке доступность ресурсов рассчитывается дискретным образом. В [[35]] авторами рассматривается возможность их произвольного непрерывного исчисления и разделения (continuous resource). Это вполне естественно для проектов с ресурсами, представляющими собой, например, энергетические источники или какие-либо жидкие расходные материалы. Примечательно, что данная возможность применима как к возобновляемым, так и не возобновляемым ресурсам [[36]]. В нотации $\alpha|\beta|\gamma$ использование данного ресурсного концепта обозначается как $\alpha = PS; Co$.

Более сложные постановки, связанные с непрерывно разделяемыми ресурсами, рассматривались в [[37], [38], [39], [40]]. Следует отметить публикацию [[41]], в которой фактор непрерывности применялся для ограничено возобновляемых ресурсов.

3.5. Эксклюзивные ресурсы

Эксклюзивными называются ресурсы (dedicated resource), которые в один момент времени могут быть использованы только в одной работе [[42], [43]]. Такие ресурсы могут моделироваться возобновляемыми ресурсами с лимитом использования $R_k = 1$ в любой момент времени. Поэтому постановки с подобными ресурсами являются частными случаями RCPSP-задачи. Авторы работ [[44], [45]] использовали дизъюнктивную функцию планирования для интерпретации RCPSP-задач с эксклюзивными ресурсами. В $\alpha|\beta|\gamma$ нотации данный вид ресурсов представляется как $\alpha = PS; Rm, 1, 1$, что говорит о наличие в проекте m возобновляемых ресурсов с доступным количеством 1 и использованием в работах в количестве 1.

3.6. Ресурсы с переменной доступностью

В классической постановке уровень доступности возобновляемых ресурсов фиксирован на протяжении всего проекта. Подобное допущение не всегда выполняется на практике. Например, в реальных проектах количество доступных специалистов может варьироваться в связи с периодическими отпусками, занятостью в других работах и т.п.

Чтобы учитывать данный фактор, вводится понятие доступности возобновляемого ресурса k в момент времени t как функции времени $R_k(t)$. Ресурсы с переменной доступностью рассматривались в работах [[33], [46], [47], [48], [49], [50], [51]]. Часто данная модель применяется в задачах планирования работ с прерываниями (см. раздел 0). В [[52]] обсуждаются аспекты применения данной модели в медицинских исследовательских проектах для контроля доступности исследовательского персонала и лабораторного оборудования.

В нотации $\alpha|\beta|\gamma$ переменная доступность ресурсов отражается как $\alpha = PS; Rm; *,*$, что означает наличие в проекте m возобновляемых ресурсов с переменной доступностью и произвольными уровнями потребления в работах.

Авторы публикации [[9]] ввели понятие дизъюнктивных ресурсов с доступностью, изменяемой во времени от 0 до 1. Заметим, что этот случай соответствует ограниченной функции $R_t(t) \in \{0, 1\}$ и является вариацией эксклюзивных ресурсов. В работе [[53]] рассматривается задача, в которой доступность ресурсов изменяется только в определённые моменты времени, связанные с вехами проекта (milestone).

В работе [[54]] показано, что задача с ресурсами переменной доступности может быть редуцирована к задаче с ресурсами фиксированной доступности путем введения минимальных и максимальных лагов. Для этого предлагается заменить переменную доступность $R_k(t)$ ресурса k на постоянную, соответствующую максимальному значению доступности ресурса за всё время выполнения проекта ($R_k = \max_{0 \le t \le T} R_k(t)$), а в периоды, когда доступность меньше максимальной, ввести фиктивные работы, компенсирующие избыток

ресурса $(r_{jkt} = R_k - R_k(t))$. Местоположение таких фиктивных работ в расписании предполагается зафиксировать с помощью минимальных и максимальных временных лагов.

Следует отметить, что задачи с ресурсами переменной доступности являются частными случаями RCPSP-задач с частично возобновляемыми ресурсами.

4. Модели исполнения работ

4.1. Работы с прерываниями

Классическая постановка не допускает прерываний уже начатых работ. Однако рядом исследователей рассматривались работы с прерываниями (preemptive scheduling), которые могут происходить в дискретные моменты времени, обычно, кратные точности представления временных интервалов [[9], [30], [55], [56]]. В нотации Брюкера такие задачи обозначаются как $prmt$ в поле β [[25]]. Изучалась также модель выполнения, допускающая фиксированное число прерываний, не превышающее для каждой индивидуальной работы некоторый заданный порог [[57]]. В [[58], [59]] предлагается определить первоначальный интервал, на протяжении которого работы не могут прерываться. Менее содержательной, на наш взгляд, выглядит модель, описанная в работе [[55]] и устанавливающая возможность параллельного исполнения отдельных стадий работы, сформированных в результате прерываний (fast tracking). Довольно часто прерывания работ интерпретируются в терминах календарей, которые устанавливают допустимые интервалы для проведения работ [[33], [60]]. Однако календари обычно учитываются при вычислении дат начала и завершения работ с учетом выходных и рабочих дней. Поэтому их использование для моделирования прерываний кажется искусственным.

4.2. Профильное использование ресурсов

В традиционной постановке предполагается, что каждая работа j может требовать для своего выполнения некоторое фиксированное количество r_{jk} ресурса k, которое не может изменяться в ходе выполнения. Данная ресурсная модель естественным образом обобщается путем введения профиля r_{jkt} для каждого t -ого интервала выполнения работы j при потреблении ресурса k. Так называемые, зависимые от времени ресурсы (time-dependent resources) находят применение во многих приложениях [[52], [61], [62]]. Тем не менее, существует довольно элегантный прием смоделировать эту ситуацию путем представления работы эквивалентной цепочкой подзадач с фиксированными уровнями потребления ресурса [[54]]. Чтобы избежать наложения или прерывания подзадач по времени, для каждой пары соседей i и $i + 1$ в цепочке устанавливаются минимальные и максимальные лаги с нулевой задержкой (см. разделы 0 и 0).

4.3. Учет накладных временных затрат

Другим направлением в развитии модели исполнения работ является учёт времени подготовки используемых работой ресурсов (setup times), например, для переналадки оборудования. В [[63]] рассматривается три альтернативы, выражающие характер зависимости времени подготовки от последовательности работ или всего плана работ, в которых задействован ресурс. Соответственно, различают времена подготовки, независимые от последовательности (sequence-independent setup times), зависимые от последовательности (sequence-dependent setup times) и зависимые от расписания (schedule-dependent setup times). Первые учитывают зависимость времени подготовки ресурса от него самого и работы, в котором он участвует. Вторые принимают во внимание и предшествующую работу, использующую тот же ресурс. Наконец, третьи учитывают характер использования ресурсов всеми предшественниками. В нотации Брюкера [[25]] независимые от последовательности и зависимые от последовательности времена обозначаются в поле β как s_j и s_{ij} соответственно.

В [[64]] обсуждаются иные временные факторы, в частности, рассматриваются факторы перенастройки и/или отключения/консервации/вывода из эксплуатации ресурса (removal times). Хотя временные факторы подготовки ресурсов обычно рассматриваются как самостоятельные концептуальные элементы, часто для их представления и учета используется мультимодальность исполнения работ (см. раздел 0) [[33], [65]] или механизмы прерываний и параллельного исполнения работ в сочетании с мультимодальностью [[66]].

4.4. Альтернативные режимы исполнения работ

В традиционной постановке каждая работа выполняется единственным способом, однозначно определяемым заданной продолжительностью и требуемым количеством ресурсов. Начиная с пионерской работы Эльмагхраби [[67]], концепция работ была существенно пересмотрена в сторону поддержки альтернативных (или мультимодальных) режимов M_j для каждой индивидуальной работы j. Каждый режим m отражает один из реализуемых способов исполнить работу за фиксированный период p_{jm} при фиксированном количестве r_{jmk} ресурса k. Изменения режима в ходе выполнения работы и во время её прерывания не допускаются. Класс задач планирования, допускающих мультимодальное исполнение работ, называют Multimodal Resource Constrained Project Scheduling Problem или, сокращенно, MRCPSP. Решением задачи MRCPSP является расписание, которое включает в себя не только даты начала работ S_j, но и режимы m_j, в котором следует выполнять работы. В нотации Брюкера данный класс задач обозначается как MPS в соответствующем поле α.

Задачи MRCPSP с возобновляемыми ресурсами активно изучались в последнее время [[49], [68], [69], [70], [71], [72], [73]]. Постановки MRCPSP с обобщенными отношениями предшествования (лагами) рассмотрены в публикациях [[9], [48], [65], [74], [75], [76], [77]], а постановки с обобщенными ресурсными ограничениями — в [[29]]. В публикациях [[78], [79], [80]] вводится и обосновывается понятие качества мультимодального расписания, как некоторой интегральной характеристики, отражающей важность и приоритетность использования одних режимов над другими. В публикациях [[65], [81]] рассматриваются группы работ, на которые накладываются условия их согласованного выполнения в одних и тех же режимах (mode identity constraints).

4.5. Учет компромиссов

В RCPSP-задачах выделяют два рода компромиссов: компромисс между временем исполнения всего проекта и уровнями потребления ресурсов, а также компромисс между временем исполнения проекта и его стоимостью.

В задаче поиска компромиссного расписания первого рода вводится понятие нагрузки φ_j для работы j. Работа может быть выполнена при любой дискретной комбинации времени выполнения p_j и потребления ресурса r_{jk}, такой что $p_j r_{jk} \geq \varphi_j$. При этом содержательными являются комбинации минимальных допустимых пар значений, для которых имеет место $(p_j - 1)\, r_{jk} < \varphi_j$ и $p_j(r_{jk} - 1) < \varphi_j$. Задача обеспечения компромисса первого рода для возобновляемых ресурсов исследовалась в публикациях [[82], [83], [84]].

В задаче поиска компромиссного расписания второго рода во внимание принимается единственный не возобновляемый ресурс, который играет роль бюджета проекта. При этом ставится задача минимизации времени выполнения проекта при фиксированном уровне бюджета, а также двойственная ей задача минимизации бюджета проекта при заданном предельном сроке его окончания (deadline) [[85], [86]].

Примечательно, что обе постановки могут рассматриваться как частные случаи задачи MRCPSP с альтернативным исполнением работ. Единственное отличие состоит в том, что в задачах поиска компромиссных расписаний вместо допустимых комбинаций значений времени выполнения работы и потребляемыми ею ресурсами явно оперируют с нагрузками. Иногда, чтобы подчеркнуть этот нюанс, для описания этого класса задач уточняют нотацию $\alpha = T_{tr}PS$. В постановке $\alpha = T_{tc}PS$ при заданном крайнем сроке окончания проекта минимизируется бюджет, причем функция стоимости имеет квадратичную зависимость от задержки фактических сроков выполнения работ [[87]].

5. Временные ограничения

5.1. Предшествования с минимальными лагами

В классической постановке работа не может стартовать пока все предшественники не завершат выполнение. Понятие простого предшествования обычно расширяется введением минимального временного лага d_{ij}^{FS}, определяющее ограничение на возможный старт последователя j относительно времени завершения предшественника i как $C_i + d_{ij}^{FS} \leq S_j$. Отрицательные значения лагов допускаются, означая, что работы могут выполняться с наложением временных интервалов. Часто в приложениях рассматривается сразу четыре вида временных лагов d_{ij}^{FS}, d_{ij}^{SS}, d_{ij}^{SF}, d_{ij}^{FF}, связывающие времена старта или завершения предшественника со временем старта или завершения последователя. При условии фиксированной продолжительности планируемых работ данные ограничения легко трансформируются друг в друга [[54]], однако при наличии альтернативных сценариев выполнения работ и актуализации проектного плана поддержка ограничений всех видов является необходимой.

Для данных видов ограничений используется значение $temp$ для поля β в нотации Брюкера [[25]]. Ограничения данного вида рассматриваются в многочисленных публикациях [[47], [50], [51], [88], [89], [90], [91]], однако в публикации [[92]] было отмечено, что данные ограничения могут использоваться для моделирования процессов настройки ресурсов и загрузки производственных линий (sequence-independent setup times).

5.2. Предшествования с максимальными лагами

Аналогично отношениям предшествования с минимальным лагом вводятся временные ограничения с максимальным лагом. Для предшественника i и последователя j данный вид отношений выражается следующим образом: $C_i + d'^{FS}_{ij} \geq S_j$. Задание ограничения данного вида означает, что последующая работа должна стартовать не позже, чем через d'^{FS}_{ij} временных единиц после завершения предшествующей работы. Заметим, что одновременное задание нулевого минимального лага и нулевого максимального лага означает, что последователь должен стартовать сразу после завершения предшественника. Комбинация времен начала и завершения работ для отношения предшествования приводит к четырем видам максимальных лагов d'^{FS}_{ij}, d'^{SS}_{ij}, d'^{SF}_{ij}, d'^{FF}_{ij}, аналогичных ранее рассмотренных. Класс задач планирования, в которых допускается одновременное использование минимальных и максимальных лагов часто обозначается как $RCPSP/max$. Он был рассмотрен в ряде публикаций [[33], [54], [93], [94], [95], [96], [97], [98]]. В контексте анализа альтернативных сценариев выполнения работ он изучался также в публикациях [[9], [74], [75], [76], [77]]. А в [[99]] для них предложена

довольно интересная модель лагов, учитывающая режимы выполнения предшественника m_i и последователя m_j. Однако, следует отметить, что использование максимальных временных лагов способно привести к циклическим структурам. Этот аспект подробно рассмотрен в публикации [[100]].

5.3. Явные временные ограничения

В расширенной RCPSP постановке могут применяться такие понятия, как директивный срок выполнения работы и крайний срок выполнения работы (deadline). Принципиальная разница между этими понятиями заключается в том, что значение первого носит рекомендательный характер, а второго – обязательный. В первом случае работу желательно выполнить к назначенному сроку. Причём опережение или запаздывание может «наказываться» штрафом. Во втором случае работа обязана завершиться до заданного крайнего срока. Нарушение этого условия делает дальнейший поиск расписания бессмысленным. Возможны случаи, когда данное условие не может быть соблюдено ни при каких обстоятельствах. Это лишь означает, что исходная задача планирования поставлена математически некорректно.

Существуют также понятия директивного и самого раннего срока начала выполнения работы. Первое носит рекомендательный характер, второе – строго обязательный. Однако понятие директивного срока начала выполнения работы применяют крайне редко, так как оно однозначно связано с более важным директивным сроком завершения работы. Поэтому в публикациях чаще всего рассматриваются самый ранний срок начала и крайний срок завершения выполнения работы, а также директивный срок завершения работы. Задачам с директивными сроками завершения работ посвящены многочисленные публикации [[98], [101], [102], [103], [104]]. Основное внимание при этом уделяется минимизации штрафов, возникающих из-за нарушения директивных сроков выполнения работ.

Понятия самого раннего и крайнего сроков выполнения работ, а также их применения в задачах планирования подробно рассматриваются в публикациях [[37], [39], [47], [61], [89], [105]]. Отдельного внимания заслуживает работа [[105]], авторы которой рассмотрели, так называемую, кумулятивную задачу построения расписания. Это задача ресурсного планирования, в которой отсутствуют отношения предшествования между работами, однако для каждой работы определены самый ранний срок начала и крайний срок завершения. В результате все работы оказываются «запертыми» в некоторых временных рамках, а иногда и строго фиксируемыми по времени исполнения.

Следует заметить, что задание самого раннего срока начала выполнения работы эквивалентно определению минимального временного лага относительно стартовой вехи проекта, а задание крайнего срока завершения — определению максимального временного лага относительно финальной вехи

проекта. Данное наблюдение позволяет представлять и разрешать временные ограничения в рамках единой дисциплины обработки отношений предшествования.

В нотации $\alpha|\beta|\gamma$ ограничения, накладываемые на начало выполнения работы j, соответствуют обозначению r_j, а накладываемые на завершение работы — d_j.

5.4. Ограничения рабочего времени

Понятие ограниченного рабочего времени было введено в [[106]]. Временная шкала, в соответствии с которой проводится планирование, разбивается на циклически повторяющиеся рабочие и нерабочие временные окна или интервалы. Каждая работа может начать своё выполнение только в рабочее время. Продолжительность выполнения работы не распространяется на нерабочие интервалы. Например, нормированный рабочий день предполагает наличие интервалов с 9 часов утра до 17 часов вечера каждый день, кроме субботы и воскресенья и исключая возможные обеденные перерывы.

В [[9]] отмечена возможность моделирования рабочих интервалов с помощью дополнительного возобновляемого ресурса, лимит доступности которого равен 1 в рабочее время и 0 — в нерабочее. В работе [[107]] авторы применили дисциплину ограничения рабочего времени при компромиссном планировании проекта с соблюдением требований к срокам реализации проекта и его стоимости.

5.5. Иные временные ограничения

В работе [[32]] было введено отношение частичного порядка между работами. Данное отношение применялось в проектах автомобилестроительной отрасли, связанных с проведением инженерных и тестовых работ. Отношение частичного порядка между работами i и j подразумевает, что либо работа i должна быть завершена до начала работы j, либо данные работы должны выполняться в разных режимах (то есть использовать разные ресурсы). Подобное отношение может устанавливаться, если выполнение работы j предполагает уничтожение ресурса, который мог быть использован другой работой i. Например, работы по сборке автомобиля должны предшествовать краш-тесту, в результате которого автомобиль разрушается.

В работе [[52]], посвящённой медицинским исследовательским проектам, применяется отношение между работами, исключающее их завершение за один и тот же временной период. При этом не уточняется, какая работа должна быть завершена первой. В работе [[9]] вводится ограничение параллельности. Данное ограничение предполагает принудительное выполнение двух и более работ параллельно на одном и том же отрезке времени. Там же рассматривается противоположный случай, когда отрезки выполнения работ не могут пересекаться. Кроме того, авторы обобщили

данное условие введением временного лага с момента окончания одной работы до начала другой. Следует отметить, что данное ограничение не может моделироваться отношениями предшествования с минимальными и максимальными лагами, поскольку очерёдность выполнения работ не определёна. Авторы публикации [[9]] также ввели понятие «коридора параллельности» $[l, u]$, который предполагает, что ассоциируемые с ним работы должны выполняться параллельно в течении l временных единиц по крайней мере, но не более, чем в течении u временных единиц.

Представляет несомненный практический интерес также отношение предшествования с лагом, выраженном в процентном соотношении [[39]]. Оно предполагает установку процентного значения лага g_{ij} между работами i и j, и означает, что работа j не может начаться раньше, чем работа i завершится на g_{ij} процентов. При этом процент завершения стартовавшей работы j не может превышать процента завершения работы i. Такой вид временных ограничений важен для проектов, в которых моменты начала работ разделены по времени и не допускается опережение одних работ другими.

В работе [[46]] рассматривается довольно занятное ограничение, которое предполагает, что никакая работа k не может быть запланирована между заданными работами i и j. В работе [[48]] используется близкий вид ограничений, в котором еще учитывается зависимость от возобновляемого ресурса. Работа j, связанная данным ограничением с работой i, не может начаться раньше, чем завершиться работа i, при этом ни одна другая работа k, которая использует некоторый заданный в отношении ресурс r, не может начаться после завершения работы i и до начала работы j. Данный вид ограничений может использоваться, например, для обеспечения гарантии, что никакая посторонняя работа не сможет начаться между вспомогательной работой по установке ресурса и работой, непосредственно использующей данный ресурс.

Временные ограничения на перемещения ресурсов рассматриваются в [[108]]. Согласно предложенной модели перемещение ресурса r из местоположения, в котором данный ресурс использовался при выполнении работы i, в местоположение, в котором он будет использоваться работой j, занимает Δ_{ijr} временных единиц. Данное время может явно задаваться или насчитываться по местоположению проведения работ.

5.6. Логические зависимости

Авторы некоторых публикаций [[109], [110]] предпочитают ассоциировать работы с дополнительной логикой в исполнении последователей. Согласно этой модели работы, заданные в классической постановке, следует интерпретировать как логические операции «И», поскольку все последователи рано или поздно обязаны быть выполнены. Однако рассматриваются также логические операции «ИЛИ» и «Исключающее ИЛИ». Первый тип операций

предполагает, что хотя бы один последователь должен быть выполнен. При втором типе выполняется строго один последователь. По мнению авторов подобная модель может у спешно применяться при моделировании проектов научно-исследовательской и опытно-конструкторской деятельности. Например, после работы «Тестирование прототипа», ассоциируемой с операцией «Исключающее ИЛИ», должна быть выполнена либо работа «Доработка прототипа», либо работа «Начало производства». Выполнение одной работы исключает выполнение другой. При этом решение о том, какая работа будет выполняться, принимается после завершения работы предшественника. Часто в литературе подобные зависимости рассматривают в контексте стохастического подхода [[111]]. Однако, данная модель вполне жизнеспособна и для планирования детерминированных процессов.

В работе [[112]] рассматриваются иные зависимости. Авторы предлагают использовать явные взаимосвязи между двумя работами i и j, отличные от отношений предшествования. Ими могут быть связаны любые работы i и j, в том числе не имеющие общего предшественника или последователя. Изложенная в работе модель предполагает также использование трёх типов логических операций: «И», «ИЛИ» и «Исключающее ИЛИ». Подобная модель использовалась при планировании наземных процессов в аэропорту, таких как заправка топливом, чистка салона, снабжение питанием. Следует заметить, что описанная модель является более гибкой, чем описанные ранее логические зависимости. Модель также обобщается на случай отношений между двумя и более работами.

6. Целевые функции

6.1. Минимизация временных показателей проекта

Целевые функции временных показателей являются наиболее распространёнными, так как чаще всего расписание ориентированно на оптимизацию именно временных характеристик проекта. Самой распространённой целевой функцией является уже упомянутая в классической постановке минимизация времени выполнения всего проекта C_{MAX}.

Однако существуют и пользуются достаточной популярностью и другие целевые функции, оперирующие такими временными показателями, как задержка, запаздывание и опережение. Данные понятия тесно связаны с понятием директивного срока d_j выполнения работы j – временем, не позже которого работа должны быть завершена. Исходя из этого, задержкой работы j является значение $L_j = C_j - d_j$. Запаздывание схоже с задержкой с той лишь разницей, что не может быть отрицательным, то есть запаздывание $T_j = \max\{0, (C_j - d_j)\}$. Аналогично определяется и опережение $E_j = \max\{0, (d_j - C_j)\}$.

В некоторых публикациях [[30], [90], [101], [113]] рассматривается минимизация взвешенного запаздывания. Заметим, что минимизация времени выполнения всего проекта является частным случаем данной целевой функции. В работе [[98]] описывается функция минимизации наибольшей задержки и суммарного взвешенного запаздывания, а в [[114]] обсуждается целевая функция, ориентированная на минимизацию суммы всех взвешенных значений опережения и запаздывания. Такая целевая функция позволяет планировать проект с временем завершения каждой работы предельно близким к соответствующему ей директивному сроку выполнения, то есть «точно в срок». При этом если работу не удаётся выполнить в срок, весовые коэффициенты позволяют нивелировать между опережением и запаздыванием для каждой работы (то есть устанавливать приоритет одного над другим). Иногда минимизации суммы абсолютных значений опережения и запаздывания для каждой работы рассматривается в рамках мультимодальной постановки RCPSP-задачи [[115]]. Альтернативной целевой функцией может быть минимизация максимального из всех значений опережения и запаздывания. В работе [[116]] предлагается вариация предыдущей целевой функции с той лишь разницей, что опережение и запаздывание изменяются относительно некоторого определённого отрезка времени («временного окна»), в течение которого работа должна быть выполнена.

В публикации [[117]] автор предложил определить набор «временных окон» для каждой работы. Исходя из некоторых соображений качества и выгоды, желательно, чтобы работа была выполнена в одно из этих «временных окон». Целевая функция, описанная в работе, ориентирована на минимизацию штрафов, возникающих в случае выполнения работы вне выделенного ей «временного окна». Стимулом для разработки такого подхода послужил один биотехнологический проект. В работе отмечено, что предложенный подход подобен временным ограничениям, налагаемым на работу: «временные окна» могут представлять собой жёсткие временные ограничения.

В работе [[30]] предлагается минимизировать сумму всех времён завершения работ, в то время как в публикации [[118]] предлагается минимизировать их взвешенную сумму. Развитием данных подходов является целевая функция минимизации среднего времени потока [[119]], которое соответствует среднеарифметическому всех времён завершения работ. Заметим, что функции минимизации общего времени завершения и среднего времени завершения являются эквивалентными.

В нотации $\alpha|\beta|\gamma$ описанные выше целевые функции могут быть представленные в характеристике γ следующими символьными обозначениями: в случаях, когда целевой функцией является минимизация выполнения всего проекта, в характеристике указывается C_{MAX}; минимизация наибольшей задержки – L_{MAX}; минимизация общего взвешенного времени завершения – $\sum w_j C_j$; минимизация общего взвешенного запаздывания – $\sum w_j T_j$. Целевые функции могут быть и более сложные. Например, если

необходимо минимизировать общее взвешенное количество запаздывающих работ, то в характеристике γ необходимо задать функцию $\sum w_j U_j$, где $U_j \in \{0,1\}$ является битовым индикатором, который показывает, является ли работа j запаздывающей или нет (если работа запаздывает, то $U_j = 1$; в противном же случае $U_j = 0$). Если, например, необходимо минимизировать общую сумму всех взвешенных опережений и запаздываний, то есть необходимо выполнить каждую работу предельно точно в установленный срок, то γ будет содержать функцию $\sum\left(w_j^E E_j + w_j^T T_j\right)$. Таким образом, как видно, целевые функции временных показателей могут быть достаточно разнообразными и сложными.

6.2. Устойчивость расписания к задержкам

Во время выполнения проекта могут возникать непредвиденные ранее в расписании задержки при выполнении тех или иных работ. Появление таких задержек может привести к дополнительным временным и стоимостным затратам, что может быть крайне не приемлемо в условиях ограниченного бюджета. Поэтому планировщик проекта и ответственный за проект заинтересованы в построении расписания, устойчивого к непредвиденным задержкам, то есть такого расписания, в котором появление таких задержек будет сказываться только на ограниченном количестве соседних работ или не будет сказываться вовсе. Такой подход к планированию обычно называют проактивным. Не смотря на то, что большинство современных подходов, позволяющих решать подобные задачи, работают в рамках стохастического концепта, который позволяет явно задавать некоторую неопределённость, мы в данной статье ограничимся рассмотрением только детерминистических подходов.

Так в публикации [[120]] вводиться понятие свободного временного люфта sl_j работы j. Содержательный смысл данной временной характеристики заключается в том, что при появлении в работе j непредвиденной задержки $L'_j \leq sl_j$ она никак не сказывается на выполнении любой другой работы. В этой же публикации описывается двойная целевая функция («бицелевая функция») минимизации общего времени выполнения всего проекта и максимизации общего свободного временного люфта, который может выступать в роли меры надёжности расписания. Такой же подход описан в работе [[121]]. Иногда на практике используется модифицированный вариант данного подхода: в качестве основной целевой функции иногда предлагается использовать максимизацию наименьшего свободного временного люфта $\min_i sl_i$ [[122]]. При этом для достижения наименьшей продолжительности всего проекта предполагается использование использовать дополнительного временного ограничения в виде крайнего срока выполнения всего проекта (deadline).

Придерживаясь подхода, изложенного в публикации [[120]], предпринималась попытка увеличить количество учитываемых факторов [[123]]. В первую очередь рассматривались кроме свободного временного люфта ещё и суммарное количество всех непосредственных последователей и/или суммарное количество всех используемых ресурсов. Кроме того предлагались варианты данного подхода, в которых вышеперечисленные характеристики использовались вместе с весовыми коэффициентами, а свободный временной люфт заменяется бинарным значением $\alpha_j \in \{0,1\}$, которое выступало в роли индикатора наличия данного люфта (если $sl_j = 0$, то $\alpha_j = 0$; если $sl_j > 0$, то $\alpha_j = 1$). Суть последнего подхода заключается в том, чтобы избежать необъективности, возникающей в случае очень больших значений временных люфтов. Использование больших люфтов не имеет практической значимости, в то время как гораздо меньшего люфта может быть достаточно для компенсации типовых задержек.

В работе [[124]] вводиться понятие переработки (то есть выполнение работы дольше запланированной продолжительности). Значение RT_{jt} является ожидаемым временем переработки (то есть задержкой) для работы j при условии, если работа будет завершена в момент времени t; RC_{jt} – ожидаемые затраты из-за переработки. В соответствии с сущностью данных понятий предполагается, что RT_{jt} и RC_{jt} возрастают со временем. Целевой функцией в данной публикации является минимизация суммы времени и стоимости переработки.

Таким образом использование подобных целевых функций позволяет повысить устойчивость расписания к непредвиденным задержкам и минимизировать затраты связанные с их устранением.

6.3. Обеспечение консервативности расписания

Часто в ходе выполнения проекта возникают непредвиденные ситуации (например, сверхбольшие задержки или перебои с поставками ресурсов), в результате которых дальнейшее выполнение проекта в соответствии с составленным расписанием не представляется возможным. В таких случаях расписание становиться неактуальным и не соответствующим сложившимся условиям. Такое расписание должно быть пересмотрено и перепланировано. Специфика данного перепланирования заключается в том, что проект уже находиться на стадии выполнения: некоторые работы могут быть уже выполнены и должны быть проигнорированы в ходе выполнения перепланирования; некоторые работы начаты, но ещё не закончены – такие работы участвуют в процессе перепланирования, но имеют фиксированные времена начала и ожидаемого завершения своего выполнения. Кроме того при перепланировании может измениться количество и характер доступности ресурсов. В отличие от проактивного планирования, в котором построение устойчивого к задержкам расписания главным образом сводиться к

закладыванию непредвиденных задержек в само расписание, здесь имеет место возникновение непредвиденных событий, выходящих за рамки заложенной надёжности и требующие перестроения всего расписания. Такое планирование обычно называют реактивным.

Основной целевой функцией при перепланировании проекта в большинстве случаев является минимизация вносимых изменений. Этот критерий тесно связан с тем фактом, что ранее представленное расписание, как правило, удовлетворяло все заинтересованные стороны и вынуждало их планировать свою деятельность в соответствии с данным расписанием. Сильно изменённое расписание может привести к дополнительным накладным затратам, задержкам, изменениям в ресурсо-обеспечении или даже к срыву проекта в целом.

В работе [[74]] предлагается минимизировать вносимые в оригинальное расписание изменения путём минимизации количества работ, которые получать в новом расписании новое значение времени начала своего выполнения. А в публикации [[125]] предлагается перепланировать расписание так, чтобы минимизировать сумму всех отклонений нового времени завершения каждой работы от первоначального значения. Данная задача может быть решена в рамках постановки задачи «выполнить точно в срок», где минимизируется взвешенная сумма всех опережений и запаздываний каждой работы. Для редуцирования задачи необходимо оригинальные значения времени завершения каждой работы рассматривать в качестве соответствующих директивных сроков их выполнения. Иногда имеет смысл измерять и минимизировать степень нарушения расписания, суммируя все отклонения значений начала и завершения каждой из работ [[126]].

Отдельно стоит отметить особенности перепланирования проектов, выполняемых в рамках мультимодальной постановки RCPSP-задачи [[127]]. Обычно в таких постановках принято штрафовать все изменения количества и интенсивности использования ресурсов. Кроме того штрафовать предлагается и изменения режимов выполнения каждой из работ. Некоторые особо критические работы, могут быть зафиксированы с помощью введения дополнительных ограничений. Это может быть очень удобно и полезно в случаях с работами, запланированными в непосредственной близости от того момента времени, в который возникла необходимость в перепланировании проекта, то есть работами, которые должны вот-вот начать своё выполнение. Часто на практике вносить изменения в такие работы и, например, откладывать их выполнение не целесообразно. Кроме того данный подход может применяться в случаях, когда необходимо, чтобы новое составленное расписание в какой-то определённый момент времени догнало первоначальное расписание (восстановилось).

6.4. Минимизация затрат на возобновляемые ресурсы

В современной тематической литературе достаточно широко обсуждаются различные целевые функции, связанные с возобновляемыми ресурсами. Наиболее обширный и подробный обзор данных функция приведён в публикации [[128]]. Мы рассмотрим только наиболее популярные и применяемые функции.

Одной из самых распространённых является функция, ориентированная на решение так называемой задачи ресурсных инвестиций. Данная задача является зеркальной копией классической RCPSP-задачи. Так в классической постановке RCPSP-задачи основным является минимизация общего времени выполнения всего проекта, не превышая при этом заданного лимита использования каждого из возобновляемых ресурсов. В данной же задаче основным является минимизация количества используемых возобновляемых ресурсов, не превышая при этом некоторый заданный крайний срок выполнения всего проекта (deadline). Целевой функцией в данной задаче будет минимизация суммы стоимостных затрат на поддержание доступности требуемого количества каждого ресурса, то есть $\sum_k C_k(R_k)$, где R_k – выделенное количество ресурса k (квота), $C_k(R_k)$ – стоимость выделения ресурса k в количестве R_k. Очевидно, что $C_k(R_k)$ является неубывающей функцией, прямо зависящей от размера выделяемой квоты. В частном и наиболее распространённом случае данная стоимостная функция является линейной, то есть $C_k(R_k) = c_k R_k$, где c_k – стоимость поддержания одной единицы ресурса k в доступном состоянии.

Задача ресурсных инвестиция были успешно решена в ряде работ [[97], [98], [129], [130], [131]]. Стоит отметить работу [[132]] в которой представлена расширенная задача ресурсных инвестиций: вместо крайнего срока завершения проекта (deadline) в ней используется директивный срок, что допускает появления задержки завершения всего проекта. Целевой функцией является минимизация общей суммы стоимостных затрат на поддержание ресурсов и на штраф за запаздывание. Штраф за запаздывание проекта прямо пропорционален величине запаздывания.

Иногда основное внимание уделяется арендуемым возобновляемым ресурсам [[133]]. Основная особенность в данном случае заключается в том, что возобновляемый ресурс берётся в аренду, стоимость которой прямо зависит от количества арендуемого ресурса и периода аренды. Стоимость аренды на период t одной единицы ресурса k складывается из фиксированной стоимости c_k^f за факт аренды единицы ресурса и из стоимости аренды c_k^v единица ресурса за единицу времени (ставка аренды). Таким образом, если необходимо взять в аренду r_k единиц ресурса k на время t, то стоимость аренды составит $r_k\left(c_k^f + c_k^v t\right)$. Очевидно, что целевой функцией в данной постановке является минимизация суммы всех стоимостных затрат на аренду ресурсов. Как можно заметить, если ставка аренды будет нулевой ($c_k^v = 0$), то данная задача

сводиться к задаче ресурсных инвестиций. Более подробно данная задача рассмотрена в работе [[134]].

Другой не менее важной задачей является выравнивание загруженности ресурсов. Суть данной задачи сводиться к минимизации изменений интенсивности использования ресурсов при переходе от одного момента времени к другому соседнему, не превышая при этом крайнего срока выполнения проекта (deadline). Если интенсивность использование ресурсов представить в виде графика, то целью данной задачи является предельное сглаживание данного графика. В работах [[30], [97], [98]] для решения данной задачи в роли целевой функции выступила минимизация наибольшего изменения или сумма всех изменений интенсивности использования ресурсов, а в работе [[135]] предлагается минимизировать для этой цели сумму всех изменений, возведённых в квадрат.

В некоторых случаях полезно минимизировать используемое количество только тех возобновляемых ресурсов, потребление которых превышает некоторый заданный уровень [[97], [113], [136]]. Или, как в работе [[30]], иногда рассматривается минимизация накапливаемого отклонения уровня использования ресурсов от некоторого заданного константного уровня. Такой подход позволяет контролировать и устранять не только перерасход ресурсов, но и возможные простои. Интересный подход изложен в работе [[46]], где предлагается минимизировать как количество, так и продолжительность разрывов в графике (профиле) потребления ресурсов. Очевидно, что данная стратегия также ориентирована на снижение простоя ресурсов. В более развитых постановках [[37]] предлагается разделить всё доступное количество возобновляемых ресурсов на количество, доступное изнутри, и количество, доступное извне. Целевая функция при такой постановке предполагает минимизацию расходов, связанных с использованием только взятых извне ресурсов.

В нотации $\alpha|\beta|\gamma$ все описанные выше целевые функции для решения задачи по выравниванию загруженности ресурсов можно обобщенно представить в характеристике γ как $\sum c_k f\big(r_k(S,t)\big)$, где c_k – стоимость использования одной единицы ресурса k, f – функция профиля использования ресурса k в количестве $r_k(S,t)$ (t – момент времени в расписании S). Следует заметить, что если $f = R_k$, что соответствует равномерному профилю потребления ресурса k, то данная задача (задача выравнивания загруженности ресурсов) сводиться к задаче ресурсных инвестиций.

6.5. Минимизация не возобновляемых ресурсов

Минимизация потребления не возобновляемых ресурсов имеет место только в мультимодальной постановке RCPSP-задачи, так как во всех других постановках потребление ресурсов задачами фиксировано, и, следовательно, сумма потребления не возобновляемых ресурсов всегда будет одинаковой. Во

всём остальном целевые функции, ориентированные на не возобновляемые ресурсы являются, фактически, точной копией возобновляемых ресурсов. В традиционной мультимодальной постановке RCPSP-задачи минимизируется время выполнения всего проекта, не превышая при этом предела доступности ресурсов. Если же необходимо минимизировать потребление не возобновляемых ресурсов, то аналогично задаче ресурсных инвестиций в постановке задачи вводить крайний срок выполнения проекта (deadline), который не должен быть превышен.

Такая целевая функция, ориентированная на не возобновляемые ресурсы, была подробно рассмотрена в публикациях [[85], [86]] в рамках дискретной задачи компромисса времени и стоимости. В данных публикациях не возобновляемые ресурсы интерпретировались в качестве денег. Аналогичный подход изложен и в работах [[30], [80]]. А вот в публикациях [[30], [113]] предлагается минимизировать количество только тех используемых не возобновляемых ресурсов, суммарное потребление которых превышает некоторый заданный предел.

Все эти целевые функции можно обобщённо представить в нотации $\alpha|\beta|\gamma$ как $\gamma = \sum c_k f\left(r_k(S)\right)$, где c_k – стоимость потребления одной единицы не возобновляемого ресурса k, f – функция потребления ресурса k в количестве $r_k(S)$ в расписании S.

6.6. Минимизация общей стоимости проекта

Оптимизация расписания по критерию стоимости часто является важнейшим аспектом. Как правило, такая оптимизация выполняется в рамках RCPSP-задачи без ресурсов. Обычно для каждой работы j вводиться понятие стоимости её выполнения c_{jt}, зависящее от момента времени t, когда работа j начала своё выполнение [[137], [138], [139]]. Как верно замечено в работе [[139]], целевая функция, ориентированная на стоимость, является обобщением разных других хорошо известных целевых функций, таких как минимизация времени выполнения всего проекта, выполнение работ точно в срок или максимизация чистой приведённой стоимости.

Обычно минимизируют общую стоимость проекта, которая состоит из стоимостей опережений и запаздываний относительно директивных сроков выполнения работ, а также из стоимостей, связанных с продолжительностью выполнения работ [[140]]. Более экзотические постановки предполагают, что продолжительность выполнения работы может быть сокращена за дополнительную плату. При этом проект должен быть завершён не позднее некоторого заданного крайнего срока (deadline).

В публикации [[141]] рассматривается минимизация стоимости, состоящей из стоимостей сбоев выполнения работ (то есть неожиданного сокращения продолжительности их выполнения), стоимостей материальных (не возобновляемых) ресурсов и расходов на поддержание инвентаря

(возобновляемых ресурсов). Также в данной работе рассматриваются бонусы и штрафы за раннее и за позднее (относительно некоторого заданного директивного срока) завершение проекта соответственно.

Интересная постановка представлена в публикации [[142]]. Авторы рассмотрели постановку с работами, продолжительность выполнения которых находиться в некотором заданном интервале, едином для всех работ. Такой подход привёл авторов к целевой функции, в которой стоимость выполнения каждой работы, связанная с их продолжительностью, является выпуклой функцией.

Также заслуживает внимания двухкомпонентная целевая функция [[143]]. Стоимость выполнения проекта, заложенная в данную целевую функцию, прямо пропорциональна продолжительности выполнения проекта. При этом работы в проекте могут быть скомбинированы (объединены) для уменьшения продолжительности выполнения проекта, а стоимость выполнения данных работ при этом будет складываться. Данная целевая функция может быть использована только в рамках постановки без ресурсных ограничений.

6.7. Максимизация чистой приведенной стоимости

Другой не менее важный тип целевой функции появляется, когда в ходе выполнения проекта имеют место финансовые потоки. Отток денежных средств происходит в результате выполнения какой-либо работы или использования какого-либо ресурса. С другой стороны, приток денежных средств происходит в результате инвестиций, авансовых платежей или после достижения определённых ключевых точек выполнения всего проекта. Также не исключён вариант, когда приток денежных средств осуществляет один раз в конце выполнения проекта (оплата за выполнение проекта). Чистой приведённой стоимостью является разница между суммами всех притоков и оттоков денежных средств. Этот показатель является одним из важнейших для инвесторов. Поэтому целевой функций должна быть максимизация чистой приведённой стоимости.

Данная задача решается в рамках стандартных ограничений RCPSP-задачи. Обычно максимизации подвергается общая сумма оттоков и притоков денежных средств, при этом оттоки участвуют в сумме со знаком «минус». Кроме того в данной постановке допускается участие скидок при финансовых оттоках. Скидки могут быть как фиксированными, так и зависящими от каких-либо проектных показателей. Всё это детально рассматривается в работах [[144], [145], [146], [147]]. Кроме того следует отметить работы [[38], [148]], в которых исследуется максимизация чистой приведённой стоимости в рамках мультимодальной постановки, а также [[149]] – в рамках задачи ресурсных инвестиций. Так же данной целевой функции посвящены публикации [[53], [96], [150], [151], [152]].

В нотации $\alpha|\beta|\gamma$ целевая функция максимизации чистой приведённой стоимости может быть представлена как $\sum c_j^F \beta^{F_j}$, где c_j^F – денежный поток

(приток или отток) работы j, β^{F_j} – скидка, предоставляемая в данном денежном потоке.

6.8. Многоцелевые функции

Во всех ранее обсуждаемых случаях применялась целевая функция, оптимизирующая только один параметр или характеристику проекта, в то время как все остальные при необходимости контролировались с помощью введения дополнительных ограничений. Однако, в реальности часто необходимо оптимизировать выполнение проекта по нескольким критериям, то есть применить многоцелевую функцию.

Так наиболее частым и распространённым способом получения многоцелевой функции является взвешенное суммирование всех интересующих параметров и/или характеристик. Обычно взвешенному суммированию подвергаются общее время выполнения проекта, опережение и запаздывание работ, равномерность загруженности ресурсов и потребление ресурсов в целом. Такой подход можно встретить во многих работах [[30], [46], [120]].

Другим получивший широкое распространение способом получения многоцелевой функции является генерация расписаний, оптимизированных с точки зрения закона Парето. Общий принцип данного закона прост: 20% приложенных усилий дают 80% результата, а остальные 20% результата достигаются остальными 80% усилий. Данный принцип применительно к методам построения расписания был достаточно подробно изучен и успешно применён в некоторых публикациях [[136], [153], [154], [155]]. Особенно стоит отметить работу [[156]], в которой авторами представлена оригинальная многоцелевая функция, основанная на системе знаний.

Таким образом, многоцелевые функции, с одной стороны, являются наиболее применимыми в реальных условиях, в другой – наиболее сложными. При этом многоцелевые функции по своей природе является составными, что позволяет достаточно гибко выбирать оптимизируемые критерии проекта.

7. Заключение

Таким образом, представлен обзор современных моделей и методов теории расписаний. Для этой цели был выбран класс задач ресурсного планирования RCPSP, который находит широкое практическое применение и обобщает многочисленные постановки теории расписаний. Проведенный обзор позволяет выявить основные понятия и математические модели, которые могут быть положены в основу универсального объектно-ориентированного каркаса для разработки программных приложений.

Список литературы

[1]. Интернет-ресурс: https://ru.wikipedia.org/wiki/Теория_расписаний (в редакции от 4 июня 2013 14:07).

[2]. Лазарев А. А., Гафаров Е. Р. Теория расписаний: задачи и алгоритмы. Московский государственный университет им. М.В. Ломоносова (МГУ), Москва, Россия, 2011, 222 с.

[3]. Коваленко Ю. В. Сложность некоторых задач теории расписаний и эволюционные алгоритмы их решения. Диссертация на соискание учёной степени кандидата физико-математических наук, Омск, Россия, 2013, 129 с.

[4]. Sprecher A. Resource-constrained project scheduling: Exact methods for the multi-mode case. Серия книг: Lecture Notes in Economics and Mathematical Systems, том 409, Springer, Berlin, Germany, 1994, 142 с.

[5]. Интернет-ресурс: https://ru.wikipedia.org/wiki/Задача_об_упаковке_в_контейнеры (в редакции от 19 сентября 2013 21:17).

[6]. Интернет-ресурс: https://ru.wikipedia.org/wiki/Задача_о_рюкзаке (в редакции от 29 июня 2014 07:47).

[7]. Stadtler H. Multilevel capacitated lot-sizing and resource-constrained project scheduling: An integrating perspective. International Journal of Production Research, 2005, том 43, выпуск 24, с. 5253-5270.

[8]. Интернет-ресурс: https://ru.wikipedia.org/wiki/Задача_коммивояжёра (в редакции от 21 мая 2014 18:07).

[9]. Brucker P., Knust S. Resource-constrained project scheduling and timetabling. Lecture Notes in Computer Science, 2001, том 2079, с. 277-293.

[10]. Drexl A., Salewski F. Distribution requirements and compactness constraints in school timetabling. European Journal of Operational Research, 1997, том 102, выпуск 1, с. 193-214.

[11]. Pritsker A. A. B., Watters L. J., Wolfe P. M. Multiproject scheduling with limited resources: A zero-one programming approach. Management Science, 1969, том 16, выпуск 1, с. 93-107.

[12]. Blazewicz J., Lenstra J. K., Rinnooy Kan A. H. G. Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics, 1983, том 5, с. 11-24.

[13]. Ahuja H. N. Construction performance control by networks. John Wiley & Sons, New York, USA, 1976, 636 с.

[14]. Davis E. W., Patterson J. H. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. Management Science, 1975, том 21, выпуск 8, с. 944-955.

[15]. Hegazy T. Optimization of resource allocation and leveling using genetic algorithms. Journal of Construction Engineering and Management, 1999, том 125, выпуск 3, с. 167-175.

[16]. Brooks G. N., White C. R. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. Journal of Industrial Engineering, 1966, том 17, выпуск 2, с. 173-186.

[17]. Gomory R. E. An all integer programming algorithm. Глава из книги [[157]], с. 193-206.

[18]. Szwarg W. Solutions of the akers-friedman scheduling problem. Operations Research, 1960, том 8, выпуск 6, с. 782-788.

[19]. Ashour S. A Decomposition Approach for the Machine Scheduling Problem. University of Iowa, Iowa City, USA, 1967, 416 с.
[20]. Giffler B., Thompson G. L., Van Ness V. Numerical experience with the linear and Monte-Carlo algorithms for solving production scheduling problems. Глава из книги [[157]], с. 21-29.
[21]. Page E. S. An approach to the scheduling of the N jobs on M machines. Journal of the Royal Statistical Society, 1961, том 23, выпуск 2, с 484-492.
[22]. Kolisch R. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management, 1996, том 14, выпуск 3, с. 179-192.
[23]. Boctor F. F. Some efficient multi-heuristic procedures for resource constrained project scheduling. European Journal of Operational Research, 1990, том 49, выпуск 1, с. 3-13.
[24]. Graham R. L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 1979, том 5, с. 287-326.
[25]. Brucker P., Drexl A., Möhring R., Neumann K., Pesch E. Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research, 1999, том 112, выпуск 1, с. 3-41.
[26]. Słowin´ski R. Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. European Journal of Operational Research, 1981, том 7, выпуск 3, с. 265-273.
[27]. Weglarz J. On certain models of resource allocation problems. Kybernetes, 1980, том 9, выпуск 1, с. 61-66.
[28]. Böttcher J., Drexl A., Kolisch R., Salewski F. Project scheduling under partially renewable resource constraints. Management Science, 1999, том 45, выпуск 4, с. 543-559.
[29]. Zhu G., Bard J. F., Yu G. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS Journal on Computing, 2006, том 18, выпуск 3, с. 377-390.
[30]. Nudtasomboon N., Randhawa S. U. Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs. Computers and Industrial Engineering, 1997, том 32, выпуск 1, с. 227-242.
[31]. Neumann K., Schwindt C. Project scheduling with inventory constraints. Mathematical Methods of Operations Research, 2003, том 56, выпуск 3, с. 513-533.
[32]. Bartels J. H., Zimmermann J. Scheduling tests in automotive R&D projects. European Journal of Operational Research, 2009, том 193, выпуск 3, с. 805-819.
[33]. Schwindt C., Trautmann N. Batch scheduling in process industries: An application of resource-constrained project scheduling. OR Spectrum, 2000, том 22, выпуск 4, с. 501-524.
[34]. Neumann K., Schwindt C., Trautmann N. Scheduling of continuous and discontinuous material flows with intermediate storage restrictions. European Journal of Operational Research, 2005, том 165, выпуск 2, с. 495-509.
[35]. Weglarz J., Blazewicz J., Cellary W., Słowin´ski R. Algorithm 520: An automatic revised simplex method for constrained resource network scheduling. ACM Transactions on Mathematical Software, 1977, том 3, выпуск 3, с. 295-300.
[36]. Blazewicz J., Ecker K., Pesch E., Schmidt G., Weglarz J. Handbook on Scheduling: From Theory to Applications. Серия книг: International Handbooks on Information Systems, Springer, Berlin, Germany, 2007, 647 с.
[37]. Kis T. A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. Mathematical Programming, 2005, том 103, выпуск 3, с. 515-539.

[38]. Waligora G. Discrete-continuous project scheduling with discounted cash flows – A tabu search approach. Computers and Operations Research, 2008, том 35, выпуск 7, с. 2141-2153.
[39]. Kis T. RCPS with Variable intensity activities and feeding precedence constraints. Глава из книги [[158]], с. 105-129.
[40]. Jozefowska J., Mika M., Rozycki R., Waligora G., Weglarz J. Solving the discrete-continuous project scheduling problem via its discretization. Mathematical Methods of Operations Research, 2000, том 52, выпуск 3, с. 489-499.
[41]. Weglarz J. Project scheduling with continuously divisible, doubly-constrained resources. Management Science, 1981, том 27, выпуск 9, с. 1040-1053.
[42]. Bianco L., Dell'olmo P., Speranza M. G. Heuristics for multimode scheduling problems with dedicated resources. European Journal of Operational Research, 1998, том 107, выпуск 2, с. 260-271.
[43]. Bianco L., Caramia M., Dell'olmo P. Solving a preemptive project scheduling problem with coloring techniques. Глава из книги [[159]], с. 135-146.
[44]. Dorndorf U., Phan-Huy T., Pesch E. A survey of interval capacity consistency tests for time- and resource constrained scheduling. Глава из книги [[159]], с. 213-238.
[45]. Dorndorf U., Pesch E., Phan-Huy T. Constraint propagation techniques for the disjunctive scheduling problem. Artificial Intelligence, 2000, том 122, выпуск 1-2, с. 189-240.
[46]. Bomsdorf F., Derigs U. A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem. OR-Spectrum, 2008, том 30, выпуск 4, с. 751-772.
[47]. Klein R., Scholl A. Scattered branch and bound – An adaptive search strategy applied to resource-constrained project scheduling. Central European Journal of Operations Research, 1999, том 7, с. 177-201.
[48]. Nonobe K., Ibaraki T. Formulation and tabu search algorithm for the resource constrained project scheduling problem. Глава (с. 557-588) из книги под ред. Ribeiro C. C., Hansen P. Essays and Surveys in Metaheuristics. Серия книг: Operations Research/Computer Science Interfaces Series, том 15, Springer, Berlin, Germany, 2002, 664 с.
[49]. Pesch E. Lower bounds in different problem classes of project schedules with resource constraints. Глава из книги [[159]], с. 53-76.
[50]. Klein R., Scholl A. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. European Journal of Operational Research, 1999, том 112, выпуск 2, с. 322-346.
[51]. Klein R. Project scheduling with time-varying resource constraints. International Journal of Production Research, 2000, том 38, выпуск 16, с. 3937-3952.
[52]. Hartmann S. Project scheduling under limited resources: Models, methods, and applications. Серия книг: Lecture Notes in Economics and Mathematical Systems, том 478, Springer, Berlin, Germany, 1999, 221 с.
[53]. Icmeli O., Rom W. O. Solving the resource constrained project scheduling problem with optimization subroutine library. Computers and Operations Research, 1996, том 23, выпуск 8, с. 801-817.
[54]. Bartusch M., Möhring R. H., Radermacher F. J. Scheduling project networks with resource constraints and time windows. Annals of Operations Research, 1988, том 16, выпуск 1, с. 201-240.

[55]. Debels D., Vanhoucke M. The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects. Computers and Industrial Engineering, 2008, том 54, выпуск 1, с. 140-154.
[56]. Demeulemeester E. L., Herroelen W. S. An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. European Journal of Operational Research, 1996, том 90, выпуск 2, с. 334-348.
[57]. Ballestin F., Valls V., Quintanilla S. Pre-emption in resource-constrained project scheduling. European Journal of Operational Research, 2008, том 189, выпуск 3, с. 1136-1152.
[58]. Buddhakulsomsiria J., Kim D. S. Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research, 2006, том 175, выпуск 1, с. 279-295.
[59]. Buddhakulsomsiria J., Kim D. S. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research, 2007, том 178, выпуск 2, с. 374-390.
[60]. Franck B., Neumann K., Schwindt C. Project scheduling with calendars. OR Spektrum, 2001, том 23, выпуск 3, с. 325-334.
[61]. Drezet L. E., Billaut J. C. A project scheduling problem with labour constraints and time-dependent activities requirements. International Journal of Production Economics, 2008, том 112, выпуск 1, с. 217-225.
[62]. Cavalcante C. C. B., C. de Souza C., Savelsbergh M. W. P., Wang Y., Wolsey L. A. Scheduling projects with labor constraints. Discrete Applied Mathematics, 2001, том 112, выпуск 1-3, с. 27-52.
[63]. Mika M., Waligora G., Weglarz J. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research, 2008, том 187, выпуск 3, с. 1238-1250.
[64]. Mika M., Waligora G., Weglarz J. Modelling setup times in project scheduling. Глава из книги [[158]], с. 131-165.
[65]. Drexl A., Nissen R., Patterson J. H., Salewski F. Progen/πx – An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. European Journal of Operational Research, 2000, том 125, выпуск 1, с. 59-72.
[66]. Vanhoucke M. Setup times and fast tracking in resource-constrained project scheduling. Computers and Industrial Engineering, 2008, том 54, выпуск 4, с. 1062-1070.
[67]. Elmaghraby S. E. Activity networks: Project planning and control by network models. Wiley, New York, USA, 1977, 443 с.
[68]. Alcaraz J., Maroto C., Ruiz R. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society, 2003, том 54, выпуск 6, с. 614-626.
[69]. Bouleimen K., Lecocq H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research, 2003, том 149, выпуск 2, с. 268-281.
[70]. Hartmann S. Project scheduling with multiple modes: A genetic algorithm. Annals of Operations Research, 2001, том 102, выпуск 1-4, с. 111-135.
[71]. Jarboui B., Damak N., Siarry P., Rebai A. A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation, 2008, том 195, выпуск 1, с. 299-308.

[72]. Jozefowska J., Mika M., Rozycki R., Waligora G., Weglarz J. Simulated annealing for multi-mode resource-constrained project scheduling. Annals of Operations Research, 2001, том 102, выпуск 1-4, с. 137-155.
[73]. Özdamar L. A genetic algorithm approach to a general category project scheduling problem. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 1999, том 29, выпуск 1, с. 44-59.
[74]. Calhoun K. M., Deckro R. F., Moore J. T., Chrissis J. W., Hove J. C. V. Planning and re-planning in project and production scheduling. Omega, 2002, том 30, выпуск 3, с. 155-170.
[75]. B. de Reyck, Herroelen W. S. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research, 1999, том 119, выпуск 2, с. 538-556.
[76]. Heilmann R. Resource-constrained project scheduling: A heuristic for the multi-mode case. OR Spektrum, 2001, том 23, выпуск 3, с. 335-357.
[77]. Heilmann R. A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. European Journal of Operational Research, 2003, том 144, выпуск 2, с. 348-365.
[78]. Li H., Womer K. Modeling the supply chain configuration problem with resource constraints. International Journal of Project Management, 2008, том 26, выпуск 6, с. 646-654.
[79]. Tiwari V., Patterson J. H., Mabert V. A. Scheduling projects with heterogeneous resources to meet time and quality objectives. European Journal of Operational Research, 2009, том 193, выпуск 3, с. 780-790.
[80]. Tareghian H. R., Taheri S. H. A solution procedure for the discrete time, cost and quality tradeoff problem using electromagnetic scatter search. Applied Mathematics and Computation, 2007, том 190, выпуск 2, с. 1136-1145.
[81]. Salewski F., Schirmer A., Drexl A. Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. European Journal of Operational Research, 1997, том 102, выпуск 1, с. 88-110.
[82]. Demeulemeester E. L., B. de Reyck, Herroelen W. S. The discrete time/resource trade-off problem in project networks – A branch-and-bound approach. IIE Transactions, 2000, том 32, выпуск 11, с. 1059-1069.
[83]. Ranjbar M., Kianfar F. Solving the discrete time/resource trade-off problem in project scheduling with genetic algorithms. Applied Mathematics and Computation, 2007, том 191, выпуск 2, с. 451-456.
[84]. Ranjbar M., B. de Reyck, Kianfar F. A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. European Journal of Operational Research, 2009, том 193, выпуск 1, с. 35-48.
[85]. Akkan C., Drexl A., Kimms A. Network decomposition-based benchmark results for the discrete time-cost tradeoff problem. European Journal of Operational Research, 2005, том 165, выпуск 2, с. 339-358.
[86]. Demeulemeester E. L., B. de Reyck, Foubert B., Herroelen W. S., Vanhoucke M. New computational results on the discrete time/cost trade-off problem in project networks. Journal of the Operational Research Society, 1998, том 49, выпуск 11, с. 1153-1163.
[87]. Deckro R. F., Hebert J. E., Verdini W. A., Grimsrud P. H., Venkateshwar S. Nonlinear time/cost tradeoff models in project management. Computers and Industrial Engineering, 1995, том 28, выпуск 2, с. 219-229.

[88]. Chassiakos A. P., Sakellaropoulos S. P. Time-cost optimization of construction projects with generalized activity constraints. Journal of Construction Engineering and Management, 2005, том 131, выпуск 10, с. 1115-1124.
[89]. Klein R., Scholl A. PROGRESS: Optimally solving the generalized resource-constrained project scheduling problem. Mathematical Methods of Operations Research, 2000, том 52, выпуск 3, с. 467-488.
[90]. Kolisch R. Integrated scheduling, assembly area- and part-assignment for large-scale, make-to-order assemblies. International Journal of Production Economics, 2000, том 64, выпуск 1-3, с. 127-141.
[91]. Vanhoucke M. Work continuity constraints in project scheduling. Journal of Construction Engineering and Management, 2006, том 132, выпуск 1, с. 14-25.
[92]. Demeulemeester E. L., Herroelen W. S. Modelling setup times, process batches and transfer batches using activity network logic. European Journal of Operational Research, 1996, том 89, выпуск 2, с. 355-365.
[93]. Cesta A., Oddi A., Smith S. F. A constraint-based method for project scheduling with time windows. Journal of Heuristics, 2002, том 8, выпуск 1, с. 109-136.
[94]. Dorndorf U., Pesch E., Phan Huy T. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalized precedence constraints. Management Science, 2000, том 46, выпуск 10, с. 1365-1384.
[95]. Schwindt C. Generation of resource-constrained project scheduling problems subject to temporal constraints. Technical Report WIOR-543, Universität Karlsruhe, Germany, 1998.
[96]. Neumann K., Zimmermann J. Exact and truncated branch-and-bound procedures for resource-constrained project scheduling with discounted cash flows and general temporal constraints. Central European Journal of Operations Research, 2002, том 10, выпуск 4, с. 357-380.
[97]. Neumann K., Zimmermann J. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. European Journal of Operational Research, 2000, том 127, выпуск 2, с. 425-443.
[98]. Neumann K., Schwindt C., Zimmermann J. Recent results on resource-constrained project scheduling with time windows: Models, solution methods, and applications. Central European Journal of Operations Research, 2002, том 10, выпуск 2, с. 113-148.
[99]. Sabzehparvar M., Seyed-Hosseini S. M. A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. Journal of Supercomputing, 2008, том 44, выпуск 3, с. 257-273.
[100]. Franck B., Neumann K. Resource-constrained project scheduling with time windows: Structural questions and priority rule methods. Technical Report WIOR-492, Universität Karlsruhe, Germany, 1997.
[101]. Ballestin F., Valls V., Quintanilla S. Due dates and RCPSP. Глава из книги [[158]], с. 79-104.
[102]. Brânzei R., Ferrari G., Fragnelli V., Tijs S. Two approaches to the problem of sharing delay costs in joint projects. Annals of Operations Research, 2002, том 109, выпуск 1-4, с. 359-374.
[103]. Chiu H. N., Tsai D. M. An efficient search procedure for the resource-constrained multi-project scheduling problem with discounted cash flows. Construction Management and Economics, 2002, том 20, выпуск 1, с. 55-66.
[104]. Özdamar L., Ulusoy G., Bayyigit M. A heuristic treatment of tardiness and net present value criteria in resource constrained project scheduling. International Journal of

Physical Distribution and Logistics Management, 1998, том 28, выпуск 9/10, с. 805-824.
[105]. Baptiste P., Pape C. L., Nuijten W. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research, 1999, том 92, с. 305-333.
[106]. Yang H. H., Chen Y. L. Finding the critical path in an activity network with time-switch constraints. European Journal of Operational Research, 2000, том 120, выпуск 3, с. 603-613.
[107]. Vanhoucke M., Demeulemeester E. L., Herroelen W. S. Discrete time/cost trade-offs in project scheduling with time-switch constraints. Journal of the Operational Research Society, 2002, том 53, выпуск 7, с. 741-751.
[108]. Krüger D., Scholl A. Managing and modelling general resource transfers in (multi-)project scheduling. OR Spektrum, 2010, том 32, выпуск 2, с. 369-394.
[109]. Elmaghraby S. E. An algebra for the analysis of generalized activity networks. Management Science, 1964, том 10, выпуск 3, с. 494-514.
[110]. Belhe U., Kusiak A. Resource-constrained scheduling of hierarchically structured design activity networks. IEEE Transactions on Engineering Management, 1995, том 42, выпуск 2, с. 150-158.
[111]. Neumann K. Stochastic project networks: Temporal analysis, scheduling and cost minimization. Серия книг: Lecture Notes in Economics and Mathematical Systems, том 344, Springer, Berlin, Germany, 1990, 237 с.
[112]. Kuster J., Jannach D. Handling airport ground processes based on resource-constrained project scheduling. Lecture Notes in Computer Science, 2006, том 4031, с. 166-176.
[113]. Viana A., P. de Sousa J. Using metaheuristics in multiobjective resource constrained project scheduling. European Journal of Operational Research, 2000, том 120, выпуск 2, с. 359-374.
[114]. Vanhoucke M., Demeulemeester E. L., Herroelen W. S. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. Annals of Operations Research, 2001, том 102, выпуск 1-4, с. 179-196.
[115]. Franck B., Schwindt C. Different resource-constrained project scheduling models with minimal and maximal time-lags. Technical Report WIOR-450, Universität Karlsruhe, Germany, 1995.
[116]. Lorenzoni L. L., Ahonen H., G. de Alvarenga A. A multi-mode resource-constrained scheduling problem in the context of port operations. Computers and Industrial Engineering, 2006, том 50, выпуск 1-2, с. 55-65.
[117]. Vanhoucke M. Scheduling an R&D project with quality-dependent time slots. Lecture Notes in Computer Science, 2006, том 3982, с. 621-630.
[118]. Rom W. O., Tukel O. I., Muscatello J. R. MRP in a job shop environment using a resource constrained project scheduling model. Omega, 2002, том 30, выпуск 4, с. 275-286.
[119]. Nazareth T., Verma S., Bhattacharya S., Bagchi A. The multiple resource constrained project scheduling problem: A breadth-first approach. European Journal of Operational Research, 1999, том 112, выпуск 2, с. 347-366.
[120]. Al-Fawzan M., Haouari M. A bi-objective model for robust resource-constrained project scheduling. International Journal of Production Economics, 2005, том 96, выпуск 2, с. 175-187.
[121]. Abbasi B., Shadrokh S., Arkat J. Bi-objective resource-constrained project scheduling with robustness and makespan criteria. Applied Mathematics and Computation, 2006, том 180, выпуск 1, с. 146-152.

[122]. Kobylanski P., Kuchta D. A note on the paper by M. A. Al-Fawzan and M. Haouari about a bi-objective problem for robust resource-constrained project scheduling. International Journal of Production Economics, 2007, том 107, выпуск 2, с. 496-501.
[123]. Chtourou H., Haouari M. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. Computers and Industrial Engineering, 2008, том 55, выпуск 1, с. 183-194.
[124]. Icmeli-Tukel O., Rom W. O. Ensuring quality in resource constrained project scheduling. European Journal of Operational Research, 1997, том 103, выпуск 3, с. 483-496.
[125]. Van de Vonder S., Demeulemeester E. L., Herroelen W. S. A classification of predictive-reactive project scheduling procedures. Journal of Scheduling, 2007, том 10, выпуск 3, с. 195-207.
[126]. Sakkout H., Wallace M. Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints, 2000, том 5, выпуск 4, с. 359-388.
[127]. Zhu G., Bard J. F., Yu G. Disruption management for resource-constrained project scheduling. Journal of the Operational Research Society, 2005, том 56, с. 365-381.
[128]. Neumann K., Schwindt C., Zimmermann J. Resource-constrained project scheduling with time windows: Recent developments and new applications. Глава из книги [[158]], с. 375-407.
[129]. Drexl A., Kimms A. Optimization guided lower and upper bounds for the resource investment problem. Journal of the Operational Research Society, 2001, том 52, выпуск 3, с. 340-351.
[130]. Ranjbar M., Kianfar F., Shadrokh S. Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm. Applied Mathematics and Computation, 2008, том 196, выпуск 2, с. 879-888.
[131]. Yamashita D. S., Armentano V. A., Laguna M. Robust optimization models for project scheduling with resource availability cost. Journal of Scheduling, 2007, том 10, выпуск 1, с. 67-76.
[132]. Shadrokh S., Kianfar F. A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. European Journal of Operational Research, 2007, том 181, выпуск 1, с. 86-101.
[133]. Nübel H. The resource renting problem subject to temporal constraints. OR-Spektrum, 2001, том 23, выпуск 3, с. 359-381.
[134]. Ballestin F. A genetic algorithm for the resource renting problem with minimum and maximum time lags. Lecture Notes in Computer Science, 2007, том 4446, с. 25-35.
[135]. Bandelloni M., Tucci M., Rinaldi R. Optimal resource leveling using non-serial dynamic programming. European Journal of Operational Research, 1994, том 78, выпуск 2, с. 162-177.
[136]. Davis K. R., Stam A., Grzybowski R. A. Resource constrained project scheduling with multiple objectives: A decision support approach. Computers and Operations Research, 1992, том 19, выпуск 7, с. 657-669.
[137]. Maniezzo V., Mingozzi A. The project scheduling problem with irregular starting time costs. Operations Research Letters, 1999, том 25, выпуск 4, с. 175-182.
[138]. Möhring R. H., Schulz A. S., Stork F., Uetz M. On project scheduling with irregular starting time costs. Operations Research Letters, 2001, том 28, выпуск 4, с. 149-154.
[139]. Möhring R. H., Schulz A. S., Stork F., Uetz M. Solving project scheduling problems by minimum cut computations. Management Science, 2003, том 49, выпуск 3, с. 330-350.

[140]. Achuthan N., Hardjawidjaja A. Project scheduling under time dependent costs – A branch and bound algorithm. Annals of Operations Research, 2001, том 108, выпуск 1-4, с. 55-74.
[141]. Dodin B., Elimam A. A. Integrated project scheduling and material planning with variable activity duration and rewards. IIE Transactions, 2001, том 33, выпуск 11, с. 1005-1018.
[142]. Nonobe K., Ibaraki T. A metaheuristic approach to the resource constrained project scheduling with variable activity durations and convex cost functions. Глава из книги [[158]], с. 225-248.
[143]. Rummel J. L., Walter Z., Dewan R., Seidmann A. Activity consolidation to improve responsiveness. European Journal of Operational Research, 2005, том 161, выпуск 3, с. 683-703.
[144]. Kimms A. Maximizing the net present value of a project under resource constraints using a lagrangian relaxation based heuristic with tight upper bounds. Annals of Operations Research, 2001, том 102, выпуск 1-4, с. 221-236.
[145]. Mika M., Waligora G., Weglarz J. Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. European Journal of Operational Research, 2005, том 164, выпуск 3, с. 639-668.
[146]. Padman R., Zhu D. Knowledge integration using problem spaces: A study in resource-constrained project scheduling. Journal of Scheduling, 2006, том 9, выпуск 2, с. 133-152.
[147]. Vanhoucke M., Demeulemeester E. L., Herroelen W. S. On maximizing the net present value of a project under renewable resource constraints. Management Science, 2001, том 47, выпуск 8, с. 1113-1121.
[148]. Varma V. A., Uzsoy R., Pekny J., Blau G. Lagrangian heuristics for scheduling new product development projects in the pharmaceutical industry. Journal of Heuristics, 2007, том 13, выпуск 5, с. 403-433.
[149]. Najafi A. A., Niaki S. T. A. A genetic algorithm for resource investment problem with discounted cash flows. Applied Mathematics and Computation, 2006, том 183, выпуск 2, с. 1057-1070.
[150]. Herroelen W. S., P. van Dommelen, Demeulemeester E. L. Project network models with discounted cash flows: A guided tour through recent developments. European Journal of Operational Research, 1997, том 100, выпуск 1, с. 97-121.
[151]. Dayanand N., Padman R. On modelling payments in projects. Journal of the Operational Research Society, 1997, том 48, выпуск 9, с. 906-918.
[152]. Etgar R., Shtub A., LeBlanc L. J. Scheduling projects to maximize net present value – the case of time-dependent, contingent cash flows. European Journal of Operational Research, 1997, том 96, выпуск 1, с. 90-96.
[153]. Hapke M., Jaszkiewicz A., Slowinski R. Interactive analysis of multiple-criteria project scheduling problems. European Journal of Operational Research, 1998, том 107, выпуск 2, с. 315-324.
[154]. Slowinski R., Soniewicki B., Weglarz J. DSS for multiobjective project scheduling. European Journal of Operational Research, 1994, том 79, выпуск 2, с. 220-229.
[155]. Dörner K. F., Gutjahr W. J., Hartl R. F., Strauss C., Stummer C. Nature-inspired metaheuristics for multiobjective activity crashing. Omega, 2008, том 36, выпуск 6, с. 1019-1037.
[156]. Nabrzynski J., Weglarz J. Knowledge-based multiobjective project scheduling problems. Глава из книги [[159]], с. 383-411.

[157]. Под ред. Muth J. F., Thompson G. L. Industrial Scheduling. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1963, 467 с.
[158]. Под ред. Jozefowska J., Weglarz J. Perspectives in Modern Project Scheduling. Springer Science & Business Media, New York, USA, 2006, 466 с.
[159]. Под ред. Weglarz J. Project Scheduling: Recent Models, Algorithms and Applications. Springer, New York, USA, 1999, 535 с.

A survey of emerging models and methods of scheduling

A. S. Anichkin
<anton.anichkin@ispras.ru>
V. A. Semenov
<sem@ispras.ru>
SP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. The paper is addressed the important problem of systematization and conceptualization of scheduling theory. Scheduling is widely applied in such subject areas as production management, traffic flow organization, planning of projects and resource management into computing systems. However, a diversity of mathematical models and methods of scheduling poses usually the problem of design of fast algorithm as well as the problem of efficient software implementation taking into account specificity of subject area. A usage of typical solvers from shared mathematical libraries is exceedingly limited. A usage of object-oriented frameworks for software implementations is more perspective. In the paper we make an attempt to systematize and to generalize models and methods of scheduling theory with the aim of construction of such a framework. The main attention is paid to resource-constrained project scheduling problems. These problems are widely used in practice on the one hand and bring together different mathematical statements arising in related subject areas and disciplines on the other hand.

Keywords: scheduling, resource-constrained project scheduling problem, software engineering, object-oriented programming

References

[1]. Internet page: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%80%D0%B0%D1%81%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D0%B9 (as amended June 4, 2013 at 14:07).

[2]. Lazarev A. A., Gafarov E. R. Teoriya raspisanij: zadachi i algoritmy [Scheduling theory: problems and algorithms]. Lomonosov Moscow State University, Moscow, Russia, 2011, 222 p.

[3]. Kovalenko YU. V. Slozhnost' nekotorykh zadach teorii raspisanij i ehvolyutsionnye algoritmy ikh resheniya [Complexity of some scheduling problems and evolutionary algorithms for their solution]. PhD thesis, Omsk, Russia, 2013, 129 p.

[4]. Sprecher A. Resource-constrained project scheduling: Exact methods for the multi-mode case. Series: Lecture Notes in Economics and Mathematical Systems, vol. 409, Springer, Berlin, Germany, 1994, 142 p.

[5]. Internet page: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE%D0%B1_%D1%83%D0%BF%D0%B0%D0%BA%D0%BE%D0%B2%D0%BA%D0%B5_%D0%B2_%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80%D1%8B (as amended September 19, 2013 at 21:17).
[6]. Internet page: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D1%80%D0%B0%D0%BD%D1%86%D0%B5 (as amended June 29, 2014 at 07:47).
[7]. Stadtler H. Multilevel capacitated lot-sizing and resource-constrained project scheduling: An integrating perspective. International Journal of Production Research, 2005, vol. 43, no. 24, pp. 5253-5270.
[8]. Internet page: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D1%91%D1%80%D0%B0 (as amended May 21, 2014 at 18:07).
[9]. Brucker P., Knust S. Resource-constrained project scheduling and timetabling. Lecture Notes in Computer Science, 2001, vol. 2079, pp. 277-293.
[10]. Drexl A., Salewski F. Distribution requirements and compactness constraints in school timetabling. European Journal of Operational Research, 1997, vol. 102, no. 1, pp. 193-214.
[11]. Pritsker A. A. B., Watters L. J., Wolfe P. M. Multiproject scheduling with limited resources: A zero-one programming approach. Management Science, 1969, vol. 16, no. 1, pp. 93-107.
[12]. Blazewicz J., Lenstra J. K., Rinnooy Kan A. H. G. Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics, 1983, vol. 5, pp. 11-24.
[13]. Ahuja H. N. Construction performance control by networks. John Wiley & Sons, New York, USA, 1976, 636 p.
[14]. Davis E. W., Patterson J. H. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. Management Science, 1975, vol. 21, no. 8, pp. 944-955.
[15]. Hegazy T. Optimization of resource allocation and leveling using genetic algorithms. Journal of Construction Engineering and Management, 1999, vol. 125, no. 3, pp. 167-175.
[16]. Brooks G. N., White C. R. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. Journal of Industrial Engineering, 1966, vol. 17, no. 2, pp. 173-186.
[17]. Gomory R. E. An all integer programming algorithm. Chapter from book [[157]], pp. 193-206.
[18]. Szwarg W. Solutions of the akers-friedman scheduling problem. Operations Research, 1960, vol. 8, no. 6, pp. 782-788.
[19]. Ashour S. A Decomposition Approach for the Machine Scheduling Problem. University of Iowa, Iowa City, USA, 1967, 416 p.
[20]. Giffler B., Thompson G. L., Van Ness V. Numerical experience with the linear and Monte-Carlo algorithms for solving production scheduling problems. Chapter from book [[157]], pp. 21-29.
[21]. Page E. S. An approach to the scheduling of the N jobs on M machines. Journal of the Royal Statistical Society, 1961, vol. 23, no. 2, pp. 484-492.
[22]. Kolisch R. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management, 1996, vol. 14, no. 3, pp. 179-192.

[23]. Boctor F. F. Some efficient multi-heuristic procedures for resource constrained project scheduling. European Journal of Operational Research, 1990, vol. 49, no. 1, pp. 3-13.
[24]. Graham R. L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 1979, vol. 5, pp. 287-326.
[25]. Brucker P., Drexl A., Möhring R., Neumann K., Pesch E. Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research, 1999, vol. 112, no. 1, pp. 3-41.
[26]. Słowin´ski R. Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. European Journal of Operational Research, 1981, vol. 7, no. 3, pp. 265-273.
[27]. Weglarz J. On certain models of resource allocation problems. Kybernetes, 1980, vol. 9, no. 1, pp. 61-66.
[28]. Böttcher J., Drexl A., Kolisch R., Salewski F. Project scheduling under partially renewable resource constraints. Management Science, 1999, vol. 45, no. 4, pp. 543-559.
[29]. Zhu G., Bard J. F., Yu G. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS Journal on Computing, 2006, vol. 18, no. 3, pp. 377-390.
[30]. Nudtasomboon N., Randhawa S. U. Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs. Computers and Industrial Engineering, 1997, vol. 32, no. 1, pp. 227-242.
[31]. Neumann K., Schwindt C. Project scheduling with inventory constraints. Mathematical Methods of Operations Research, 2003, vol. 56, no. 3, pp. 513-533.
[32]. Bartels J. H., Zimmermann J. Scheduling tests in automotive R&D projects. European Journal of Operational Research, 2009, vol. 193, no. 3, pp. 805-819.
[33]. Schwindt C., Trautmann N. Batch scheduling in process industries: An application of resource-constrained project scheduling. OR Spectrum, 2000, vol. 22, no. 4, pp. 501-524.
[34]. Neumann K., Schwindt C., Trautmann N. Scheduling of continuous and discontinuous material flows with intermediate storage restrictions. European Journal of Operational Research, 2005, vol. 165, no. 2, pp. 495-509.
[35]. Weglarz J., Blazewicz J., Cellary W., Słowin´ski R. Algorithm 520: An automatic revised simplex method for constrained resource network scheduling. ACM Transactions on Mathematical Software, 1977, vol. 3, no. 3, pp. 295-300.
[36]. Blazewicz J., Ecker K., Pesch E., Schmidt G., Weglarz J. Handbook on Scheduling: From Theory to Applications. Series: International Handbooks on Information Systems, Springer, Berlin, Germany, 2007, 647 p.
[37]. Kis T. A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. Mathematical Programming, 2005, vol. 103, no. 3, pp. 515-539.
[38]. Waligora G. Discrete-continuous project scheduling with discounted cash flows – A tabu search approach. Computers and Operations Research, 2008, vol. 35, no. 7, pp. 2141-2153.
[39]. Kis T. RCPS with Variable intensity activities and feeding precedence constraints. Chapter from book [[158]], pp. 105-129.
[40]. Jozefowska J., Mika M., Rozycki R., Waligora G., Weglarz J. Solving the discrete-continuous project scheduling problem via its discretization. Mathematical Methods of Operations Research, 2000, vol. 52, no. 3, pp. 489-499.
[41]. Weglarz J. Project scheduling with continuously divisible, doubly-constrained resources. Management Science, 1981, vol. 27, no. 9, pp. 1040-1053.

[42]. Bianco L., Dell'olmo P., Speranza M. G. Heuristics for multimode scheduling problems with dedicated resources. European Journal of Operational Research, 1998, vol. 107, no. 2, pp. 260-271.
[43]. Bianco L., Caramia M., Dell'olmo P. Solving a preemptive project scheduling problem with coloring techniques. Chapter from book [[159]], pp. 135-146.
[44]. Dorndorf U., Phan-Huy T., Pesch E. A survey of interval capacity consistency tests for time- and resource constrained scheduling. Chapter from book [[159]], pp. 213-238.
[45]. Dorndorf U., Pesch E., Phan-Huy T. Constraint propagation techniques for the disjunctive scheduling problem. Artificial Intelligence, 2000, vol. 122, no. 1-2, pp. 189-240.
[46]. Bomsdorf F., Derigs U. A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem. OR-Spectrum, 2008, vol. 30, no. 4, pp. 751-772.
[47]. Klein R., Scholl A. Scattered branch and bound – An adaptive search strategy applied to resource-constrained project scheduling. Central European Journal of Operations Research, 1999, vol. 7, pp. 177-201.
[48]. Nonobe K., Ibaraki T. Formulation and tabu search algorithm for the resource constrained project scheduling problem. Chapter (pp. 557-588) from book Ribeiro C. C., Hansen P. (eds.) Essays and Surveys in Metaheuristics. Series: Operations Research/Computer Science Interfaces Series, vol. 15, Springer, Berlin, Germany, 2002, 664 p.
[49]. Pesch E. Lower bounds in different problem classes of project schedules with resource constraints. Chapter from book [[159]], pp. 53-76.
[50]. Klein R., Scholl A. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. European Journal of Operational Research, 1999, vol. 112, no. 2, pp. 322-346.
[51]. Klein R. Project scheduling with time-varying resource constraints. International Journal of Production Research, 2000, vol. 38, no. 16, pp. 3937-3952.
[52]. Hartmann S. Project scheduling under limited resources: Models, methods, and applications. Series: Lecture Notes in Economics and Mathematical Systems, vol. 478, Springer, Berlin, Germany, 1999, 221 p.
[53]. Icmeli O., Rom W. O. Solving the resource constrained project scheduling problem with optimization subroutine library. Computers and Operations Research, 1996, vol. 23, no. 8, pp. 801-817.
[54]. Bartusch M., Möhring R. H., Radermacher F. J. Scheduling project networks with resource constraints and time windows. Annals of Operations Research, 1988, vol. 16, no. 1, pp. 201-240.
[55]. Debels D., Vanhoucke M. The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects. Computers and Industrial Engineering, 2008, vol. 54, no. 1, pp. 140-154.
[56]. Demeulemeester E. L., Herroelen W. S. An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. European Journal of Operational Research, 1996, vol. 90, no. 2, pp. 334-348.
[57]. Ballestin F., Valls V., Quintanilla S. Pre-emption in resource-constrained project scheduling. European Journal of Operational Research, 2008, vol. 189, no. 3, pp. 1136-1152.
[58]. Buddhakulsomsiria J., Kim D. S. Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research, 2006, vol. 175, no. 1, pp. 279-295.

[59]. Buddhakulsomsiria J., Kim D. S. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research, 2007, vol. 178, no. 2, pp. 374-390.
[60]. Franck B., Neumann K., Schwindt C. Project scheduling with calendars. OR Spektrum, 2001, vol. 23, no. 3, pp. 325-334.
[61]. Drezet L. E., Billaut J. C. A project scheduling problem with labour constraints and time-dependent activities requirements. International Journal of Production Economics, 2008, vol. 112, no. 1, pp. 217-225.
[62]. Cavalcante C. C. B., pp. de Souza C., Savelsbergh M. W. P., Wang Y., Wolsey L. A. Scheduling projects with labor constraints. Discrete Applied Mathematics, 2001, vol. 112, no. 1-3, pp. 27-52.
[63]. Mika M., Waligora G., Weglarz J. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research, 2008, vol. 187, no. 3, pp. 1238-1250.
[64]. Mika M., Waligora G., Weglarz J. Modelling setup times in project scheduling. Chapter from book [[158]], pp. 131-165.
[65]. Drexl A., Nissen R., Patterson J. H., Salewski F. Progen/πx – An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. European Journal of Operational Research, 2000, vol. 125, no. 1, pp. 59-72.
[66]. Vanhoucke M. Setup times and fast tracking in resource-constrained project scheduling. Computers and Industrial Engineering, 2008, vol. 54, no. 4, pp. 1062-1070.
[67]. Elmaghraby S. E. Activity networks: Project planning and control by network models. Wiley, New York, USA, 1977, 443 p.
[68]. Alcaraz J., Maroto C., Ruiz R. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society, 2003, vol. 54, no. 6, pp. 614-626.
[69]. Bouleimen K., Lecocq H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research, 2003, vol. 149, no. 2, pp. 268-281.
[70]. Hartmann S. Project scheduling with multiple modes: A genetic algorithm. Annals of Operations Research, 2001, vol. 102, no. 1-4, pp. 111-135.
[71]. Jarboui B., Damak N., Siarry P., Rebai A. A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation, 2008, vol. 195, no. 1, pp. 299-308.
[72]. Jozefowska J., Mika M., Rozycki R., Waligora G., Weglarz J. Simulated annealing for multi-mode resource-constrained project scheduling. Annals of Operations Research, 2001, vol. 102, no. 1-4, pp. 137-155.
[73]. Özdamar L. A genetic algorithm approach to a general category project scheduling problem. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 1999, vol. 29, no. 1, pp. 44-59.
[74]. Calhoun K. M., Deckro R. F., Moore J. T., Chrissis J. W., Hove J. C. V. Planning and re-planning in project and production scheduling. Omega, 2002, vol. 30, no. 3, pp. 155-170.
[75]. B. de Reyck, Herroelen W. S. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research, 1999, vol. 119, no. 2, pp. 538-556.
[76]. Heilmann R. Resource-constrained project scheduling: A heuristic for the multi-mode case. OR Spektrum, 2001, vol. 23, no. 3, pp. 335-357.

[77]. Heilmann R. A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. European Journal of Operational Research, 2003, vol. 144, no. 2, pp. 348-365.
[78]. Li H., Womer K. Modeling the supply chain configuration problem with resource constraints. International Journal of Project Management, 2008, vol. 26, no. 6, pp. 646-654.
[79]. Tiwari V., Patterson J. H., Mabert V. A. Scheduling projects with heterogeneous resources to meet time and quality objectives. European Journal of Operational Research, 2009, vol. 193, no. 3, pp. 780-790.
[80]. Tareghian H. R., Taheri S. H. A solution procedure for the discrete time, cost and quality tradeoff problem using electromagnetic scatter search. Applied Mathematics and Computation, 2007, vol. 190, no. 2, pp. 1136-1145.
[81]. Salewski F., Schirmer A., Drexl A. Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. European Journal of Operational Research, 1997, vol. 102, no. 1, pp. 88-110.
[82]. Demeulemeester E. L., B. de Reyck, Herroelen W. S. The discrete time/resource trade-off problem in project networks – A branch-and-bound approach. IIE Transactions, 2000, vol. 32, no. 11, pp. 1059-1069.
[83]. Ranjbar M., Kianfar F. Solving the discrete time/resource trade-off problem in project scheduling with genetic algorithms. Applied Mathematics and Computation, 2007, vol. 191, no. 2, pp. 451-456.
[84]. Ranjbar M., B. de Reyck, Kianfar F. A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. European Journal of Operational Research, 2009, vol. 193, no. 1, pp. 35-48.
[85]. Akkan C., Drexl A., Kimms A. Network decomposition-based benchmark results for the discrete time-cost tradeoff problem. European Journal of Operational Research, 2005, vol. 165, no. 2, pp. 339-358.
[86]. Demeulemeester E. L., B. de Reyck, Foubert B., Herroelen W. S., Vanhoucke M. New computational results on the discrete time/cost trade-off problem in project networks. Journal of the Operational Research Society, 1998, vol. 49, no. 11, pp. 1153-1163.
[87]. Deckro R. F., Hebert J. E., Verdini W. A., Grimsrud P. H., Venkateshwar S. Nonlinear time/cost tradeoff models in project management. Computers and Industrial Engineering, 1995, vol. 28, no. 2, pp. 219-229.
[88]. Chassiakos A. P., Sakellaropoulos S. P. Time-cost optimization of construction projects with generalized activity constraints. Journal of Construction Engineering and Management, 2005, vol. 131, no. 10, pp. 1115-1124.
[89]. Klein R., Scholl A. PROGRESS: Optimally solving the generalized resource-constrained project scheduling problem. Mathematical Methods of Operations Research, 2000, vol. 52, no. 3, pp. 467-488.
[90]. Kolisch R. Integrated scheduling, assembly area- and part-assignment for large-scale, make-to-order assemblies. International Journal of Production Economics, 2000, vol. 64, no. 1-3, pp. 127-141.
[91]. Vanhoucke M. Work continuity constraints in project scheduling. Journal of Construction Engineering and Management, 2006, vol. 132, no. 1, pp. 14-25.
[92]. Demeulemeester E. L., Herroelen W. S. Modelling setup times, process batches and transfer batches using activity network logic. European Journal of Operational Research, 1996, vol. 89, no. 2, pp. 355-365.
[93]. Cesta A., Oddi A., Smith S. F. A constraint-based method for project scheduling with time windows. Journal of Heuristics, 2002, vol. 8, no. 1, pp. 109-136.

[94]. Dorndorf U., Pesch E., Phan Huy T. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalized precedence constraints. Management Science, 2000, vol. 46, no. 10, pp. 1365-1384.
[95]. Schwindt C. Generation of resource-constrained project scheduling problems subject to temporal constraints. Technical Report WIOR-543, Universität Karlsruhe, Germany, 1998.
[96]. Neumann K., Zimmermann J. Exact and truncated branch-and-bound procedures for resource-constrained project scheduling with discounted cash flows and general temporal constraints. Central European Journal of Operations Research, 2002, vol. 10, no. 4, pp. 357-380.
[97]. Neumann K., Zimmermann J. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. European Journal of Operational Research, 2000, vol. 127, no. 2, pp. 425-443.
[98]. Neumann K., Schwindt C., Zimmermann J. Recent results on resource-constrained project scheduling with time windows: Models, solution methods, and applications. Central European Journal of Operations Research, 2002, vol. 10, no. 2, pp. 113-148.
[99]. Sabzehparvar M., Seyed-Hosseini S. M. A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. Journal of Supercomputing, 2008, vol. 44, no. 3, pp. 257-273.
[100]. Franck B., Neumann K. Resource-constrained project scheduling with time windows: Structural questions and priority rule methods. Technical Report WIOR-492, Universität Karlsruhe, Germany, 1997.
[101]. Ballestin F., Valls V., Quintanilla S. Due dates and RCPSP. Chapter from book [[158]], pp. 79-104.
[102]. Brânzei R., Ferrari G., Fragnelli V., Tijs S. Two approaches to the problem of sharing delay costs in joint projects. Annals of Operations Research, 2002, vol. 109, no. 1-4, pp. 359-374.
[103]. Chiu H. N., Tsai D. M. An efficient search procedure for the resource-constrained multi-project scheduling problem with discounted cash flows. Construction Management and Economics, 2002, vol. 20, no. 1, pp. 55-66.
[104]. Özdamar L., Ulusoy G., Bayyigit M. A heuristic treatment of tardiness and net present value criteria in resource constrained project scheduling. International Journal of Physical Distribution and Logistics Management, 1998, vol. 28, no. 9/10, pp. 805-824.
[105]. Baptiste P., Pape C. L., Nuijten W. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research, 1999, vol. 92, pp. 305-333.
[106]. Yang H. H., Chen Y. L. Finding the critical path in an activity network with time-switch constraints. European Journal of Operational Research, 2000, vol. 120, no. 3, pp. 603-613.
[107]. Vanhoucke M., Demeulemeester E. L., Herroelen W. S. Discrete time/cost trade-offs in project scheduling with time-switch constraints. Journal of the Operational Research Society, 2002, vol. 53, no. 7, pp. 741-751.
[108]. Krüger D., Scholl A. Managing and modelling general resource transfers in (multi-)project scheduling. OR Spektrum, 2010, vol. 32, no. 2, pp. 369-394.
[109]. Elmaghraby S. E. An algebra for the analysis of generalized activity networks. Management Science, 1964, vol. 10, no. 3, pp. 494-514.
[110]. Belhe U., Kusiak A. Resource-constrained scheduling of hierarchically structured design activity networks. IEEE Transactions on Engineering Management, 1995, vol. 42, no. 2, pp. 150-158.

[111]. Neumann K. Stochastic project networks: Temporal analysis, scheduling and cost minimization. Series: Lecture Notes in Economics and Mathematical Systems, vol. 344, Springer, Berlin, Germany, 1990, 237 p.
[112]. Kuster J., Jannach D. Handling airport ground processes based on resource-constrained project scheduling. Lecture Notes in Computer Science, 2006, vol. 4031, pp. 166-176.
[113]. Viana A., P. de Sousa J. Using metaheuristics in multiobjective resource constrained project scheduling. European Journal of Operational Research, 2000, vol. 120, no. 2, pp. 359-374.
[114]. Vanhoucke M., Demeulemeester E. L., Herroelen W. S. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. Annals of Operations Research, 2001, vol. 102, no. 1-4, pp. 179-196.
[115]. Franck B., Schwindt C. Different resource-constrained project scheduling models with minimal and maximal time-lags. Technical Report WIOR-450, Universität Karlsruhe, Germany, 1995.
[116]. Lorenzoni L. L., Ahonen H., G. de Alvarenga A. A multi-mode resource-constrained scheduling problem in the context of port operations. Computers and Industrial Engineering, 2006, vol. 50, no. 1-2, pp. 55-65.
[117]. Vanhoucke M. Scheduling an R&D project with quality-dependent time slots. Lecture Notes in Computer Science, 2006, vol. 3982, pp. 621-630.
[118]. Rom W. O., Tukel O. I., Muscatello J. R. MRP in a job shop environment using a resource constrained project scheduling model. Omega, 2002, vol. 30, no. 4, pp. 275-286.
[119]. Nazareth T., Verma S., Bhattacharya S., Bagchi A. The multiple resource constrained project scheduling problem: A breadth-first approach. European Journal of Operational Research, 1999, vol. 112, no. 2, pp. 347-366.
[120]. Al-Fawzan M., Haouari M. A bi-objective model for robust resource-constrained project scheduling. International Journal of Production Economics, 2005, vol. 96, no. 2, pp. 175-187.
[121]. Abbasi B., Shadrokh S., Arkat J. Bi-objective resource-constrained project scheduling with robustness and makespan criteria. Applied Mathematics and Computation, 2006, vol. 180, no. 1, pp. 146-152.
[122]. Kobylanski P., Kuchta D. A note on the paper by M. A. Al-Fawzan and M. Haouari about a bi-objective problem for robust resource-constrained project scheduling. International Journal of Production Economics, 2007, vol. 107, no. 2, pp. 496-501.
[123]. Chtourou H., Haouari M. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. Computers and Industrial Engineering, 2008, vol. 55, no. 1, pp. 183-194.
[124]. Icmeli-Tukel O., Rom W. O. Ensuring quality in resource constrained project scheduling. European Journal of Operational Research, 1997, vol. 103, no. 3, pp. 483-496.
[125]. Van de Vonder S., Demeulemeester E. L., Herroelen W. S. A classification of predictive-reactive project scheduling procedures. Journal of Scheduling, 2007, vol. 10, no. 3, pp. 195-207.
[126]. Sakkout H., Wallace M. Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints, 2000, vol. 5, no. 4, pp. 359-388.
[127]. Zhu G., Bard J. F., Yu G. Disruption management for resource-constrained project scheduling. Journal of the Operational Research Society, 2005, vol. 56, pp. 365-381.

[128]. Neumann K., Schwindt C., Zimmermann J. Resource-constrained project scheduling with time windows: Recent developments and new applications. Chapter from book [[158]], pp. 375-407.
[129]. Drexl A., Kimms A. Optimization guided lower and upper bounds for the resource investment problem. Journal of the Operational Research Society, 2001, vol. 52, no. 3, pp. 340-351.
[130]. Ranjbar M., Kianfar F., Shadrokh S. Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm. Applied Mathematics and Computation, 2008, vol. 196, no. 2, pp. 879-888.
[131]. Yamashita D. S., Armentano V. A., Laguna M. Robust optimization models for project scheduling with resource availability cost. Journal of Scheduling, 2007, vol. 10, no. 1, pp. 67-76.
[132]. Shadrokh S., Kianfar F. A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. European Journal of Operational Research, 2007, vol. 181, no. 1, pp. 86-101.
[133]. Nübel H. The resource renting problem subject to temporal constraints. OR-Spektrum, 2001, vol. 23, no. 3, pp. 359-381.
[134]. Ballestin F. A genetic algorithm for the resource renting problem with minimum and maximum time lags. Lecture Notes in Computer Science, 2007, vol. 4446, pp. 25-35.
[135]. Bandelloni M., Tucci M., Rinaldi R. Optimal resource leveling using non-serial dynamic programming. European Journal of Operational Research, 1994, vol. 78, no. 2, pp. 162-177.
[136]. Davis K. R., Stam A., Grzybowski R. A. Resource constrained project scheduling with multiple objectives: A decision support approach. Computers and Operations Research, 1992, vol. 19, no. 7, pp. 657-669.
[137]. Maniezzo V., Mingozzi A. The project scheduling problem with irregular starting time costs. Operations Research Letters, 1999, vol. 25, no. 4, pp. 175-182.
[138]. Möhring R. H., Schulz A. S., Stork F., Uetz M. On project scheduling with irregular starting time costs. Operations Research Letters, 2001, vol. 28, no. 4, pp. 149-154.
[139]. Möhring R. H., Schulz A. S., Stork F., Uetz M. Solving project scheduling problems by minimum cut computations. Management Science, 2003, vol. 49, no. 3, pp. 330-350.
[140]. Achuthan N., Hardjawidjaja A. Project scheduling under time dependent costs – A branch and bound algorithm. Annals of Operations Research, 2001, vol. 108, no. 1-4, pp. 55-74.
[141]. Dodin B., Elimam A. A. Integrated project scheduling and material planning with variable activity duration and rewards. IIE Transactions, 2001, vol. 33, no. 11, pp. 1005-1018.
[142]. Nonobe K., Ibaraki T. A metaheuristic approach to the resource constrained project scheduling with variable activity durations and convex cost functions. Chapter from book [[158]], pp. 225-248.
[143]. Rummel J. L., Walter Z., Dewan R., Seidmann A. Activity consolidation to improve responsiveness. European Journal of Operational Research, 2005, vol. 161, no. 3, pp. 683-703.
[144]. Kimms A. Maximizing the net present value of a project under resource constraints using a lagrangian relaxation based heuristic with tight upper bounds. Annals of Operations Research, 2001, vol. 102, no. 1-4, pp. 221-236.
[145]. Mika M., Waligora G., Weglarz J. Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and

different payment models. European Journal of Operational Research, 2005, vol. 164, no. 3, pp. 639-668.
[146]. Padman R., Zhu D. Knowledge integration using problem spaces: A study in resource-constrained project scheduling. Journal of Scheduling, 2006, vol. 9, no. 2, pp. 133-152.
[147]. Vanhoucke M., Demeulemeester E. L., Herroelen W. S. On maximizing the net present value of a project under renewable resource constraints. Management Science, 2001, vol. 47, no. 8, pp. 1113-1121.
[148]. Varma V. A., Uzsoy R., Pekny J., Blau G. Lagrangian heuristics for scheduling new product development projects in the pharmaceutical industry. Journal of Heuristics, 2007, vol. 13, no. 5, pp. 403-433.
[149]. Najafi A. A., Niaki S. T. A. A genetic algorithm for resource investment problem with discounted cash flows. Applied Mathematics and Computation, 2006, vol. 183, no. 2, pp. 1057-1070.
[150]. Herroelen W. S., P. van Dommelen, Demeulemeester E. L. Project network models with discounted cash flows: A guided tour through recent developments. European Journal of Operational Research, 1997, vol. 100, no. 1, pp. 97-121.
[151]. Dayanand N., Padman R. On modelling payments in projects. Journal of the Operational Research Society, 1997, vol. 48, no. 9, pp. 906-918.
[152]. Etgar R., Shtub A., LeBlanc L. J. Scheduling projects to maximize net present value – the case of time-dependent, contingent cash flows. European Journal of Operational Research, 1997, vol. 96, no. 1, pp. 90-96.
[153]. Hapke M., Jaszkiewicz A., Slowinski R. Interactive analysis of multiple-criteria project scheduling problems. European Journal of Operational Research, 1998, vol. 107, no. 2, pp. 315-324.
[154]. Slowinski R., Soniewicki B., Weglarz J. DSS for multiobjective project scheduling. European Journal of Operational Research, 1994, vol. 79, no. 2, pp. 220-229.
[155]. Dörner K. F., Gutjahr W. J., Hartl R. F., Strauss C., Stummer C. Nature-inspired metaheuristics for multiobjective activity crashing. Omega, 2008, vol. 36, no. 6, pp. 1019-1037.
[156]. Nabrzynski J., Weglarz J. Knowledge-based multiobjective project scheduling problems. Chapter from book [[159]], pp. 383-411.
[157]. Muth J. F., Thompson G. L. (eds.) Industrial Scheduling. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1963, 467 p.
[158]. Jozefowska J., Weglarz J. (eds.) Perspectives in Modern Project Scheduling. Springer Science & Business Media, New York, USA, 2006, 466 p.
[159]. Weglarz J. (eds.) Project Scheduling: Recent Models, Algorithms and Applications. Springer, New York, USA, 1999, 535 p.

Прототип интегрированной программной платформы для сопровождения вычислительного эксперимента в комплексных задачах математического моделирования [1]

М.П. Галанин <Galan@keldysh.ru>
М.М. Горбунов-Посадов <Gorbunov@keldysh.ru>
А.В. Ермаков <Ermakov@keldysh.ru>
В.В. Лукин <VVLukin@gmail.com>
А.С. Родин <rals@bk.ru>
К.Л. Шаповалов <ShapovalovKL@yandex.ru>
ИПМ им. М.В. Келдыша РАН, 125047, Москва, Миусская пл., д.4

Аннотация. Представлена архитектура, интерфейсы и форматы входных и выходных данных платформы, реализованные пилотные версии инструментальных и проблемно-ориентированных модулей программной платформы Теметос для сопровождения вычислительного эксперимента. Платформа Теметос предназначена для полномасштабного математического моделирования в различных прикладных и фундаментальных областях. Она предоставляет возможности подготовки геометрической и физической моделей исследуемой конструкции или физического процесса к расчету, инструменты настройки внешних или встроенных проблемно-ориентированных модулей, позволяет осуществлять запуск, контроль прохождения и анализ результатов расчетов.

Ключевые слова: математическое моделирование; вычислительный эксперимент; программная архитектура; интегрированная программная платформа; неоднородные среды.

1. Введение

Для решения широкого класса инженерных и прикладных научных задач становятся все более востребованными физически, математически и

[1] Работа выполнена при частичной финансовой поддержке РФФИ (проекты №№ 12-01-31193, 12-01-00109, 12-02-00687), а также гранта по поддержке ведущих научных школ НШ-1434.2012.2. Авторы выражают благодарность А.В. Плеханову, М.А. Семериковой, П.В. Тюфягину за сотрудничество в выполнении данной работы.

вычислительно сложные модели, требующие применения высокоточных методов [1] . Подобные методы ведут к большому объёму вычислительной работы, поэтому математическое моделирование в научно-технических приложениях требует высокой эффективности разрабатываемых алгоритмов, включая распараллеливание вычислений.

С другой стороны, многие этапы построения, программной реализации и исследования математической модели (задание геометрии области, построение сеток, ввод начальных и граничных данных модели, запуск и отслеживание работы параллельной версии расчетных процедур, визуализация и анализ полученного решения) являются общими для задач из различных областей теории и практики. Важным в такой ситуации становится создание интегрированной программной платформы, позволяющей задействовать высокоэффективные (в том числе параллельные) численные алгоритмы для решения задач различной размерности, содержащих уравнения разных типов и разного физического содержания (в рамках связанных задач).

Существует большое количество вычислительных программных платформ, позволяющих решать стандартные задачи математической физики в различных областях инженерной и прикладной научной практики. Большая часть из них является коммерческой (такие как ANSYS [2] , Nastran, Abaqus, Fluent и ряд других), имеет продвинутый пользовательский интерфейс, включает пре- и постпроцессор, а также множество закрытых проблемно-ориентированных *модулей-решателей* для моделирования термомеханических, газовых и других процессов с применением метода конечных элементов (МКЭ) и (реже) метода конечных разностей (МКР). Обычно подобные системы создаются как универсальные, применимые для широкого круга практически важных областей. Универсальность подобных программных платформ приводит к большим трудностям при попытках решения в рамках таких пакетов физически и математически сложных задач в областях, включающих разного рода неоднородности физических параметров или геометрических характеристик. При этом, как правило, квалифицированноый пользователь не имеет возможности эффективным образом включать в состав подобного программного комплекса собственные проблемно-ориентированные программные модули, поскольку многие из интерфейсов обработки геометрии области и исходных данных являются также закрытыми.

В последнее время активно развиваются открытые программные платформы для проведения инженерных и, реже, научных расчетов (например, OpenFOAM [3], Salome, Code-Aster, Code Athena), создаваемые сообществами программистов из разных стран, зачастую не объединенных в коммерческие корпорации. Подобные программные пакеты изначально строятся на открытых интерфейсах обработки данных, обеспечивая универсальность применения пакета и позволяя группам разработч иков расширять возможности платформы за счет включения собственных МКЭ или МКР

расчетных кодов. В то же время в рамках подобных платформ реализованы, как правило, наиболее стандартные подходы к решению задач математической физики без учета особенностей конкретных постановок практически значимых задач. Среди таких особенностей – необходимость применения методов повышенного порядка аппроксимации, контроль консервативности расчетной схемы и др., т.е. все то, что чаще всего отличает авторские прикладные программные комплексы, разрабатываемые под специальные задачи. Кроме того, задание геометрии области, начальных и краевых данных в подобных платформах часто необходимо осуществлять без использования графической оболочки путем правки текстовых конфигурационных файлов, что существенно усложняет вспомогательные технические вопросы математического моделирования.

В данной статье представлены результаты разработки прототипа интегрированной программной платформы для сопровождения вычислительного эксперимента в комплексных задачах математического моделирования – кроссплатформенной графической инструментальной среды Теметос. Платформа предназначена для построения и исследования математических моделей объектов и процессов, включая моделирование электромагнитных, тепловых, упругих и газодинамических полей в областях, содержащих геометрически сложные подобласти с резко неоднородными свойствами. Основная цель разработки платформы заключается в создании программного окружения, позволяющего с помощью ряда открытых интерфейсов передачи данных и управления расчетом оснастить авторский проблемно-ориентированный модуль-решатель инструментами подготовки, проведения и анализа результатов вычислительного эксперимента. В качестве базовых решателей для платформы адаптированы следующие два модуля.

1. Параллельный программный комплекс для решения двумерных задач идеальной магнитной гидродинамики разрывным методом Галеркина (RKDG-метод) [4]. Модуль используется для моделирования процессов развития магнито-ротационной неустойчивости в околозвездной плазме.

2. Термомеханический решатель [5], позволяющий, в частности, исследовать прочность конструкций в рамках связанной термоупругопластической задачи с разрушением [6]. Модуль используется для численного исследования работы магнитного компрессора, а также моделирования элементов конструкций, испытывающих высокие тепловые нагрузки.

2. Общая архитектура платформы Теметос

Блок-схема платформы сопровождения вычислительного эксперимента состоит из групп процедур, представленных на рис. 1.

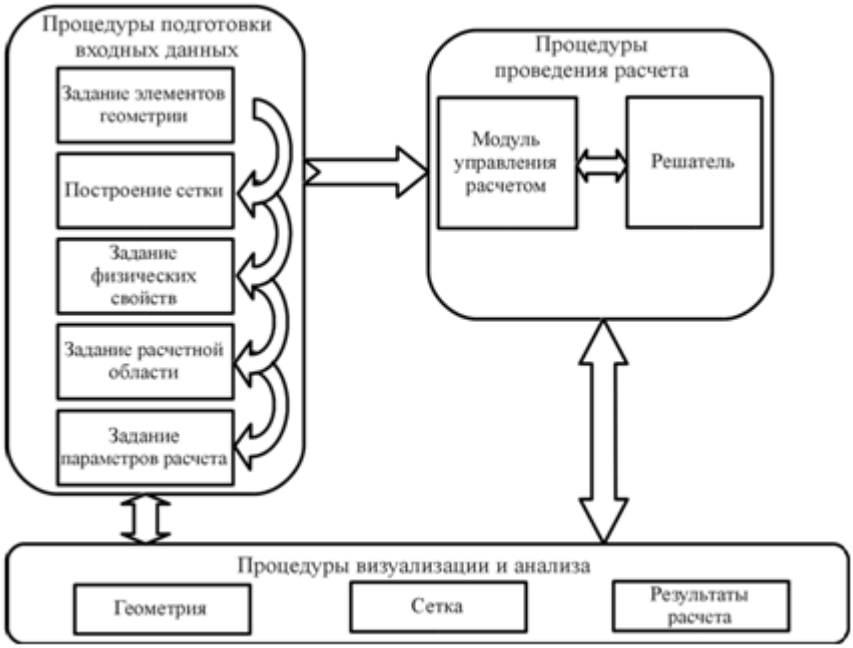

Рис. 1. Процедурная блок-схема платформы Темetос.

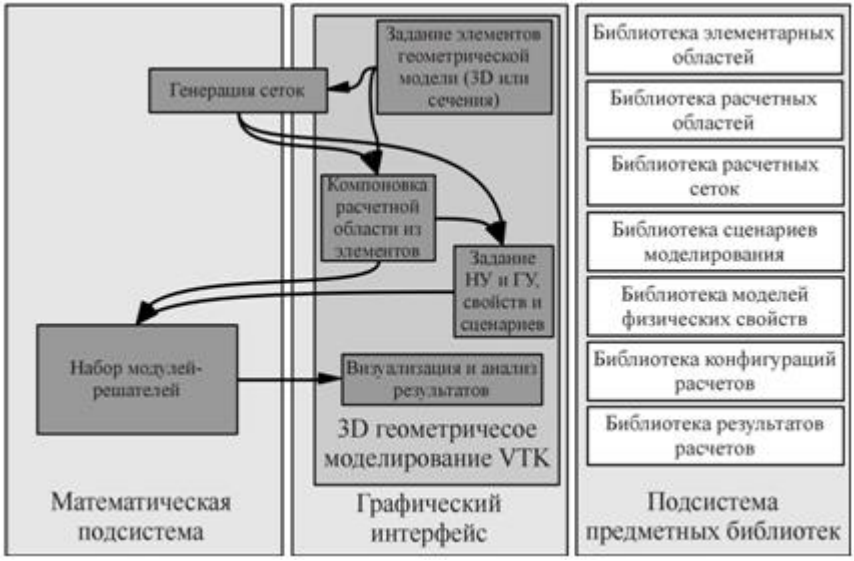

Рис. 2. Подсистемы платформы Темetос.

В соответствии с указанным разбиением основных процедур моделирования на группы программная платформа Теметос состоит из трех основных подсистем (см. рис. 2).

1. *Математическая подсистема*. Включает в себя основные средства проведения моделирующих расчетов, прежде всего набор программ-решателей, реализующих выбранные методы решения систем уравнений той или иной предметной области: уравнений магнитной гидродинамики, связанной термомеханики и др. Также подсистема содержит набор программ-построителей сеток (двумерных треугольных и четырехугольных; трехмерных). Подсистема состоит из набора консольных утилит. При этом утилиты могут быть как встроенными в платформу, так и исходно внешними (независимыми). Для этого разработаны интерфейсы, обеспечивающие трансформацию данных, которыми оперирует платформа, в форматы входных данных модулей-решателей.

2. *Инструментальная графическая подсистема*. Обеспечивает визуальную (трехмерную) подготовку геометрической модели рассматриваемого объекта или процесса, в случае необходимости – компоновку сложной расчетной области из простых подобластей, задание начальных и граничных условий, запуск расчета, визуализацию полученных результатов, а также взаимодействие с полным набором используемых для моделирования предметных библиотек. С программной точки зрения подсистема является кроссплатформенной, написана на языке C++ с использованием программной библиотеки оконного интерфейса Qt [8] и технологии трехмерной визуализации VTK [9] . Перечисленные библиотеки и технологии являются открытыми и свободнораспространяемыми.

3. *Подсистема предметных библиотек*. Содержит структурированную пополняемую иерархическую базу знаний о моделируемых объектах и физических процессах, в частности, наборы исследуемых расчетных областей и соответствующих им сеток, реализованные в виде программных модулей модели физических свойств материалов и сред, а также средства и стандарты взаимодействия библиотек и работы с ними. В основу подсистемы положен стандарт описания данных XML. Структура библиотек фиксирована в файловой системе платформы Теметос.

В процессе расчета взаимодействие пользователя с каждой из подсистем может осуществляться как через графическую среду, так и непосредственно. Осуществление вычислительного эксперимента для конкретной математической модели в платформе Теметос представляется в виде последовательной работы с рядом библиотек, осуществляемой в рамках единого интерфейса.

3. Физическая подсистема предметных библиотек

Выбор программных средств при разработке интегрированной платформы сопровождения вычислительного эксперимента направлен на обеспечение возможности простого расширения набора моделируемых объектов, физических и математических моделей, используемых сеток, модулей-решателей. Для этого используется концепция безболезненного роста программы [7] , причем точки роста платформы предполагается оформлять в виде предметных библиотек, которые могут содержать как текстовые (XML) файлы описания данных, так и скомпилированные программные библиотеки, подключаемые к платформе при помощи стандартизуемых интерфейсов.

Информационная структура платформы Теметос основана на системе расширяемых предметных библиотек, призваных охватыватить базу знаний о классе моделируемых объектов и явлений. В рамках данной базы осуществляется поддержка задания геометрически элементарных составляющих области моделирования, компоновки сложных расчетных областей, реализации моделей физических свойств сред и материалов, определения сценариев моделирования (начальных и граничных условий, временнх и пространственных распределений источников и стоков и проч.). Пополнение предметных библиотек позволяет описывать и моделировать полный набор исследуемых объектов. Наполнение библиотек определяется прежде всего областью моделирования и, соответственно, используемым решателем. Содержимое библиотек хранится в файлах расширяемого формата представления данных XML, что делает их структуру прозрачной и позволяет осуществлять их редактирование и пополнение как с помощью разработанных редакторов библиотек, так и вручную в любом текстовом редакторе.

Элементы библиотек связаны друг с другом принципом соответствия, подобным применяемому в реляционных базах данных. Разработана система работы с набором предметных библиотек, позволяющая на каждой стадии подготовки расчета конструкции получать полную информацию о всех составляющих элементах модели.

В состав платформы входят следующие библиотеки.

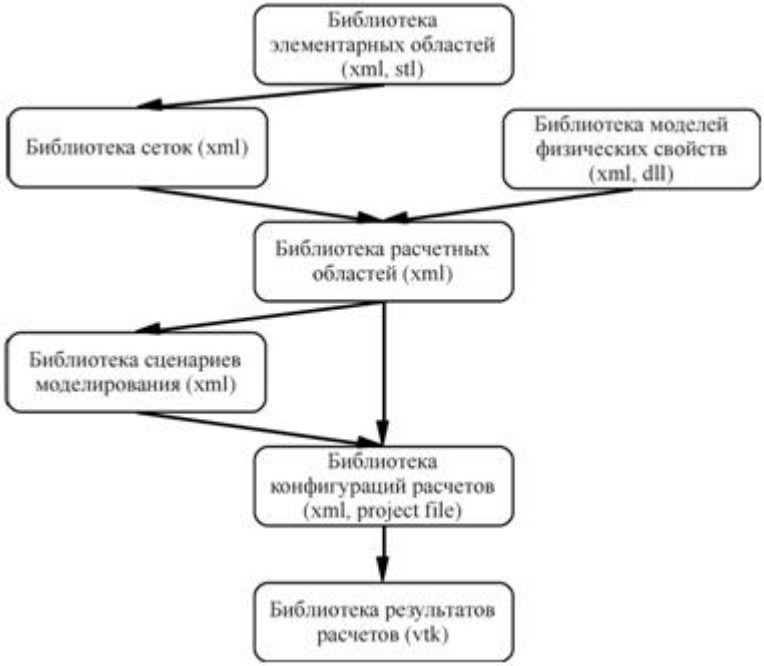

Рис. 3. Иерархия предметных библиотек.

1. Геометрическая библиотека элементарных областей (БЭО), содержащая наиболее общее (векторное) описание используемых при моделировании геометрических областей (в случае инженерных расчетов – элементов конструкций: лопаток, шестерней и проч.).
2. Библиотека сеток (БС), содержащая сетки, заданные в элементарных областях из БЭО.
3. Библиотека моделей физических свойств (БМФС), включающая как xml-описание физических свойств моделируемых материалов и сред в простых случаях, так и скомпилированные dll-библиотеки подпрограмм, описывающих сложное поведение параметров среды (теплопроводности, пористости, электрической проводимости и др.); набор доступных для описания физических параметров определяется возможностями конкретного модуля-решателя.
4. Библиотека расчетных областей (БРО), содержащая скомпонованные расчетные области, состоящие из относительно простых элементов, хранящихся в БЭО, вместе с выбранными для каждого элемента сеткой из БС и заданными моделями среды из БМФС.
5. Библиотека сценариев моделирования (БСМ), содержащая наборы входных данных, начальных и граничных условий, временных и

пространственных распределений источников для расчета объекта, компоновка которого содержится в БРО.

6. Библиотека конфигураций расчетов (БКР), содержащая наборы настроек модуля-решателя, включая составляющие решаемых систем уравнений, настройки численных методов и алогоритмов (в том числе параллельных), применяемых к расчету объекта из БРО с заданным сценарием моделирования из БСМ.
7. Библиотека результатов расчетов (БРР), содержащая полный набор выходных данных решателя при проведении расчета из БКР.

Иерархические связи предметных библиотек показаны на рис. 3. Взаимосвязи элементов различных библиотек применительно к конкретному объекту моделирования и конкретной постановке расчетной задачи фиксируются в xml-файле проекта.

4. Инструментальная графическая подсистема и алгоритм работы с платформой

Задача платформы Теметос – оснастить исследователя, разрабатывающего собственный или использующего готовый проблемно-ориентированный модуль-решатель, максимально удобными (прежде всего, графическими) средствами подготовки и проведения вычислительного эксперимента с применением этого решателя. В состав платформы включены графические редакторы предметных библиотек, а также инструменты, позволяющие «собрать» из заранее подготовленных единиц (геометрических моделей, моделей физических свойств, сеток, конфигураций решателя) расчет, произвести его запуск и отслеживание, проанализировать его результаты.

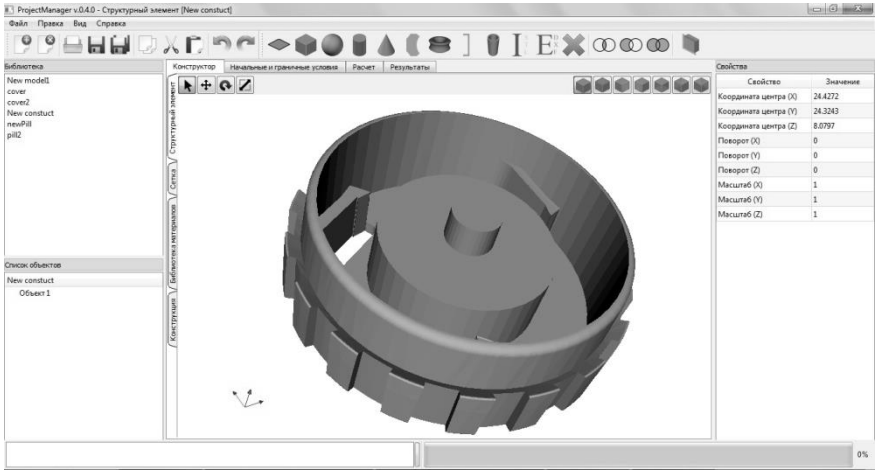

Рис. 4. Редактор элементарных трехмерных областей платформы Теметос.

Сразу после запуска платформы пользователь попадает в редактор элементарных трехмерных областей (рис. 4). Описание геометрии расчетной области исследуемого объекта ведется на различных, но взаимоувязанных, уровнях. Полная расчетная область описывается в два этапа:

- задание чертежей элементарных областей в виде трехмерных или плоских областей (с учетом осевой симметрии) с указанием всех размеров и подробным заданием формы каждого элемента;
- формирование из предварительно подготовленного набора элементарных областей своеобразной схемы-сборки расчетной области с указанием положения и набора физических моделей для каждого из элементов.

Подобный подход оказывается удобным в ряде случаев моделирования конструкций. В частности, численный алгоритм термомеханического решателя с учетом контакта и мультиконтакта упругих тел позволяет моделировать различные элементы инженерной конструкции как независимые твердые тела, взаимодействующие через поверхности контакта. Это позволяет независимо задавать геометрию, физические и модельные свойства каждого элемента, и моделирование конструкции в целом начинается именно с проектирования (введения в соответствующие библиотеки) индивидуальных элементов с наиболее подробным учетом особенностей их формы. Реализован импорт трехмерных областей, заданных в формате STL, импорт в который осуществляется из многих популярных CAD-систем.

Окно графической подсистемы меняется в зависимости от того, с каким именно блоком в данный момент работает пользователь. Работа с каждым из блоков представляет собой просмотр, редактирование и выбор для расчета элементов соответствующей блоку предметной библиотеки.

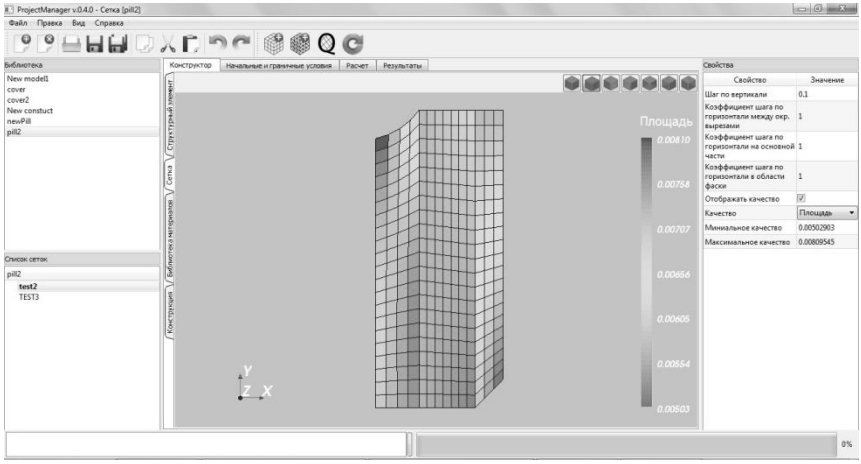

Рис. 5. Задание треугольной сетки в произвольной области.

Для каждой элементарной подобласти пользователь может задать одну или несколько сеток (рис. 5), причем одна и та же подобласть может входить в состав расчетной области многократно, имея при этом разные сетки. Платформа Теметос позволяет визуально и численно (используя ряд критериев) оценить качество сетки и при необходимости провести ее оптимизацию.

Далее пользователь производит выбор физических моделей сред и материалов, которые будут использоваться в расчете. Каждая среда представляется набором физических свойств, каждое из которых может быть задано либо непосредственно численно (значения сохраняются в xml-файле среды), либо указанием функции в подключаемой dll-библиотеке (ссылка на библиотеку и функцию также указывается в xml-файле среды). Модуль-решатель должен поддерживать подключение внешних библиотек. В рамках работы над платформой разработан унифицированный интерфейс подключения материальных библиотек, который может быть реализован в виде API платформы, доступного для использования во внешних решателях.

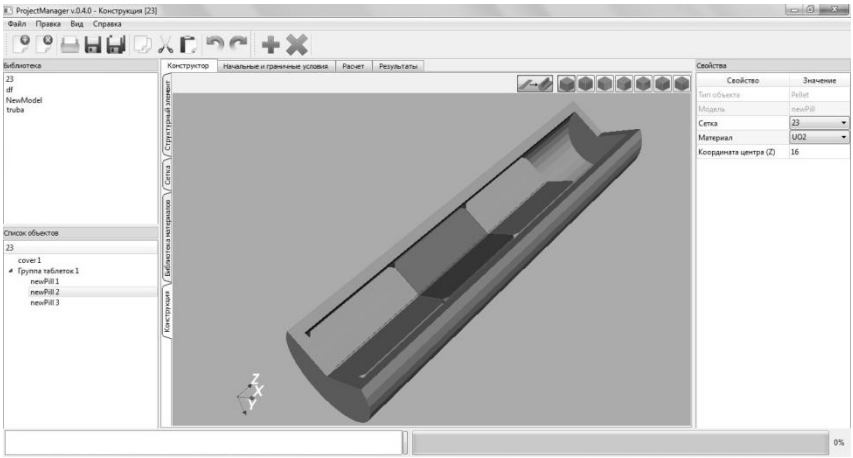

Рис. 6. Редактор расчетных областей – трехмерная форма отображения конструкции.

Из подготовленных элементарных подобластей с выбором для каждой из них сетки и набора физических свойств осуществляется сборка расчетной области в соответствующем редакторе (см. рис. 6). Далее для выбранной расчетной области (конструкции) задаются начальные и граничные условия, сценарий моделирования, распределения объемных и поверхностных источников. Для реализации этой возможности создан редактор сценариев, включающий парсер формул библиотеки VTK.

После того, как задана расчетная область, определены используемые модели материалов и сценарий моделирования, производится настройка и запуск на

расчет решателя. В соответствующем блоке платформы задаются параметры численного метода, используемые классы физических и математических моделей, осуществляется выбор моделируемой конструкции твэла, соответствующего ей сценария нагружения, а также общие параметры расчета, такие как конечное время, размерность задачи и проч. В процессе расчета предоставляется соответствующая телеметрическая информация.

Доступно как двумерное, так и трехмерное (в т.ч. двумерное осесимметричное) отображение результатов расчетов. Кроме того разработаны средства по трансформации способа отображения результатов – инструменты по созданию контурных графиков распределений физических величин, отображению векторных полей, соответствующих переменным задачи, а также генерации результирующей формы конструкции путем смещения узлов сетки на соответствующий вектор перемещения (см. рис. 7).

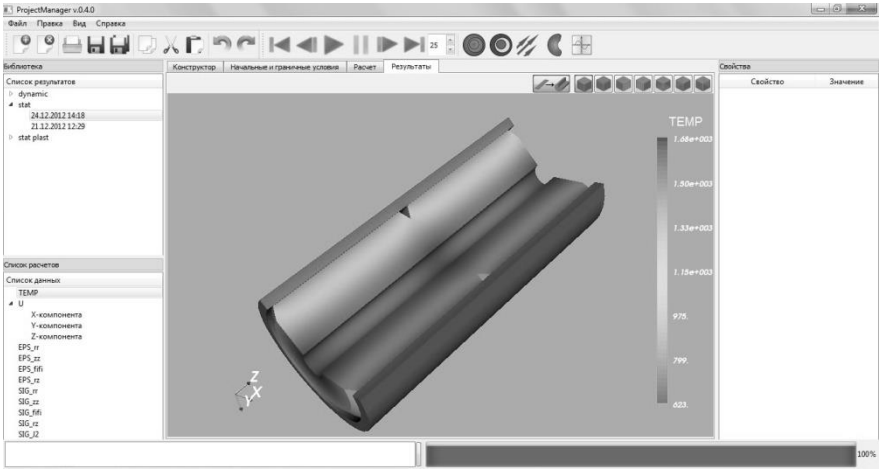

Рис. 7. Окно представления результатов расчетов.

5. Математическая подсистема

Математическая подсистема платформы Теметос включает основные расчетные программы и комплексы, используемые для проведения вычислительного эксперимента – программы-построители сеток и модули-решатели, реализующие методы численного исследования систем уравнений, соответствующих выбранной физической задаче. Архитектурой платформы Теметос предусмотрено, что каждая расчетная программа является независимой, обмен данными между расчетной программой и платформой производится только через файлы входных данных и настроек, при наличии входных файлов расчетная программа может быть запущена непосредственно из командной строки вне графической оболочки. При этом предполагается,

что порядок запуска расчетной программы и состав входных файлов описываются в отдельном интерфейсном файле xml-формата.

На данный момент платформа Теметос включает несколько построителей неструктурированных и структурированных сеток, причем каждый из них является независимым программным продуктом, а платформа содержит необходимые конвертеры для преобразования данных о геометрии в формат построителя и получаемых данных о сетке в другие используемые форматы. Поддерживается построение как неструктурированных треугольных (программа Gridder 2D [10]), так и четырехугольных сеток различных видов – как в областях общего вида, так и в специальных областях, соответствующих специфике задачи (параметрически заданные топливные таблетки в задаче о моделировании твэла).

На пилотном этапе разработке к платформе адаптируются два расчетных модуля – параллельный программный комплекс для решения двумерных задач идеальной магнитной гидродинамики RKDG-методом и термомеханический программный комплекс, включающий математические модели термоупругопластичного материала и хрупкого разрушения. Платформа применяется для задания расчетной области, запуска последовательного расчета и анализа получаемых результатов. Предполагается, что в дальнейшем в состав комплекса Теметос будут включены инструменты подготовки и проведения расчета на параллельных вычислительных комплексах, включая кластерные системы (К-100 ИПМ им. М.В. Келдыша РАН, УЭВК кафедры «Прикладная математика» МГТУ им. Н.Э. Баумана [11] и др.).

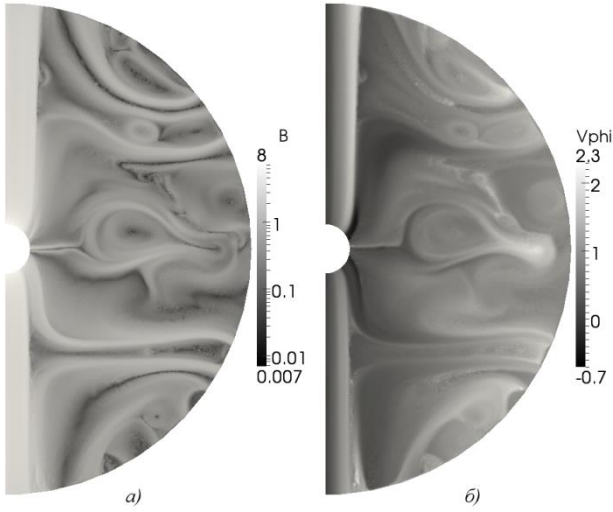

Рис. 8. Моделирование МРН в аккрецирующей плазме: модуль магнитного поля (а) и азимутальная скорость течения плазмы (б)

Результаты расчетов в двумерной осесимметричной постановке для задач о развитии магниторотационной неустойчивости (МРН) в аккрецирующей околозвездной плазме (постановку задачи см. в [12]) и задаче о разрушении цилиндрической конструкции под действием тепловой нагрузки при нагревании [6] приведены на рис. 8 и 9.

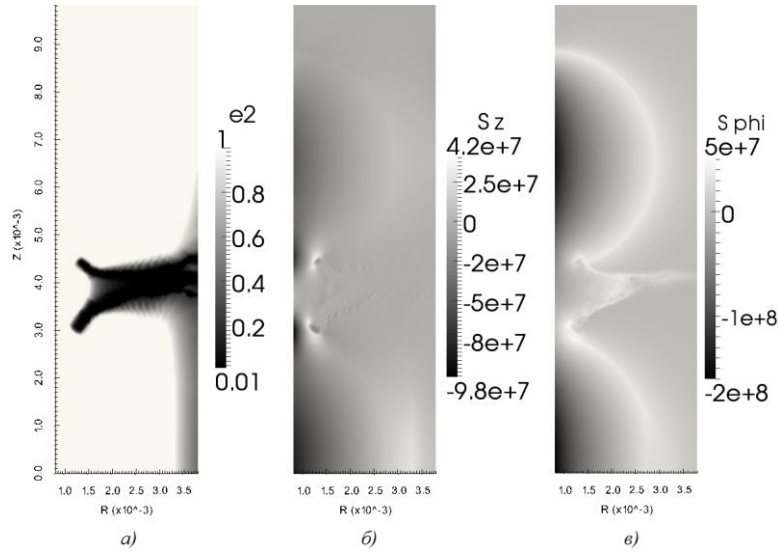

Рис. 9. Моделирование разрушения хрупкой конструкции под действием тепловой нагрузки в рамках модели размазанных трещин. а) – распределение функции памяти e_2, эффективным образом описывающей появление трещины в материале ($e_2 = 1$ – неповрежденный материал, $e_2 = 0$ – полностью разрушенный). б) и в) – компоненты тензора напряжений σ_{zz} и $\sigma_{\varphi\varphi}$ соответственно.

6. Заключение

Представлен прототип интегрированной программной платформы для сопровождения вычислительного эксперимента в комплексных задачах математического моделирования. Предложена архитектура платформы, состоящая из трех подсистем различного назначения – математической, графическо-инструментальной и подсистемы семи предметных библиотек. Оформление в виде пополняемых библиотек элементов исследуемой модели – расчетных областей, физических свойств сред, сценариев моделирования – открывает возможности безболезненного расширения платформы. При этом каждая такая библиотека требует создания форматов данных и интерфейсов подключения внешних, не зависящих от платформы Теметос, потоков данных

и процедур. Дальнейшее развитие платформы будет осуществляться по следующим основным направлениям:

1. развитие и унификация форматов данных, прежде всего двумерных и трехмерных смешанных неструктурированных сеток;
2. развитие редактора задания сценария моделирования, адаптация его к различным предметным областям;
3. создание инструментария управления расчетом на удаленных машинах, в том числе на суперкомпьютерах кластерного типа;
4. реализация запуска параллельных версий модулей-решателей с получением подробной телеметрии параллельного расчета;
5. адаптация и включение в состав платформы новых модулей-решателей, превращение платформы в мультифизичный программный комплекс математического моделирования.

Список литературы

[1]. Галанин М.П., Савенков Е.Б. Методы численного анализа математических моделей. М.: Изд-во МГТУ им. Н. Э. Баумана, 2010. 591 с.

[2]. ANSYS // ANSYS, Inc. Режим доступа: http://www.ansys.com/ (дата обращения: 29.05.2013).

[3]. OpenFOAM // The OpenFOAM Foundation. Режим доступа: http://www.openfoam.org (дата обращения: 29.05.2013).

[4]. Галанин М.П., Лукин В.В., Шаповалов К.Л. Параллельный алгоритм RKDG метода второго порядка для решения двумерных уравнений идеальной магнитной гидродинамики // Параллельные вычислительные технологии (ПаВТ'2013): труды международной научной конференции (1-5 апреля 2013 г., г. Челябинск). Челябинск: Издательский центр ЮУрГУ. 2013. С. 116-126.

[5]. Богатырь С.М., Галанин М.П., Крупкин А.В., Кузнецов В.И., Лукин В.В., Новиков В.В., Родин А.С., Станкевич И.В., Яковлев М.Е. Математическое моделирование термоупругого контактного взаимодействия осесимметричных тел. Инженерный журнал: наука и инновации, 2013, вып. 4. URL: http://engjournal.ru/catalog/mathmodel/hidden/667.html.

[6]. Семерикова М.А. Математическое моделирование хрупкого материала под действием тепловых нагрузок // Молодежный научно-технический вестник. МГТУ им. Н.Э. Баумана. Электрон. журн. 2013. № 3. Режим доступа: http://sntbul.bmstu.ru/file/505616.html (дата обращения: 29.05.2013).

[7]. Горбунов-Посадов М.М. Как растет программа // ИПМ им. М.В. Келдыша РАН. Режим доступа: http://www.keldysh.ru/gorbunov/grow.htm (дата обращения: 29.05.2013).

[8]. Qt // Digia. Режим доступа: http://qt.digia.com/ (дата обращения: 29.05.2013).

[9]. The Visualization Toolkit // Kitware. Режим доступа: http://www.vtk.org/ (дата обращения: 29.05.2013).

[10]. Щеглов И.А. Программа для триангуляции сложных двумерных областей Gridder2d. // Препринт ИПМ им. М.В.Келдыша РАН №60. 2008. 32 с.

[11]. Лукин В.В., Марчевский И.К., Морева В.С., Попов А.Ю., Шаповалов К.Л., Щеглов Г.А. Учебно-экспериментальный вычислительный кластер. Ч.2. Примеры решения задач // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. 2012. № 4. С. 82-102.

[12]. Велихов Е. П., Сычугов К. Р., Чечеткин В. М., Луговский А. Ю., Колдоба А.В. Магниторотационная неустойчивость в аккрецирующей оболочке протозвезды и образование крупномасштабной структуры магнитного поля // Астрономический журнал. 2012. Т. 89, №. 2. С. 107-119.

Prototype of an integrated software platform for tracking computer simulations to solve complex problems of mathematical modeling

M.P. Galanin <Galan@keldysh.ru>
M.M. Gorbunov-Posadov <Gorbunov@keldysh.ru>
A.V. Ermakov <Ermakov@keldysh.ru>
V.V. Lukin <VVLukin@gmail.com>
A.S. Rodin <rals@bk.ru>
K.L. Shapovalov <ShapovalovKL@yandex.ru>
Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, Moscow, 125047, Russia

Abstract. The software platform for mathematical modeling support Temetos is decribed. The Temetos platform is the integrated graphical tool environment for the study of mathematical models of physical processes and technical systems. Temetos platform provides opportunities for preparation of geometric and physical models of the object; external or built-in problem-oriented modules configuration, including numerical method parameters configuration; computational module launch and executing control; calculations results visualization and analysis. The platform consists of three subsystems: the mathematical subsystem, including command-line utilities for the computational domain discretization and triangulation and software modules for the numerical solution of the mathematical model differntial equation systems; instrumental graphics subsystem, including tools for the 3D geometric design of the investigated object model; subject libraries subsystem containing structured knowledgebase for the simulated structures, constructions and physical processes (elements of the libraries are: 3D object models, meshes, material physical models and others.). The subject libraries subsystem is the base of platform information structure. It contains of free format XML files decribing knowledgebase elements. Hierarchical library architecture and universal data interfaces leads the platform to grow and to implement the new subject areas and mathematical models in easy way. As examples of platform implementation the simulation of plasma flow instability in astrophysical conditions and model of brittle material failure under heat load are considered.

Keywords: mathematical modeling, integrated software platform, computational experiment, model knowledgebase

References

[1]. Galanin M.P., Savenkov E.B. Metody chislennogo analiza matematicheskikh modelej [Methods for mathematical models numerical analisys]. MGTU im. N. E. Baumana Publ. 2010. 591 pp. (In Russian)

[2]. ANSYS - ANSYS, Inc.: http://www.ansys.com/

[3]. OpenFOAM // The OpenFOAM Foundation: http://www.openfoam.org

[4]. Galanin M.P., Lukin V.V., SHapovalov K.L. Parallel'nyj algoritm RKDG metoda vtorogo poryadka dlya resheniya dvumernykh uravnenij ideal'noj magnitnoj gidrodinamiki [Parallel algorithm of 2nd order RKDG method for 2D ideal MHD equations solution] Parallel'nye vychislitel'nye tekhnologii (PaVT'2013): trudy mezhdunarodnoj nauchnoj konferentsii [Parallel computing technologies: proceedings of international conference] Chelyabinsk, YurGU. 2013. Pp. 116-126.

[5]. Bogatyr' S.M., Galanin M.P., Krupkin A.V., Kuznetsov V.I., Lukin V.V., Novikov V.V., Rodin A.S., Stankevich I.V., Yakovlev M.E. Matematicheskoe modelirovanie termouprugogo kontaktnogo vzaimodejstviya osesimmetrichnykh tel [Mathematical modeling of axyssimetric bodies thermoelastic contact interaction]. Inzhenernyj zhurnal: nauka i innovatsii [Engineer journal: science and innovations]. 2013, 4.

[6]. Semerikova M.A. Matematicheskoe modelirovanie khrupkogo materiala pod dejstviem teplovykh nagruzok [Mathematical modeling of brittle material behavior under the influence of heat loads] Molodezhnyj nauchno-tekhnicheskij vestnik. MGTU im. N.E. Baumana. [Young scientists scientific and technick journal] 2013. № 3.

[7]. Gorbunov-Posadov M.M. Kak rastet programma [How the program grows] KIAM RAS: http://www.keldysh.ru/gorbunov/grow.htm

[8]. Qt Digia. http://qt.digia.com

[9]. The Visualization Toolkit Kitware. http://www.vtk.org

[10]. Shcheglov I.A. Programma dlya triangulyatsii slozhnykh dvumernykh oblastej Gridder2d [The software for complex 2D domains triangulation Gridder2D] KIAM RAS Preprint 60. 2008. 32 pp.

[11]. Lukin V.V., Marchevskij I.K., Moreva V.S., Popov A.YU., Shapovalov K.L., Shcheglov G.A. Uchebno-ehksperimental'nyj vychislitel'nyj klaster. Ch.2. Primery resheniya zadach [Computational cluster for studies and experiments. P. 2. Examples of problems solution] Vestnik MGTU im. N.E. Baumana. Estestvennye nauki. [BMSTU journal: Natural sciences] 2012, 4, pp. 82-102.

[12]. E.P. Velikhov, K.R. Sychugov, V.M. Chechetkin, A.Yu. Lugovskii, A.V. Koldoba. Magneto-rotational instability in the accreting envelope of a protostar and the formation of the large-scale magnetic field. Astronomy Reports, 2012. 56, 2. Pp. 84-95. doi: 10.1134/S106377291201009X

Обзор масштабируемых систем межмодульных оптимизаций

Долгорукова К.Ю.
<unerkannt@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. Большинство приложений имеют модульную структуру, но оптимизация таких программ при сборке по раздельной схеме “компиляция-связывание” ограничивается отдельным модулем. И, хотя многие компиляторы поддерживает межмодульные оптимизации, в случае больших приложений их полноценное проведение зачастую неприемлемо ввиду значительных затрат времени и памяти. Решающие эту проблему компиляторные системы, способные производить межмодульные оптимизации с учетом возможностей аппаратуры и требований пользователя по затратам ресурсов, называются масштабируемыми системами межмодульных оптимизаций, или оптимизаций времени связывания.

В данной статье мы ставим задачу рассмотреть прежде всего различные подходы к проблеме масштабируемости инфраструктуры относительно потребляемых ресурсов, а не сами межпроцедурные и межмодульные оптимизации или их эффективность в той или иной реализации, так как они по большому счёту не зависят от используемой инфраструктуры. Интерес представляют возможности распараллеливания тех или иных стадий компиляции целой многомодульной программы, а также способы экономии памяти при межмодульном анализе.

Данный обзор включает в себя несколько компиляторных систем для языков общего назначения: C, C++, Fortran, – но нередко системы способны оптимизировать и другие языки, если для них реализован соответствующий генератор промежуточного представления.

В заключении статьи будут представлены предварительные результаты по масштабированию компонентов связывания инфраструктуры LLVM.

Ключевые слова: компиляторы; межпроцедурные оптимизации; межмодульные анализ и оптимизации; системы межмодульных оптимизаций.

1. Введение

Б*о*льшая часть инфраструктур межпроцедурных оптимизаций имеет в своей основе общую модель компиляции, в которой представлены 3 основные фазы: генерация промежуточного кода (ГПК), фаза межпроцедурных оптимизаций и генерация машинного кода (ГМК).

Во время ГПК компилятор получает файлы с кодом на исходном языке программирования, проводит небольшое количество локальных оптимизаций в каждом файле отдельно, а также генерирует дополнительную информацию о зависимостях между функциями в так называемые summary, или аннотации, представляющие собой некую информацию о программе, необходимую для проведения межпроцедурного анализа. В результате работы первого этапа генерируется расширенный файл в промежуточном представлении (либо в двоичном формате), содержащий также дополнительную информацию. Обычно этап ГПК легко параллелизуем.

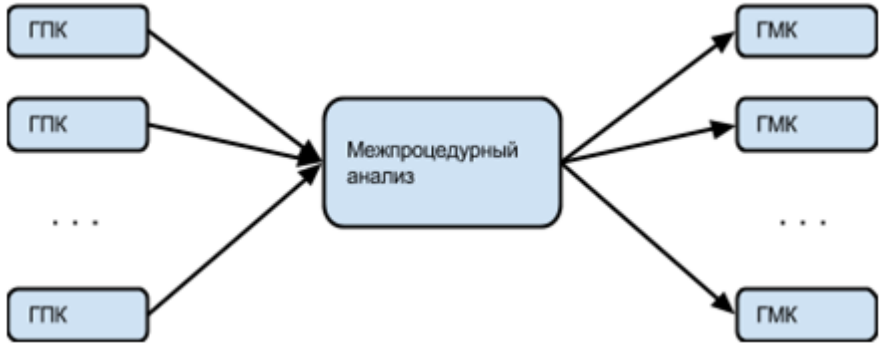

Рис. 1. Схема работы систем оптимизаций времени связывания

Второй этап как правило проходит во время связывания и читает файлы, полученные на первом этапе, анализирует межпроцедурные зависимости, проводит некоторые оптимизации, и снова генерирует файлы в промежуточном представлении и некоторое количество дополнительной информации. Этот процесс трудно параллелизуем, и все компиляторные системы с открытым исходным кодом выполняют межмодульный анализ в один поток, поэтому именно анализ – узкое место этих систем.

Последняя фаза принимает файлы, сгенерированные на втором этапе и генерирует из них объектный код, который затем подается ассемблеру. Иногда на этой стадии проводятся дополнительные оптимизации. Эта фаза у разных компиляторов может быть параллелизуемой или непараллелизуемой, и в результате может получаться либо один объектный файл, либо несколько, которые впоследствии передаются стандартному системному компоновщику.

Несмотря на то, что развитие многих представленных в статье систем проходило примерно в одно время – и разработка большинства из них ведется до сих пор, – мы постараемся рассмотреть эти системы в хронологической последовательности относительно их появления, делая исключения лишь для наследуемых один от другого систем для простоты сравнения. Так, несмотря на относительную “свежесть” SYZYGY, мы рассмотрим его вторым из-за сильной родственной связи с HLO. Затем мы рассмотрим Open64 и

межпроцедурный компилятор из GCC. В следующей главе речь пойдет о LLVM, как об альтернативном взгляде на межмодульные оптимизации. Последним рассматривается легковесный компилятор от Google, который учёл все успехи и ошибки предшественников и стоит особняком от остальных представителей систем межпроцедурных оптимизаций из-за радикальности подхода.

Для того, чтобы не повторяться, говоря о похожих сущностях, в каждой посвященной какому-то определенному фреймворку главе мы будем говорить только о нововведениях, а при необходимости будем ссылаться на реализованные ранее подходы.

2. HLO

High-Level Optimizer (оптимизатор высокого уровня) [1] – одна из первых промышленных платформ с возможностью масштабирования по памяти. Ограничение памяти достигается, прежде всего, сохранением неиспользуемых в текущий момент данных оптимизатора на диск.

Авторы системы протестировали её на скомпонованных без оптимизаций, с межпроцедурными оптимизациями, оптимизациями с профилем и межпроцедурными оптимизациями с профилем крупных тестах SPECint95 и ISV Apps, и пришли к выводу, что самый большой положительный скачок (до 71%) производительности программам даёт именно совместное использование межпроцедурных оптимизаций и профиля. Поэтому авторы системы проектировали HLO с ориентацией именно на такие оптимизации[2].

HLO проектировался как компонент компилятора HP-UX, схема работы которого представляет собой конвейер и показана на рисунке ниже.

Внутри HLO используется система проходов, в которых есть проходы как анализа, так и преобразований, при этом есть и внутрипроцедурные, и межпроцедурные проходы. Но все они, как легко понять из названия оптимизатора, машинно-независимые. Внутри LLO (Low-Level Optimizer – низкоуровневый оптимизатор) также по принципу проходов производятся машинно-зависимые оптимизации.

Инструментирование производится на межпроцедупные ветви и вызовы функций, при этом в компиляторе есть возможность выбрать, на какой стадии его внедрять. После запуска инструментированной программы генерируется база данных, при этом в задачу компилятора входит поддержка соответствия собранного профиля коду исполняемой программы.

Для возможности межмодульной оптимизации ГПК сохраняет промежуточное представление в объектные файлы, соответствующие компилируемым модулям. Когда компоновщик считывает все объекты промежуточного представления, он передает их оптимизатору и генератору кода.

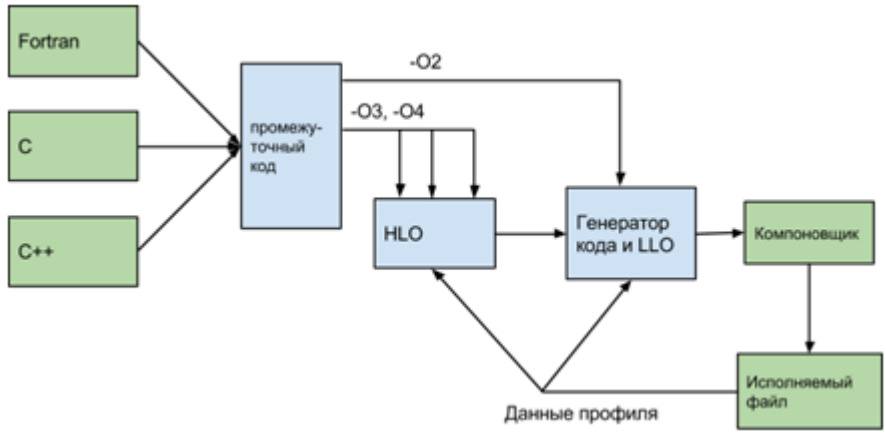

Рис. 2. Схема работы компилятора HP-UX

2.1 Модель “не всё в памяти”

Так как потребление памяти при межмодульных оптимизациях больших программ может быть невероятно огромным (до 1.7 кб на каждую строку исходного кода!), разработчики из HP предложили модель NAIM (Not-All-In-Memory – “не всё в памяти”), позволяющую частично сбрасывать данные на диск для ресурсоёмких оптимизаций.

Снижение потребления достигается несколькими методиками: сжатием структур, классификацией данных на используемые и неиспользуемые в данный момент, и выгрузкой объектов из памяти вместе со сбросом на диск.

Для сжатия объекты классифицируются как *глобальные, временные* и *производные*. *Глобальные* объекты всегда находятся в памяти и представляют собой такие структуры, как граф вызовов, таблица символов всей программы, и таблица модулей компиляции, в которой хранятся ссылки на объекты временных структур. Временные структуры – это таблицы символов для каждого модуля и процедуры в промежуточном представлении.

Все *временные* объекты могут находиться в трёх состояниях: в развернутом виде в памяти, в сжатом виде в памяти, либо в сжатом виде сброшенные на диск.

Третья категория содержит объекты, которые всегда можно сгенерировать на основе временных и глобальных объектов. К этим объектам относятся рёбра потоков данных, деревья интервалов, аннотации переменных индукции. *Производные* объекты не нужно сбрасывать на диск, достаточно их удалить, а при необходимости сгенерировать заново. Организация объектов показана на рис.2.

Сам механизм выгрузки объектов работает так: данные, необходимые для выполнения текущего прохода, загружаются в память, а после его выполнения ненужные уже структуры упаковываются и сбрасываются на диск, либо помечаются для удаления. Во время упаковки объекты приводятся к независящей от указателей форме, для этого каждому объекту назначается идентификатор (PID – persistent identifier). Во время распаковки каждый PID преобразовывается в адрес.

Для более эффективного управления памятью в HLO используется динамическое размещение в памяти и сборщик мусора. При этом, менеджер памяти старается учитывать локальность кода и складывать зависимые объекты близко друг к другу.

Сама же модель “не всё в памяти” используется не всегда. В частности, если имеет место оптимизация маленьких приложений, то программа загрузится в память целиком. Для включения разных механизмов NAIM, потребление памяти должно перейти несколько границ, зависящих от свойств аппаратного обеспечения и параметров пользователя. По достижении первой границы, система сначала включает механизм упаковки тел функций, затем, при переходе через вторую отметку, начинают упаковываться таблицы символов. Если же количество потребляемой памяти перешагнет и через третий порог, включится механизм выгрузки функций. Выгрузка работает “ленивым” образом, то есть объекты начинают выгружаться, когда система займёт всю отведённую ей память.

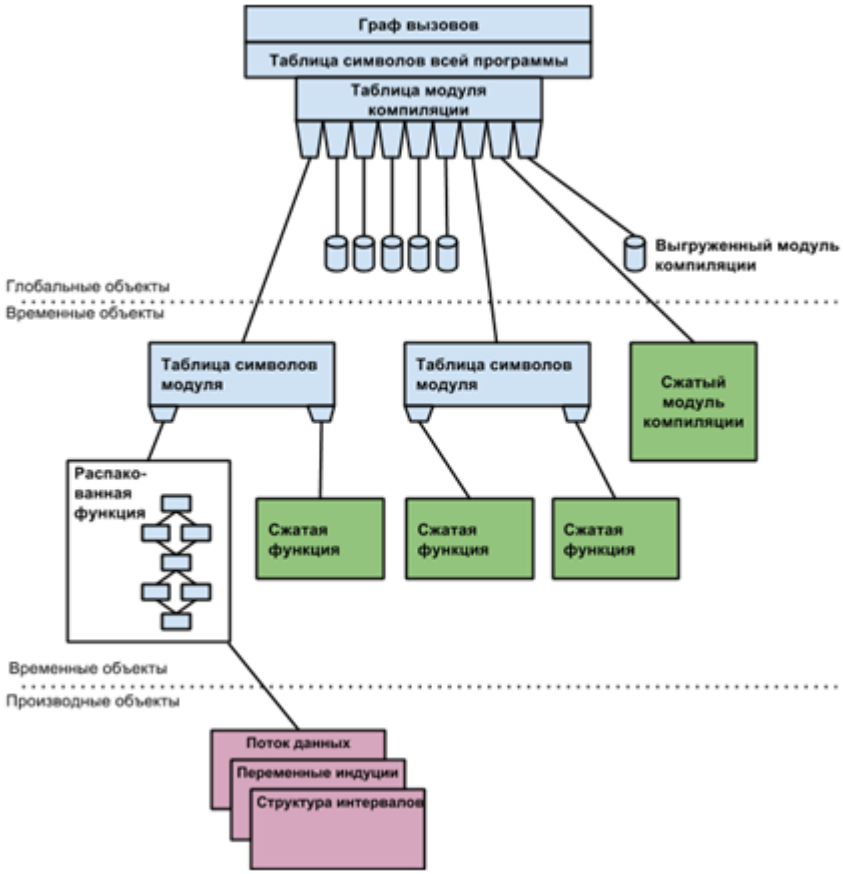

Рис. 3. Схема "не всё в памяти" компилятора HLO

2.2 Выбор участков кода для оптимизации и профилирование

Было замечено, что для получения максимальной производительности на деле достаточно оптимизировать не более 20% кода[12]. Чтобы не оптимизировать участки кода, которые заведомо в этом не нуждаются, в системе реализованы механизмы крупнозернистой и мелкозернистой выборки. Крупнозернистая выборка просматривает данные профиля и выбирает наиболее горячие модули для оптимизации. Для этого компилятор сортирует вызовы по частоте согласно профилю в убывающем порядке. Далее отбираются вызовы, набравшие заданный пользователем процент, производится поиск модулей, содержащих вызывающие и вызываемые функции – и к ним применяются межмодульные оптимизации с

профилем. К остальным применяются обычные внутримодульные оптимизации с профилем.

При мелкозернистой выборке межмодульные оптимизации производятся над горячими функциями, независимо от содержащих их модулей.

Сбор профиля производится посредством встраивания, при этом даже при изменении исходного кода есть возможность использовать старый профиль повторно.

Система межмодульной оптимизации HLO была внедрена в компиляторную систему HP-UX начиная с версии 9.0, модель “не всё в памяти” была добавлена в неё начиная с версии 10.20. Прирост производительности сгенерированного системой кода составлял от 20 до 70% относительно обычной компиляции без межмодульных оптимизаций.

3. SYZYGY

Компилятор SYZYGY[3] – прямой последователь HLO, он также был разработан как компонента для системы HP-UX. Но в данном компиляторе уже используется модель оптимизации из двух стадий: решение об оптимизациях и сами оптимизации. В целях ограничения потребляемой памяти фреймворк использует алгоритмы, которые поддерживают открытыми не более некоторого числа файлов с промежуточным представлением одновременно, и минимизируют количество открытий и закрытий файлов. После оптимизаций промежуточное представление записывается в файлы обратно, и на них прогоняется параллелизуемый ГМК.

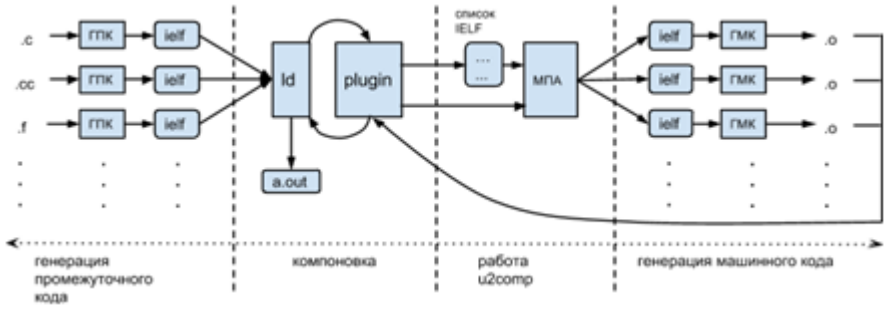

Рис. 4. Схема работы SYZYGY

Сбор данных о процедурах и модулях для аннотаций происходит во время работы генерации промежуточного кода. Перед сбором данных ГПК производит над кодом такие оптимизирующие преобразования, как свёртка констант, канониканизацию промежуточного представления, упрощение алгебраических выражений. Это необходимо для уменьшения размеров и увеличения точности аннотаций.

Стадия межпроцедурного анализа (МПА) строит таблицу глобальных символов, производит разрешение имён и унификацию типов, а также строит граф вызовов.

Оптимизации, которые требуют контекст из других модулей, – например, встраивание функций, клонирование процедур и распространение неявных вызовов, – разделены в SYZYGY на две стадии. Первая стадия производится во время анализа, вторая – во время генерации машинного кода (ГМК). Так как ГМК работает только с одним файлом за раз, необходимая информация о функциях в других файлах выносится в промежуточное представление во время стадии анализа. Во время стадии ГМК производятся только внутримодульные оптимизации и генерация кода.

3.1 IELF

Промежуточное представление хранится в формате, представляющем собой обычный 32 или 64-битный файл формата ELF[4], включающий заголовок, таблицу символов, таблицу строк, а также несколько секций внутреннего представления скомпилированной программы в двоичном виде.

В памяти промежуточное представление сохраняется в областях – блоках памяти, управляемых компилятором. Ссылки в областях представляются идентификаторами, а не указателями, для этого у каждой области есть массив идентификаторов. В файле IELF этот массив хранится в отдельной секции, также есть секция отображений для быстрого поиска этого массива. Считывание массива идентификатора происходит во время чтения или записи файлов, когда это необходимо. Во время работы анализатора в памяти поддерживается всего несколько файлов, остальные сбрасываются на диск. Обычно размер файла в формате IELF в среднем в пять раз больше, чем обычный объектный файл, но в худшем случае этот показатель может достигать 700. Это связано с тем, что в файле должна содержаться информация о всех используемых типах. Если, например, объектный файл состоит из нескольких строк, в одной из которых используется ссылка на поле объекта очень большого класса из другого файла, то вся необходимая информация о типах будет указана и в файле, ссылающемся на объект.

3.2 Схема работы

Модель компиляции SYZYGY тоже состоит из стадий: генерация промежуточного кода, анализ и генерация машинного кода, – и содержит соответствующие компиляторные компоненты. ГПК и ГМК выполняют обработку одного файла за раз и могут быть запущены в параллельных потоках. Анализатор работает со всеми файлами в один поток.

ГПК получает на вход файлы исходного кода и генерирует из них IELF файлы после применения немногочисленных простых оптимизаций. Во время связывания компоновщик, представленный утилитой ld, разрешает

зависимости, после чего передает эстафету плагину, который собирает список файлов формата IELF, распаковывает некоторые из этих файлов и объединяет с информацией, полученной от компоновщика. Затем плагин вызывает программу u2comp, передавая ей список IELF-файлов и собранную компоновщиком и плагином информацию. u2comp – это утилита, производящая межпроцедурный анализ и некоторые преобразования в промежуточном представлении. После окончания всех вышеперечисленных действий она генерирует новый набор IELF-файлов, а затем файл инструкций сборки “makefile”, в которых в многопоточном режиме запускается ГМК.

3.3 Межпроцедурные оптимизации

Как было отмечено ранее, такие оптимизации, как встраивание, разделены, так как требуют дополнительных подготовительных действий. Встраивание функций может сильно изменить структуру программы, поэтому, чтобы избежать изменений в аннотациях, оно проводится в самом конце фазы анализа.

Программа u2comp начинает работу с построения глобальных структур данных, таких как таблица глобальных символов. Затем происходит удаление лишних секций COMDAT, содержащих мёртвый код и дублирующие участки кода – и изменения фиксируются в файлах. Удаления COMDAT-секций необходимо для уменьшения размера IELF-файлов, а значит, и для облегчения последующего чтения файлов.

Затем происходит несколько межпроцедурных оптимизаций. Необходимые для них результаты анализа либо содержатся в памяти, либо сбрасываются в файлы, в зависимости от размера. Чтобы не переписывать после каждой оптимизации анализ, изменения, произведенные оптимизирующими преобразованиями, записываются в список изменений, а перед самой процедурой встраивания они применяются одна за другой к промежуточному представлению программы. Для облегчения обновления аннотаций также поддерживается список “заплаток”, который потом разом применяется к промежуточному представлению во время очередного чтения соответствующего участка кода.

Во время самого ресурсоёмкого, но весьма эффективного преобразования – встраивания, – программа не держит всё промежуточное представление в памяти. Также в постоянной памяти находится лишь незначительный набор аннотаций, или ядро аннотаций. Количество одновременно открытых файлов определяется потребляемой ими памятью. Если программа достигла лимита предоставленной ей памяти, она начинает закрывать неиспользуемые файлы перед открытием новых. В SYZYGY всего одна фаза анализа и одна – преобразований, а значит, встраивание проводится всего один раз. Поэтому аннотации обновляются после каждого решения о встраивании.

В сравнении со своим предшественником, HLO, SYZYGY работает быстрее в среднем в 2 раза, показывая похожие результаты производительности. Для однопоточного запуска издержки составляют 2.3x, а для запуска в 4 потока – 1.2x на SPEC2000.

4. Open64

Open64 – небольшая исследовательская компиляторная система, изначально созданная для работы на процессорах семейства Intel, что, впрочем, не мешает ей работать и на многих современных архитектурах[5]. Мы поверхностно упомянем применяемые в нем методы, а затем глубже разберем его прямого и более развитого и документированного потомка – WHOPR из GCC.

Open64 имеет стандартную структуру, то есть содержит параллелизуемые ГПК, ГМК и непараллелизуемый межпроцедурный этап. Во время работы ГПК создаются аннотации, и записывается вместе с промежуточным представлением WHIRL в файл. Межпроцедурный этап содержит фазы анализа и оптимизирующих преобразований. На фазе анализа компилятор работает только с аннотациями и не открывает промежуточное представление, поэтому она проходит быстро относительно предшественников. На фазе оптимизации компилятор уже работает с промежуточным представлением файлов, загружая их все в память сразу. Затем он записывает оптимизированную программу в объектный файл формата ELF, причём, количество выходных файлов может не соответствовать количеству входных: компилятор предварительно разбивает программу в зависимости от размера и внутренних зависимостей процедур. Open64 производит также один промежуточный объектный файл, содержащий все статически “продвинутые” и глобальные переменные, а также межпроцедурную таблицу символов и типов данных. Во время ГМК этот файл обрабатывается первым, перед параллельной компиляцией остальных промежуточных файлов. После этого на полученных объектные файлах запускается ГМК[6].

5. GCC

В GCC режим масштабируемых межмодульноых оптимизаций называется WHOPR – WHole Program Optimization[7]. WHOPR отчасти похож на Open64, но в нём также реализованы такие новые особенности, как виртуальные клоны и функции перехода, а также компилятор способен классифицировать функции по частоте использования согласно профилю.

Дополнительные секции, добавляемые в ELF во время ГПК (фаза *LGEN* – Local Generation, локальная генерация) – это опции командной строки (.gnu.lto_.opts), таблица символов (.gnu.lto_.symtab), глобальные типы и объявления (gnu.lto_.decls), граф вызовов (.gnu.lto_.refs), тела функций в промежуточном представлении, инициализаторы статических переменных (.gnu.lto_.vars) и, собственно, аннотации.

Во время второй фазы (*WPA* – Whole Program Analysis) работа ведется без доступа к телам функций и инициализаторам переменных, то есть на основе аннотаций. И обновляется в файлах только участки с аннотациями, куда записываются решения об изменении функций.

ГМК (в GCC он называется *LTRANS* – Local Transformation, локальное преобразование), – в отличие от Open64, стремящемся целиком записать граф вызовов в один файл, когда это позволяет память, – работает над файлами, тела которых остаются в там же, за исключением, быть может, мёртвых и полностью встроенных функций в промежуточном представлении. Фаза ГМК производит локальные для файлов оптимизации на основе информации, находящейся в секциях с аннотациями, в параллелизуемом режиме.

5.1 Виртуальные клоны и функции перехода

Отличительной особенностью компилятора GCC является наличие инфраструктуры так называемых виртуальных клонов, необходимой для поддержания целостности функций во время преобразований. Виртуальные клоны в графе вызовов – это функции без тел, имеющие только описание, как создать эту функцию на основе другой функции. Чтобы не “ломать” структуры программы, что может не только ухудшить эффективность кода, но и изменить семантику, а также привести к висячим ссылкам, все преобразования на фазе анализа производятся над этими клонами. Описание клонов содержит изменения в сигнатуре, в теле функции, и указатель на вызывающую функцию для функций, которые должны быть встроены. По сути, на фазе WPA над графом вызовов производятся полноценные оптимизирующие преобразования, но изменения представляются графом вызовов и клонами, а сами функции остаются нетронутыми. Потом, на фазе ГМК, из клонов строятся полноценные функции, и граф вызовов перестраивается согласно аннотациям.

Также в GCC были введены так называемые *функции перехода*. Каждая функция перехода описывает значение параметра определенного вызова функции. Эта сущность была введена для таких оптимизирующих преобразований, как межпроцедурное распространение констант, встраивание и девиртуализация.

Оптимизации также производятся на всех трех этапах, но на первом и последнем – над промежуточным представлением, на втором – над аннотациями, графом вызовов, а также над клонами. На первом этапе проводятся такие оптимизации, как распространение констант, удаление мертвого кода, простые межпроцедурные оптимизации в пределах модуля, создание статического профиля, разбор дополнительных атрибутов функций (таких, как nonreturn, nonthrow и пр.), разбиение функций и некоторые другие.

5.2 Классификация функций и распространение профиля

В фазе анализа важнейшей частью является процедура распространения профиля. Функции разбиваются на 4 класса по частоте исполнения: *горячие*, *нормальные*, *исполняемые единожды* и *маловероятно исполняемые*. Эта информация нужна, чтобы определить, оптимизировать размер или быстродействие функций. Так, горячие и нормальные функции оптимизируются для быстродействия, тогда как исполняемые единожды функции оптимизируются только внутри циклов, а маловероятно исполняемые – по размеру. Если профиль исполнения недоступен, этот компиляторный проход распространяет статическую информацию следующим образом:

- Когда все вызовы функции маловероятно исполняемые – то есть вызывающая функция вероятно неисполняемая или функция холодная согласно профилю, – функция помечается как маловероятно исполняемая.
- Когда все вызывающие функции маловероятно исполняемые или исполняемые единожды, и вызывают рассматриваемую функцию по одному разу каждая, то функция помечается как исполняемая единожды.
- Когда все вызывающие функции вызываются только в момент запуска программы, то сама функция тоже помечается как вызываемая только в момент запуска.

Горячие функции помещаются в специальную подсекцию текстового сегмента файла .text.hot. Маловероятные функции помещаются в подсекцию .text.unlikely, а исполняемые при запуске – .text.startup.

5.3 Основные проблемы WHOPR-компиляции больших приложений

Несмотря на то, что в GCC применяется множество механизмов для уменьшения потребления памяти (использование только аннотаций и графа вызовов вместо промежуточного представления), потребление памяти компилятором всё еще высоко. Так, для компиляции теста cc1 требуется в 50 раз больше памяти, чем размер файла. Для компиляции же браузера Firefox нужно в 130 раз больше памяти, чем занимают все вместе взятые его исходные файлы. GCC делает отображение на память во время чтения, а также многократно открывает и закрывает файлы для считывания (в отличие от SYZYGY, где чтение и запись строго контролируются и их количество ограничено). Также представление высокоуровневых типов в памяти занимает очень много места (3.7 Гб для Firefox)[8]. Проблема необходимости перекомпиляции при изменении всего одного файла также пока открыта.

6. Google's LIPO (Lightweight Inter-Procedural Optimization)

Схема работы компиляторных инструментов от Google[10] несколько отличается от предшественников. Так как разработчики сделали упор на оптимизации с использованием профиля – а именно, встраивание функций и продвижения неявных вызовов, – они сделали процедуру сбора профиля неотъемлемой частью системы. Вместо традиционной цепочки ГПК – анализ – ГМК, в LIPO две последние фазы заменены на компоненту динамического анализа времени исполнения программы и оптимизирующий компилятор соответственно.

ГПК обрабатывает по одному файлу и генерирует инструментированный код непосредственно в исполняемом формате, без какого-либо промежуточного представления. Для того, чтобы можно было построить граф вызовов прямо во время исполнения, на вызовы функций добавляется дополнительное инструментирование.

Runtime компонента включает в себя инструмент запуска инструментированной программы, профилировщик, анализатор и генератор аннотаций. Межпроцедурный анализ начинает работу сразу после исполнения программы, происходит динамическое построение графа вызовов, причём для неявных вызовов участки графа строятся по результатам профилирования, а для явных используются результаты, собранные с помощью дополнительного инструментирования, упомянутого ранее. Анализатор изучает граф вызовов и профиль и разделяет для улучшения встраивания модули на кластеры, или супермодули, посредством жадного алгоритма. Например, если функция содержит "горячий" вызов другой функции из другого файла, то в кластер, относящийся к файлу с вызывающей функцией, будет добавлен файл с вызываемой функцией. После кластеризации дублирующие и неиспользуемые функции удаляются из кластеров, после чего на основе структуры кластера строятся аннотации. По окончании анализа суммарные данные профилирования и результаты анализа записываются в файл профиля, а аннотации – прямо в раздел данных программы для непосредственного использования в следующем этапе.

На этапе оптимизаций компилятор читает данные профиля и аннотации, а также данные о кластерах. Если в кластер входит несколько модулей, то открываются дополнительные модули и производится встраивание и продвижение неявных вызовов прямо на исполняемом коде.

6.1 Сборка

При первичной сборке проекта с использованием LIPO всё происходит по вышеописанной схеме ГПК – исполнение и анализ – оптимизация. Если же изменилось несколько файлов с исходным кодом, необходимо компилировать заново все супермодули, в которые входят измененные файлы. Поддержка

повторной компиляции реализована как отдельный инструмент к компилятору.

На тестах SPEC2006 программа показала улучшение производительности на 4.4%, при увеличении размеров файлов 25% и издержками 32% на фазе исполнения инструментированного кода и анализа. При этом большая часть издержек приходится на инструментирование, тогда как анализ по времени в среднем занимает лишь 1%, что удивительно быстро по сравнению с другими средствами межпроцедурных оптимизаций. Но, к сожалению, из-за отсутствия промежуточного представления, проводить другие оптимизации становится проблематично, и это ограничивает потенциал такого подхода к оптимизациям.

7. LLVM

Инфраструктура LLVM (Low-Level Virtual Machine – низкоуровневая виртуальная машина)[9] и все его компиляторные компоненты используют промежуточное представление – биткод – набор RISC-подобных трёхадресных инструкций. Процесс сборки программы посредством LLVM также составляют три стандартные фазы. Типичный жизненный цикл обычной сборки с межпроцедурными оптимизациями выглядит так, как показано на рисунке 5.

Цепь состоит из двух инструментов: компилятора Clang и утилиты ld с подключенным к нему плагином Gold. Здесь межпроцедурные оптимизации происходят благодаря работе Gold-плагина, а Clang производит локальные оптимизации.

Так как система предназначена для оптимизации программ среднего и малого размера, на заключительной фазе загружается вся программа помодульно, производится связывание модулей в промежуточном представлении, после чего над полученным промежуточным кодом проводятся анализ и оптимизационные преобразования по многопроходной схеме. После оптимизации производится генерация машинного кода.

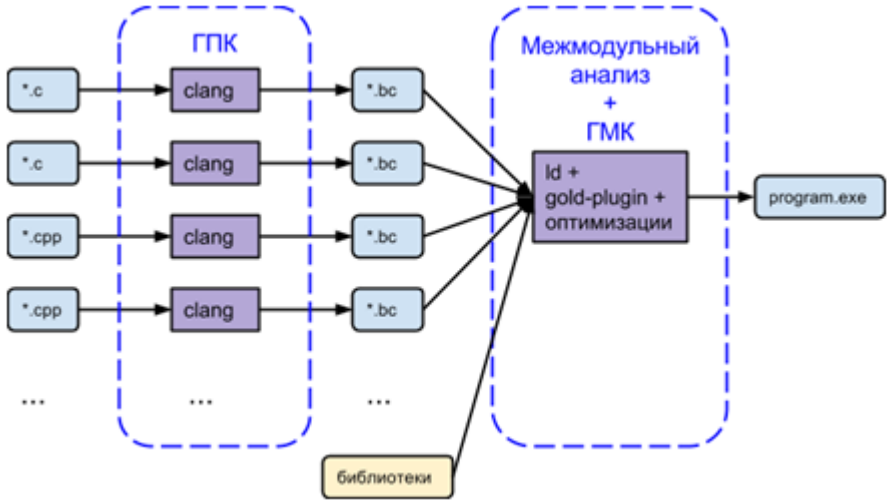

Рис. 5. Схема работы LLVM

В разрезе масштабируемости в системе LLVM присутствует ряд проблем. Промежуточное представление LLVM делится на модули, которые состоят из функций, представленных, в свою очередь, базовыми блоками, которые состоят из инструкций. Анализирующие и оптимизирующие проходы подразделяются соответственно уровню иерархии промежуточного представления на модульные, проходы функций, циклов и базовых блоков. Чаще всего используются первые два, при этом они могут запросить любые данные по иерархии ниже. Стандартная многопроходная схема последовательно работает над скомпонованной программой, при этом за модуль обработки принимается вся программа – а это значит, что проходы, работающие над модулем, могут потребовать любую часть программы во всей иерархии промежуточного представления. Это значит, что в памяти необходимо сохранять программу целиком. По этой же причине последний этап сложно проводить параллельно.

7.1 Работа над масштабируемостью LLVM

Для возможности управления потреблением ресурсов мы приняли решение сделать следующее:

- Разделить и распараллелить последнюю стадию этапа сборки программы. Вероятнее всего, разделив модули по группам с учетом зависимости по вызовам, будет возможно сделать оптимизации времени связывания более эффективными, как это происходит, например, в GCC.

- Реализовать ленивую загрузку функций на стадии оптимизации. В данный момент «лениво» функции загружаются только во время считывания файла. Оптимизации не начнутся, пока не будут загружены и скомпонованы все модули программы.
- Подстроить некоторые оптимизирующие преобразования под использование частичных данных о биткоде – аннотаций. Также необходимо собирать аннотации во время стадии ГПК и считывать для анализа.

Мы выполнили распараллеливание оптимизаций времени связывания для LLVM, разделив файлы жадным алгоритмом на количество групп, равное числу процессоров на машине. Суммарный размер в байтах каждой группы мы брали приблизительно одинаковым. Затем над группами мы провели оптимизации независимо и параллельно, только затем скомпоновали. Работы по распараллеливанию велись на основе разработанного ранее в ИСП РАН двухфазного компилятора[13].

Необходимость загружать тела функций «лениво» - то есть по мере требования оптимизациями, - родилась из того факта, что пиковая нагрузка на память приходится на момент окончания загрузки всех модулей, составляющих компилируемую программу, и далее почти не меняется. Загрузка кода во время оптимизаций также возможна за счет того, что далеко не все оптимизации времени связывания требуют код вплоть до инструкций: некоторым нужен только граф вызовов и описатели глобальных значений (например, для интернализации), некоторым – только формат данных и глобальные значения (например, для слияния констант) и т.д. Таким образом, реализовав ленивую загрузку тел функций, можно распределить нагрузку на память почти на всё время работы конвейера преобразований.

Необходимость использования аннотаций продиктована несколькими причинами: увеличением эффективности ленивой загрузки и возможностью в дальнейшем вовсе отделить фазу анализа и оптимизации от связывания и генерации машинного кода. В первом случае увеличение эффективности отложенной загрузки достигается за счет максимального оттягивания необходимости загрузки тела функций: посчитав на этапе генерации промежуточного представления такие данные, как граф вызовов, количество инструкций и константные значения, можно провести до трети от всего количества преобразований без использования тел функций.

Мы реализовали аннотации как новый тип блока в биткоде – FUNCTION_DESCRIPTOR_BLOCK_ID. Такой блок присутствует перед каждым телом функции и содержит записи, которых может быть несколько видов: запись размеров, запись с количеством значений и запись с описателями вызовов функций из текущей функции. Эти аннотации с незначительными дополнениями в коде оптимизирующих проходов позволяют продвинуться на 4 прохода в стандартном оптимизирующем наборе оптимизаций времени связывания без загрузки тел функций.

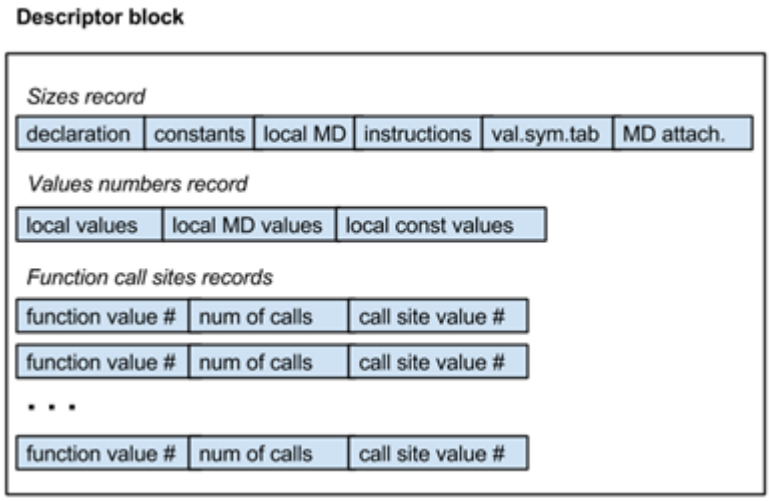

Рис. 6. Расположение данных в описателе функций

8. Заключение и результаты работы

Большинство коммерческих компиляторов не отличаются от описанной во введении модели и очень часто имеют в своей основе один из описанных выше систем. Например, межпроцедурный компилятор от Intel очень похож на WHOPR от GCC[11]. В заключение представим сводную таблицу по возможностям масштабирования всех описанных в статье систем.

Заметим, что три этапа ГПК - анализ - ГМК присутствуют во всех системах в том или ином виде. От стандартного подхода отличается только компилятор от Google, где фаза анализа заменена на анализатор времени исполнения, тем не менее, её конечным назначением тоже является межпроцедурный анализ. Системы развивались от сложных механизмов отгрузки на диск в сторону сокращения объемов необходимой межпроцедурной информации, облегчения промежуточного представления вплоть до полного отказа от него. Между тем, отказ от использования промежуточного представления сильно урезает возможности оптимизаций, усложняет анализ и саму инфраструктуру, хотя и сокращает время компиляции и объем промежуточных файлов. Если говорить о подходе LLVM – то есть об использовании легковесного промежуточного представления, – то текущая политика компилировать единый файл для оптимизаций лишает возможности проводить ГМК в параллельном режиме, хоть и вдобавок избавляет от необходимости в аннотациях. Заметим, что в LLVM на данный момент отсутствует механизм регулирования потребляемой памяти, что затрудняет компиляцию больших приложений.

	Параллелизуемые участки	**Масштабирование по памяти, способ**	**Промежуточное представление / формат файла**	**Возможность использовать повторно аннотации/ профиль**	**Увеличение размера промежуточного файла**	**Увеличение времени на оптимизацию**
HP's HLO	ГПК, ГМК	Модель «не всё в памяти»: сжатие и отгрузка на диск	IL для HP-UX / IELF	Нет / да	>x 10	x2.5-6
SYZYGY	ГПК, ГМК	Ограничение количества открытых файлов, ограничения промежуточного представления, находящегося в памяти	IL для HP-UX / IELF	? / ?	>x 5	x1.2-2.3
Open64	ГПК, ГМК	Работа только с аннотациями и графом вызовов	WHIRL / fat ELF	? / ?	x 1.7	?
GCC's WHOPR	ГПК, ГМК	Работа только с аннотациями и графом вызовов	GIMPLE / fat ELF	Нет / нет	x 1.7	x0.8
LLVM	ГПК	Облегчённое промежуточное представление, отсутствие аннотаций	LLVM / любой объектный формат	- / нет	x 1.25	-
Google's LIPO	ГПК, ГМК	Отсутствие промежуточного кода, анализ производится во время исполнения, только динамический анализ	Нет / ELF	Да / да	x 1.25	x1.32

Табл. 1. Сравнение систем межмодульных оптимизаций

Однако до сих пор узким местом остался межпроцедурный межмодульный анализ, требующий всего графа вызовов. Подход разработчиков Google строить граф динамически и пользоваться только “фактическим графом вызовов” – а не статически построенным полным графом вызовов, – облегчает анализ, но вводит в необходимость стадию промежуточного прогона программы, для которой необходимо подавать близкие к реальному использованию данные, чтобы получить реалистичные граф вызовов и

профиль. Во время промежуточного прогона нагруженная инструментированием программа будет работать в несколько раз медленнее.

8.1 Результаты тестирования масштабированной версии LLVM

На данный момент получены результаты разбиения модулей на группы и их распараллеливания. Тестирование проводилось на тестах Cray, Evas, Coremark, Lame, Clucene и Lzma с использованием архитектуры x86-64.

Имя теста	**Компоновка в 1 поток**	**Компоновка в 4 потока**	**Ускорение сборки**
Cray	0.349s	0.307s	12%
Evas	4m49.425s	3m2.489s	37%
Coremark	0.381s	0.425s	-11%
Lame	23.466s	22.537s	4%
Clucene	51.900s	51.548s	<1%
Lzma	15.3s	14.6s	4%
Smallpt	0.35s	0.32s	8%

Табл. 2. Результаты тестирования сборки с разбиением на группы для связывания

Такое заметное ускорение теста Evas связано прежде всего с его большим размером и большому количеству файлов исходного кода. Проседание производительности на smallpt связано же, наоборот, с тем, что тест очень маленький.

Имя теста	**Запуск теста, собранного в 1 поток**	**Запуск теста, собранного в 4 потока**	**Ускорение работы теста**
Cray	2m48.354s	2m48.218s	-
Evas	19.039fps	19.038fps	-
Coremark	16.300s	16.230s	<1%
Lame	23.466s	22.537s	4%
Clucene	12m30s	12m21s	1%
lzma	50.31s	50.59s	-5%
smallpt	17.11s	17.13s	-1%

Табл. 3. Результаты измерения производительности тестов, собранных с разделением на группы

Как видно из таблицы, тесты в большинстве случаев не потеряли в производительности. Лишь только lzma и smallpt немного ухудшились ввиду отсутствия учета связей в модулях во время разбиения на группы.

В настоящий момент ведутся работы по модификации проходов для использования аннотаций, а также по реализации ленивой загрузки функций.

Литература

[1]. Andrew Ayers, Stuart de Jong, John Peyton, and Richard Schooler. Scalable cross-module optimization. SIGPLAN Not., 33(5):301–312, 1998. ISSN 0362-1340. doi:http://doi.acm.org/10.1145/277652.277745.

[2]. Andrew Ayers, Richard Schooler, and Robert Gottlieb. Aggressive inlining. SIGPLAN Not., 32(5):134–145, 1997. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/258916.258928.

[3]. Sungdo Moon, Xinliang D. Li, Robert Hundt, Dhruva R. Chakrabarti, Luis A. Lozano, Uma Srinivasan, and Shin-Ming Liu. SYZYGY - a framework for scalable cross-module IPO. In CGO '04: Proceedings of the international symposium on Code generation and optimization, page 65, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2102-9.

[4]. Hewlett-Packard Company, "ELF Object File Format", http://devrsrc1.external.hp.com/STKT/partner/elf-64-hp.pdf.

[5]. The Open64 Compiler Suite. www.open64.net. URL http://www.open64.net.

[6]. Gautam Chakrabarti, Fred Chow. Structure Layout Optimizations in the Open64 Compiler: Design, Implementation and Measurements.

[7]. Preston Briggs, Doug Evans, Brian Grant, Robert Hundt, William Maddox, Diego Novillo, Seongbae Park, David Sehr, Ian Taylor, Ollie Wild. WHOPR - Fast and Scalable Whole Program Optimizations in GCC. Initial Draft, 12 Dec 2007.

[8]. Taras Glek, Jan Hubička. Optimizing real-world applications with GCC Link Time Optimization. 3 Nov 2010.

[9]. Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis and transformation. In Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO04), March 2004.

[10]. Xinliang David Li, Raksit Ashok, Robert Hundt. Lightweight Feedback-Directed Cross-Module Optimization. CGO'10, April 24–28, 2010, Toronto, Ontario, Canada. ACM 978-1-60558-635-9/10/04.

[11]. Ануфриенко Андрей Владимирович. Межпроцедурный анализ и оптимизации (I), 29 октября 2013, http://habrahabr.ru/company/intel/blog/199112.

[12]. Принцип Парето. Википедия, Свободная Энциклопедия. http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%9F%D0%B0%D1%80%D0%B5%D1%82%D0%BE

[13]. С.С. Гайсарян, Ш.Ф. Курмангалеев, К.Ю. Долгорукова, В.В. Савченко, С.С. Саргсян. Применение метода двухфазной компиляции на основе LLVM для распространения приложений с использованием облачного хранилища. Стр. 315-326. Труды Института системного программирования РАН, том 26, 2014 г. Выпуск 1. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print), DOI: 10.15514/ISPRAS-2014-26(1)-11

Overview of Scalable Frameworks of Cross-Module Optimization

Ksenia Dolgorukova
<unerkannt@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. Most applications consist of separated modules. Optimization of such applications during building in the traditional way "compiling-linking" is reduced to a single module optimization. However, in spite of support of cross-module optimizations by many compilers full optimizations are often unacceptable for large applications because of significant time and memory consumption. Compiler frameworks that could solve these problems with regard to hardware capabilities and user demands are called scalable cross-module optimization frameworks. In this article, we aim to consider all the different approaches to the problem of scalability of with respect to resources consumed rather than inter-procedural or cross-module optimizations or their effectivenesses because of their independence of used framework. The possibility of parallelization of various stages of whole program compilation as well as ways to save memory consumed by cross-module analysis are of interest. This overview observes several compiler frameworks for general-purpose programming languages like C, C++ and Fortran. Although, quite often these frameworks are able to optimize other languages in case of existence of corresponding front-ends.

Keywords: compilers, inter-procedural optimization, cross-module analysis and optimization.

References

[1]. Andrew Ayers, Stuart de Jong, John Peyton, and Richard Schooler. Scalable cross-module optimization. SIGPLAN Not., 33(5):301–312, 1998. ISSN 0362-1340. doi:http://doi.acm.org/10.1145/277652.277745.

[2]. Andrew Ayers, Richard Schooler, and Robert Gottlieb. Aggressive inlining. SIGPLAN Not., 32(5):134–145, 1997. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/258916.258928.

[3]. Sungdo Moon, Xinliang D. Li, Robert Hundt, Dhruva R. Chakrabarti, Luis A. Lozano, Uma Srinivasan, and Shin-Ming Liu. SYZYGY - a framework for scalable cross-module IPO. In CGO '04: Proceedings of the international symposium on Code generation and optimization, page 65, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2102-9.

[4]. Hewlett-Packard Company, "ELF Object File Format", http://devrsrc1.external.hp.com/STKT/partner/elf-64-hp.pdf.

[5]. The Open64 Compiler Suite. www.open64.net. URL http://www.open64.net.

[6]. Gautam Chakrabarti, Fred Chow. Structure Layout Optimizations in the Open64 Compiler: Design, Implementation and Measurements.
[7]. Preston Briggs, Doug Evans, Brian Grant, Robert Hundt, William Maddox, Diego Novillo, Seongbae Park, David Sehr, Ian Taylor, Ollie Wild. WHOPR - Fast and Scalable Whole Program Optimizations in GCC. Initial Draft, 12 Dec 2007.
[8]. Taras Glek, Jan Hubička. Optimizing real-world applications with GCC Link Time Optimization. 3 Nov 2010.
[9]. Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis and transformation. In Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO04), March 2004.
[10]. Xinliang David Li, Raksit Ashok, Robert Hundt. Lightweight Feedback-Directed Cross-Module Optimization. CGO'10, April 24–28, 2010, Toronto, Ontario, Canada. ACM 978-1-60558-635-9/10/04.
[11]. Anufrienko Andrej Vladimirovich. Mezhprotsedurnyj analiz i optimizatsii (I) [Interprocedural analysis and optimizations (I)], October 29, 2013, http://habrahabr.ru/company/intel/blog/199112.
[12]. Pareto principle. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Pareto_principle
[13]. Sergey Gaissaryan, Shamil Kurmangaleev, Kseniya Dolgorukova, Valery Savchenko, Sevak Sargsyan. Primenenie metoda dvukhfaznoj kompilyatsii na osnove LLVM dlya rasprostraneniya prilozhenij s ispol'zovaniem oblachnogo khranilishha. [Applying two-stage LLVM-based compilation approach to application deployment via cloud storage]. Trudy Instituta sistemnogo programmirovaniya RAN [Proceedings of the Institute for System Programming of RAS], vol.26, 2014, Issue 1. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print) pp. 315-326. DOI: 10.15514/ISPRAS-2014-26(1)-11.

Оптимизация метода решения линейных систем уравнений в OpenFOAM для платформы MPI + CUDA[1]

А.В. Монаков
<amonakov@ispras.ru>
В.А. Платонов
<soh@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, 25.

Аннотация. Описываются работы по оптимизации решателя систем линейных уравнений пакета OpenFOAM для гетерогенной вычислительной системы с GPU-ускорителями в модели MPI + CUDA. В ходе работ предложена параллельная GPU-версия метода предобуславливания AINV и проведен ряд специфичных GPU-оптимизаций, в основном направленных на минимизацию обменов данными с GPU. Представлены экспериментальные результаты тестирования решателя на некоторых задачах OpenFOAM.

Ключевые слова: метод сопряженных градиентов, AINV предобуславливание, OpenFOAM, GPU, MPI

1. Введение

Одним из перспективных методов повышения производительности высокопроизводительных систем является использование гетерогенных систем, в которых многоядерные процессоры дополняются акселераторами, которые за счет специализированной архитектуры позволяют получить высокую производительность для определенных классов задач. В настоящее время с такой целью широко используются графические акселераторы (GPU), архитектура которых оптимизирована для задач с высоким параллелизмом.

В работе рассматривается задача ускорения расчетов в пакете OpenFOAM на кластерах с узлами, оборудованными GPU. OpenFOAM [1] является крупным открытым пакетом, ориентированным в основном на задачи вычислительной гидродинамики. Распараллеливание расчетов в OpenFOAM выполнено на

1 Работа поддержана грантом РФФИ 13-07-12102

основе разделения расчетной области между MPI-процессами. Схема распараллеливания является одноуровневой: каждый MPI-процесс является однопоточным, хотя более предпочтительной может быть двухуровневая схема, в которой MPI-процессы используют механизмы распараллеливания на общей памяти, такие как OpenMP.

Большинство расчетов в OpenFOAM сводится к многократному решению линейных систем. Матрицы этих систем являются сильно разреженными, и расположение ненулевых элементов полностью определяется графом смежности ячеек расчетной сетки (за исключением блочных систем размерность матрицы равна количеству ячеек сетки, а количество ненулевых элементов выше главной диагонали равно количеству внутренних граней). Для решения систем применяются итерационные методы. Обычно основное время счета приходится на решение симметричных линейных систем, причем для решения используется метод сопряженных градиентов или многосеточный метод. Таким образом, перенос итерационных методов решения на акселератор может позволить существенно сократить общее время счета. В этой статье рассматривается перенос метода сопряженных градиентов с простым предобуславливателем на GPU-акселераторы.

2. Метод сопряженных градиентов на GPU

В OpenFOAM используется классический вариант метода сопряженных градиентов с предобуславливанием [2] для системы $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$:

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$
2: $\boldsymbol{p} = \boldsymbol{0}$
3: do
4: $\boldsymbol{s} = \boldsymbol{M}^{-1}\boldsymbol{r}$
5: $\delta_i = (\boldsymbol{s}, \boldsymbol{r})$
6: $\beta = \delta_i / \delta_{i-1}$
7: $\boldsymbol{p} = \boldsymbol{s} + \beta\boldsymbol{p}$
8: $\boldsymbol{s} = \boldsymbol{A}\,\boldsymbol{p}$
9: $\gamma = (\boldsymbol{s}, \boldsymbol{p})$
10: $\alpha = \delta_i / \gamma$
11: $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{p}$
12: $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{s}$
13: while (!stopCondition(|$\boldsymbol{r}$|, ++i))

(где $\boldsymbol{M}^{-1}$ — предобуславливатель, $\boldsymbol{r}$ — вектор невязки).

В случае многопроцессного счета на каждой итерации алгоритма выполняются следующие MPI-коммуникации:

- три глобальные редукции: в операциях взятия скалярного произведения и нормы вектора;
- обмен приграничными коэффициентами вектора $\boldsymbol{p}$ в операции домножения на матрицу $\boldsymbol{A}$ во всех парах MPI-процессов, имеющих общую границу.

При переносе описанного алгоритма на GPU необходимы реализации домножения разреженной матрицы на вектор и предобуславливателя. Для больших систем производительность этих операций является определяющей.

В нашей реализации используется ранее разработанная библиотека, использующая автоматическую настройку для повышения производительности на GPU [3]. Кроме этого, необходимы реализации взятия скалярного произведения, нормы, линейной комбинации векторов на GPU.

Отметим, что в случае однопоточного счета без нетривиальных граничных условий алгоритм не требует синхронизации CPU и GPU за исключением копирования и проверки нормы невязки $|\boldsymbol{r}|$. Копирование можно выполнять асинхронно, обеспечив перекрытие с дальнейшими операциями на GPU. Все скалярные коэффициенты (α, β, γ, δ_i) достаточно хранить в памяти акселератора. Для этого требуются следущие простые модификации:

- процедуры, выполняющие операции линейной алгебры на GPU должны принимать коэффициенты как указатели в память акселератора;
- обновления коэффициентов ($\beta = \delta_i / \delta_{i-1}$, $\alpha = \delta_i / \gamma$) должны выполняться как запуск тривиального ядра на GPU из одной нити.

При многопоточном счете коэффициенты δ_i и γ являются результатами MPI-редукций, и поэтому не могут храниться только в памяти акселератора. Для достижения хорошей эффективности предлагается хранить коэффициенты, участвующие в MPI-редукциях в pinned-памяти: в специально зарегистрированных страницах оперативной памяти, которые доступны для модификации с акселератора (однако стоимость доступа высока, так он выполняется через шину PCI-Express). Коэффициенты, которые не участвуют в MPI-редукциях (α, β) необходимо хранить только в памяти акселератора, так как далее они многократно читаются в ядрах, выполняющих операции над векторами.

3. Предобуславливание

Выбор эффективного метода предобуславливания для GPU не очевиден. В OpenFOAM обычно используется вариант ILU-предобуславливания, но алгоритмы этого семейства плохо подходят для GPU, так как применение такого предобуславливателя по сути представляет собой решение разреженных треугольных систем уравнений, и не обладает высоким

параллелизмом. Хотя параллелизм ILU-предобуславливателей можно повышать за счет изменения нумерации сетки, это приводит к снижению эффективности предобуславливателя.

Существует ряд предобуславливателей, основанных на построении приближенной обратной матрицы системы в явном виде [4]. Применение такого предобуславливателя сводится к домножению на одну или две разреженные матрицы — операции, для которой в любом случае нужна эффективная реализация на GPU. С другой стороны, вычисление такого предобуславливателя является более дорогостоящей операцией по сравнению с ILU.

В нашей реализации используется метод AINV [5], в котором приближенная обратная матрица строится как произведение двух разреженных треугольных матриц:

$$W^T A Z \approx D$$

(W, Z — верхнетреугольные матрицы с единицами на диагонали, D — диагональная матрица). Структура разреженности множителей W и Z определяется скалярным параметром drop tolerance: при построении множителей все элементы, по модулю меньшие drop tolerance, отбрасываются.

Для повышения производительности используются следующие модификации:

- Предобуславливатель хранится с использованием вещественных чисел одинарной точности (float). Поскольку операция домножения на разреженную матрицу ограничена пропускной способностью памяти, а высокая точность для предобуславливателя не требуется, эта оптимизация допустима и ускоряет применение предобуславливателя.
- Хотя формально применение предобуславливателя состоит из домножения на две разреженные матрицы и одну диагональную, при подготовке предобуславливателя все диагональные коэффициенты можно внести в разреженные множители. Это позволяет сократить количество необходимых доступов к памяти.
- Применение предобуславливателя требует домножения на одну верхнетреугольную и одну нижнетреугольную матрицу. При домножении на нижнетреугольную матрицу первыми обрабатываются сравнительно незаполненные строки матрицы, а более плотные строки у «основания» треугольника начинают обрабатываться в последнюю очередь. Для более эффективно сбалансированной нагрузки желательно, наоборот, запускать CUDA-блоки, обрабатывающие наиболее заполненные группы строк в первую очередь. Для этого предлагается переупорядочить разреженные множители AINV в лево-верхнетреугольную форму: пусть предобуславливатель имеет вид $\boldsymbol{UL}$ ($\boldsymbol{L} = \boldsymbol{U}^T$), $\boldsymbol{J}$ — обменная (антидиагональная единичная) матрица; тогда $\boldsymbol{UL} = \boldsymbol{U}(\boldsymbol{JJ})\boldsymbol{L} = (\boldsymbol{UJ})(\boldsymbol{JL})$, где оба множителя имеют лево-вернетреугольную форму и получаются из оригинальных обращением порядка строк или

столбцов. Построение предобуславливателя в таком виде никак не влияет на остальную логику и повышает эффективность работы.

- Наконец, необходимо сократить затраты на вычисление предобуславливателя: однопоточная реализация подсчета AINV на процессоре требует времени, сопоставимого с дальнейшим решением линейной системы на GPU с этим предобуславливателем. Амортизировать эти расходы предлагается следующим образом. Во-первых, разрешить переиспользование предобуславливателя для одной системы на соседних шагах по времени. Во-вторых, переиспользование предобуславливателя открывает возможность вычислять его асинхронно в отдельной нити, параллельно с решением системы на GPU. Асинхронное вычисление предобуславливателя привносит недетерминизм в расчеты, так как моменты обновления предобуславливателя на GPU могут отличаться от запуска к запуску. Кроме того, при выходе из цикла метода сопряженных градиентов желательно останавливать побочную нить, чтобы она не конкурировала с основными расчетами за ресурсы CPU.

4. Поддержка параллельных расчетов

В OpenFOAM параллельный запуск через MPI поддерживается за счет разбиения сетки. Граничные грани получаемых подсеток принадлежат либо множеству оригинальных граничных граней, либо множеству граней процессорных границ, возникших в результате разбиения. Процессорные границы в OpenFOAM обрабатываются как частный случай общих граничных условий (однако в отличие от большинства граничных условий, обусловленных решаемой задачей и задаваемых пользователем, процессорные граничные условия возникают автоматически при разбиении расчетной области между процессами). Все граничные условия обрабатываются при выполнении операции $\boldsymbol{s} = \boldsymbol{A}\,\boldsymbol{p}$ в дополнение к домножению на явно хранимые коэффициенты матрицы. Соответственно, при параллельном счете корректная обработка приграничных коэффициентов обеспечивается неявно за счет виртуальных процессорных граничных условий (в OpenFOAM граничные условия образуют иерархию C++ классов, и обновление коэффициентов реализовано через виртуальные функции). С другой стороны, даже при однопоточном счете эффект от домножения на матрицу системы определяется не только явно хранимыми коэффициентами, но еще и набором граничных условий для текущей сетки.

Таким образом, в GPU-реализации итерационных методов необходимо поддерживать обработку граничных условий в умножении на матрицу системы, и этого будет достаточно для поддержки параллельных запусков. Обработка граничных коэффициентов в OpenFOAM выполняется запуском методов initMatrixInterfaces и updateMatrixInterfaces класса lduMatrix, вызывающих в свою очередь виртуальные методы initInterfaceMatrixUpdate и

updateInterfaceMatrix класса lduInterfaceField. В случае процессорных границ первый метод запускает пересылку локальных граничных коэффициентов, а второй принимает граничные коэффициенты «соседнего» процессора.

Для GPU эти шаги предлагается выполнять следующим образом. Перед выполнением initMatrixInterfaces на акселераторе запускается ядро, которое копирует все граничные коэффициенты в один плотный вектор; для этого по сетке заранее строится индексный массив граничных коэффициентов. Полученный вектор копируется во вспомогательный массив, лежащий в pinned-памяти. Это копирование можно перекрыть с домножением на матрицу на GPU. Далее граничные коэффициенты из вспомогательного массива копируются в массив в соответствии со своей оригинальной нумерацией, и этот массив используется для запуска initMatrixInterfaces. Далее, после выполнения updateMatrixInterfaces нужно проделать аналогичные шаги в обратном порядке: собрать обновленные коэффициенты в плотный массив на процессоре, скопировать на GPU и просуммировать с результатами домножения на матрицу системы. Для повышения эффективности updateMatrixInterfaces и последующее копирование в память акселератора можно перекрывать с другими ядрами на GPU.

5. Перекрытие MPI-коммуникаций

В методе сопряженных градиентов MPI-коммуникации возникают в четырех местах: в умножении на матрицу системы, взятии нормы вектора невязки и вычислении скалярных произведений. Хотя MPI-коммуникации крайне желательно перекрывать с вычислениями, это не выполняется для глобальных редукций после скалярых произведений, которые сразу же используются для последующих вычислений. Обмен граничными коэффициентами в OpenFOAM производится параллельно с умножением на матрицу локально, а взятие нормы вектора невязки теоретически можно выполнять параллельно с другими вычислениями со следующей итерации (однако необходимые для этого вызовы для асинхронных редукций появились только в последней ревизии стандарта MPI: 3.0, и не используются в OpenFOAM).

Таким образом, на каждой итерации цикла сопряженных градиентов выполняются две или три синхронные редукции, во время которых не выполняется никаких вычислений. Для гетерогенных систем с акселераторами проблема усугубляется тем, что перед запуском MPI-редукции необходимо дождаться окончания вычисления нужного коэффициента на акселераторе; при этом желательно избежать полной CPU-GPU синхронизации, заранее запустив другие независимые вычисления на акселераторе.

Существует несколько реализаций метода сопряженных градиентов, направленных на увеличение перекрытия вычислений и коммуникаций. В них возникает компромисс между степенью перекрытия и количеством дополнительно проводимых вычислений и хранимых векторов. Для использования на GPU нами был использован следующий вариант [6]:

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}$
2: $\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{r}$
3: $\boldsymbol{w} = \boldsymbol{Au}$
4: $\gamma_{-1} = \infty$
5: $\gamma_0 = (\boldsymbol{r}, \boldsymbol{u})$
6: $\delta = (\boldsymbol{w}, \boldsymbol{u})$
7: do
8: $\boldsymbol{m} = \boldsymbol{M}^{-1}\boldsymbol{w}$
9: $\boldsymbol{n} = \boldsymbol{Am}$
10: $\beta = \gamma_i / \gamma_{i-1}$
11: $\alpha = \gamma_i / (\delta - \beta\, \gamma_i / \alpha)$
12: $\boldsymbol{z} = \boldsymbol{n} + \beta\boldsymbol{z}$
13: $\boldsymbol{q} = \boldsymbol{m} + \beta\boldsymbol{q}$
14: $\boldsymbol{s} = \boldsymbol{w} + \beta\boldsymbol{s}$
15: $\boldsymbol{p} = \boldsymbol{u} + \beta\boldsymbol{p}$
16: $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{p}$
17: $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{s}$
18: $\boldsymbol{u} = \boldsymbol{u} - \alpha\boldsymbol{q}$
19: $\boldsymbol{w} = \boldsymbol{w} - \alpha\boldsymbol{z}$
20: $\gamma_i = (\boldsymbol{r}, \boldsymbol{u})$
21: $\delta = (\boldsymbol{w}, \boldsymbol{u})$
22: while (!stopCondition(|***r***|, ++i))

Приведенный алгоритм обладает следующими свойствами:

- глобальные редукции в операциях взятия скалярных произведений и нормы невязки можно полностью перекрыть с умножением на предобуславливатель и матрицу на следующей итерации;
- все три редукции можно выполнять одновременно как один вызов all-reduce на трехкомпонентном векторе $\{\gamma_i, \delta, |\boldsymbol{r}|\}$;
- алгоритм требует меньше ядер на итерацию: вычисления в строках 12-22 требуют запуска лишь двух ядер (первое вычисляет все линейные комбинации и частичные скалярные произведения/нормы в пределах блока нитей; второе ядро состоит из одного блока и собирает частичные суммы в одну);
- обновление граничных коэффициентов при умножении на матрицу можно перекрыть с обновлением векторов ***q***, ***s***, ***p***, ***x***, ***r***, ***u***, если вынести обновление векторов ***z*** и ***w*** в отдельное ядро.

Недостатком этого алгоритма является увеличение количества хранимых на акселераторе векторов и доступов к памяти на каждой итерации. Поскольку пропускная способность памяти является определяющим фактором в производительности, при небольшой стоимости коммуникаций этот метод будет показывать более низкую производительность по сравнению с обычной реализацией.

6. Результаты тестирования

Описанные оптимизации были реализованы и протестированы в модуле решения линейных систем для проекта FOAM-Extend. Для тестирования производительности использовались гетерогенные узлы с процессорами Xeon E5 2690v2 (10 ядер, 3.0 GHz) и акселераторами Tesla K20X. В качестве эталонной процессорной реализации использовался метод PCG с предобуславливателем DIC. Многосеточный метод (GAMG) во всех случаях показывает более высокую производительность.

На тесте motorBike (сетка с 3.6 млн. ячеек) с приложением pisoFoam были получены следующие результаты:

- при запуске 10 процессов и использованием 1 GPU, 100 шагов:
  - с использованием только процессоров:
    - 1136 секунд в методе сопряженных градиентов
    - 2046 секунд общее время счета;
  - с использованием гибридной реализации:
    - 649 секунд в методе сопряженных градиентов;
    - 1536 секунд общее время;
- при запуске 80 процессов и использованием 8 GPU, 500 шагов:
  - с использованием только процессоров:
    - 502 секунды в методе сопряженных градиентов
    - 926 секунд общее время счета;
  - с использованием гибридной реализации:
    - 651 секунда в методе сопряженных градиентов;
    - 1093 секунды общее время.

Конфигурация с одним GPU демонстрирует ускорение в 1.75 раз в методе решения линейных систем. При использовании 8 GPU наблюдается замедление от использования акселераторов. Одной из проблем является низкая эффективность работы акселераторов из-за относительно малого количества элементов (по нашему опыту, для достижения высокой пропускной способности памяти в нашей реализации вычислительные ядра должны обрабатывать векторы с размерностью больше миллиона). С другой стороны, при переходе к 80 ядрам процессорная реализация показывает

сверхлинейное ускорение, так как больше обращений попадает в процессорные кеши.

Табл. 1. Масштабирование GPU-реализации.

Количество акселераторов								
Размер сетки	1	2	3	4	6	8	12	18
Время в методе сопряженных градиентов, с								
2 млн.	220	116	95	71	58			
8 млн.				320	253	172	142	107
Относительная эффективность параллельной работы								
2 млн.	1	0.95	0.77	0.77	0.63			
8 млн.				1	0.8	0.93	0.75	0.66

На синтетическом тесте cavity3d (сетки с 2 и 8 млн. ячеек) с приложением icoFoam проводились запуски для оценки масштабирования GPU-солвера. Использовались акселераторы Tesla K40, с количеством MPI-процессов равным количеству использованных акселераторов. В табл. 1 показаны экспериментальные результаты. Видно, что при менее чем 1 млн. ячеек на акселератор происходит снижение эффективности.

7. Заключение.

В данной работе представлено несколько оптимизаций для гибридной реализации метода сопряженных градиентов. С предложенными оптимизациями на GPU-акселераторах достигается более высокая производительность по сравнению с многоядерными процессорами, но только для задач достаточно большого размера. В будущем планируется исследовать возможности дальнейшего повышения производительности, реализовать метод решения для блочных систем в OpenFOAM (когда уравнения для давления и компонент скоростей решаются совместно) и разработать гибридный вариант для многосеточного метода.

Список литературы

[1]. Страница OpenFOAM — http://openfoam.org/
[2]. Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM. 2003.
[3]. А. Монаков, Е. Велесевич, В. Платонов, А. Аветисян. Инструменты анализа и разработки эффективного кода для параллельных архитектур. Труды Института

системного программирования РАН, том 26, 2014 г. Выпуск 1. Стр. 357-374. DOI: 10.15514/ISPRAS-2014-26(1)-14

[4]. M. Benzi. Preconditioning Techniques for Large Linear Systems: A Survey. Journal of Computational Physics, 182 (2002), pp. 418-477.

[5]. R. Bridson, W.-P. Tang. Refining an Approximate Inverse. Journal on Computational and Applied Math, 123 (2000), Numerical Analysis 2000 vol. III: Linear Algebra, pp. 293-306.

[6]. P. Ghysels, W. Vanroose. Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm. Submitted to Parallel Computing, 2012.

Optimizations for linear solvers in OpenFOAM for MPI + CUDA platform [2]

A. Monakov
<amonakov@ispras.ru>
V. Platonov
<soh@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. We describe an implementation of conjugate gradient method on heterogeneous platforms (multiple nodes with GPU accelerators) to be used in OpenFOAM. Several optimizations are described. For conjugate gradient itself, we suggest using device memory for scalars used only on the GPU and pinned memory for scalars used in MPI reductions. For preconditioning, we choose AINV as a suitable preconditioner for GPUs and describe ways to make it more efficient, such as storing in it single precision, laying out factors in upper-left triangular form and computing it on the CPU asynchronously. We describe how multi-GPU computing can be supported together with arbitrary boundary conditions by copying only boundary coefficients from the accelerator to host memory and then using existing OpenFOAM methods on the CPU. To improve overlap of computations and communications, we suggest using a pipelined variant of the conjugate gradient method and describe GPU-specific adjustments. In experimental evaluation, we obtain a 1.75x speedup in the linear solver by using a Tesla K20X accelerator in addition to a 10-core Xeon CPU, but only for sufficiently large problem sizes: below 1 million cells per accelerator the efficiency of GPU computations dimishes.

Keywords: conjugate gradient method, AINV preconditioning, OpenFOAM, GPU, MPI.

References

[1]. OpenFOAM — http://openfoam.org/
[2]. Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM. 2003.
[3]. A. Monakov, E. Velesevich, V. Platonov, A. Avetisyan. Instrumenty analiza i razrabotki effektivnogo koda dlya parallel'nykh arkhitektur [Analysis and development tools for efficient programs on parallel architectures]. Trudy ISP RAN [The Proceedings of ISP RAS], 2014, vol. 26, no 1. pp. 357-374 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-14
[4]. M. Benzi. Preconditioning Techniques for Large Linear Systems: A Survey. Journal of Computational Physics, 182 (2002), pp. 418-477.
[5]. R. Bridson, W.-P. Tang. Refining an Approximate Inverse. Journal on Computational and Applied Math, 123 (2000), Numerical Analysis 2000 vol. III: Linear Algebra, pp. 293-306.

2 The paper is supported by RFBR grant 13-07-12102

[6]. P. Ghysels, W. Vanroose. Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm. Submitted to Parallel Computing, 2012.

Статический поиск ошибок повторной блокировки семафора

А. Е. Бородин
alexey.borodin@ispras.ru
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. В статье описывается алгоритм статического поиска ошибки повторной блокировки семафоров, позволяющий выдавать предупреждения с низким уровнем ложных срабатываний. Поиск ошибок рассмотрен для абстрактной библиотеки, включающей функции блокировки семафора, разблокировки семафора и условной блокировки семафора. Определено множество регулярных языков, моделирующих блокировки и разблокировки при конкретном исполнении программы. Определён домен, аппроксимирующий множество регулярных языков. Алгоритм реализован в терминах анализа потока данных. При анализе элементы домена используются в качестве свойств потока данных. Алгоритм описан для программы с одним семафором и без алиасов. В этом случае каждое выданное предупреждение должно соответствовать ошибке при конкретном выполнении. Алгоритм реализован в системе статического анализа Svace, разрабатываемой в Институте системного программирования Российской академии наук. Svace осуществляет анализ алиасов и сопоставление формальных и фактических параметров при вызове функции. Благодаря этому можно применять алгоритм поиска повторных блокировок для программы, содержащей только один семафор, а вся остальная работа будет выполнена системой Svace. Алгоритм поиска повтороных блокировок реализованный в Svace может выдавать некоторое количество ложных срабатываний, т. к. Svace выполняет неконсервативный анализ.

Ключевые слова: статический анализ; семафор; ошибка повтороной блокировки; анализ потока данных; регулярные языки.

1. Введение

Одной из ошибок использования примитивов синхронизации семафоров является повторная блокировка семафора, которая может привести к тупиковой ситуации при выполнении приложения. Конкретное поведение зависит от операционной системы и используемых библиотек.

Семафор - это примитив синхронизации, который может находиться в двух состояниях: открытом и закрытом. С помощью функции блокировки семафор из открытого состояния переводится в закрытое состояние. Блокировка семафора, уже находящегося в закрытом состоянии, может вызвать проблемы

в зависимости от операционной системы, типа семафора и настроек. Например, в операционных системах, поддерживающих стандарт POSIX, семафоры могут иметь следующие типы [1]:

PTHREAD_MUTEX_NORMAL — нет контроля повторного захвата тем же потоком;
PTHREAD_MUTEX_RECURSIVE — повторные захваты тем же потоком допустимы, ведётся счётчик таких захватов;
PTHREAD_MUTEX_ERRORCHECK — повторные захваты тем же потоком вызывают немедленную ошибку.

Таким образом повторная блокировка семафора может привести к тупиковой ситуации для PTHREAD_MUTEX_NORMAL или к ошибке времени выполнения для PTHREAD_MUTEX_ERRORCHEC.

В данной работе рассматривается алгоритм поиска ошибок повторной блокировки с помощью статического анализа исходного текста программы.

Поиск ошибок будет рассмотрен для абстрактной библиотеки, функции которой показаны на рис.1.

```
struct MUTEX;
//переводит семафор в закрытое состояние.
void lock(MUTEX*);
//переводит семафор в открытое состояние.
void unlock(MUTEX*);
//если семафор находится в открытом состоянии,
//переводит семафор в закрытое состояние и в случае успеха возвращает 0.
int trylock(MUTEX*);
```

Рис. 1. Абстрактная библиотека работы с семафорами.

При использовании на конкретной платформе эти функции могут быть легко заменены на соответствующие аналоги.

2. Модель повторной блокировки

Рассмотрим процедуры [1] lock и unlock абстрактной библиотеки работы с семафорами. Процедура trylock будет описана ниже.

1 Функции языка Си здесь будут называться процедурами, чтобы избежать неоднозначностей с математическими функциями.

Процедура lock закрывает семафор, если семафор уже был закрыт, то произойдёт ошибка повторной блокировки. Процедура unlock переводит семафор в открытое состояние. Таким образом семафор может находиться в трёх состояниях: открытом, закрытом и ошибочном. Переходы между состояниями происходят только при вызове процедур lock и unlock. Вызов любой другой библиотечной процедуры не меняет состояния семафора. Начальным состоянием является открытое состояние.

Для программы, содержащей один семафор, рассмотрим конкретное выполнение (трассу) некоторой процедуры.

Обозначим I - множество инструкций процедуры. Обозначим L = {u, l}. Определим функцию IL: I → L*, сопоставляющую каждой инструкции последовательность меток из L следующим образом: IL возвращает [l] для инструкции вызова процедуры lock; [u] для инструкции вызова unlock и [] для всех остальных инструкций.

Пусть T множество конечных и бесконечных трасс (последовательностей элементов I). Для t ∈ T, |t| - обозначает длину трассы, для бесконечных трасс |t| = ω. t_k∈I (для k-1 < |t|) - k-й элемент трассы t. $t_{1..k}$ – префикс трассы, содержащий k элементов.

Введём функцию S: T → L*, отображающую трассы в последовательности меток, такую что

$S(t_{1..1}) = IL(t_1)$

$S(t_{1..k}) = S(t_{1..k-1}) \cdot IL(t_k)$, где k-1 < |t|.

Состоянием шага k трассы t будем называть $S(t_{1..k})$.

Будем говорить, что трасса содержит нелокальную повторную блокировку, если для некоторой инструкции вызова процедуры входящей в трассу, сегмент трассы, соответствующий исполнению этой инструкции, отображается функцией S на последовательность меток, содержащую подстроку ll.

Пусть T* - множество трасс, не содержащих нелокальные повторные блокировки. Рассмотрим множество префиксов трасс P ⊆ T*, заканчивающихся в некоторой точке программы. Накапливающей семантикой [2] данной точки будем называть множество {S(t) | t ∈ P}.

Для выполнения анализа будем приближать накапливающую семантику ребёр графа потока управления регулярными языками. Определим множество регулярных языков B ={ε, σ, π, e}, где e = L*llL*, ε = {ε}, σ = L*l, π = lL*.

Пусть C получено замыканием B относительно операции пересечения множеств, дополненное элементом ⊤ = L* (⊤ можно рассматривать как пересечение пустого набора множеств). Множество D определяется как замыкание C относительно операции объединения. Тогда (D,⊆) составляет полную решетку.

При анализе элементы D будут использоваться в качестве свойств потока данных. Будем говорить, что элемент d∈D корректен для точки программы, если накапливающая семантика данной точки является подмножеством d.

Интуитивно элемент e обозначает множество трасс выполнения программ, содержащих повторную блокировку; элемент σ – множество трасс, заканчивающихся блокировкой; элемент π – множество трасс, начинающихся с блокировки; ε – трассы без блокировок и разблокировок, ⊤ - все возможные трассы.

Примеры описываемых множеств последовательностей:

$ulul \subseteq \sigma$; $lul \subseteq (\pi \cap \sigma)$; $\{ulul, \varepsilon\} \subseteq (\sigma \cup \varepsilon)$.

Определение. Бинарный оператор x: DxD->D будем называть приближенной конкатенацией, если конкатенация любых элементов d1, d2 из D является подмножеством d1xd2.

Определим бинарный оператор ∘: DxD->D следующими правилами:

- Пусть c, c' ∈ C.
  - $c \circ \emptyset = \emptyset \circ c = \emptyset$.
  - $\varepsilon \circ c = c \circ \varepsilon = c$.
  - Если $c \subseteq \pi$, $c' \neq \varepsilon$, то $c \circ c' = \pi \cap c'$.
  - Если $c' \subseteq \sigma$, то $c \circ c' = \sigma \cap c$.
  - Если $c \subseteq \sigma$ и $c' \subseteq \pi$, то $c \circ c' = e$.
  - Иначе, если $c \subseteq e$ то $c \circ c' = c' \circ c = e$.
  - Иначе, $c \circ c' = \top$.
- Если d, d' ∈ D, где $d = c_1 \cup ... \cup c_m$, $d' = c_1' \cup ... \cup c_n'$, то $d \circ d' = \bigcup (c_i \cdot c'_j)$, где $0 < i \leq m$, $0 < j \leq n$.

Лемма 1. Оператор ∘ является приближённой конкатенацией.

Доказательство. 1) $c \cdot \emptyset = \emptyset \subseteq c \circ \emptyset$.

$\emptyset \cdot c = \emptyset \subseteq \emptyset \circ c$

2) $c \cdot \varepsilon = c \subseteq c \circ \varepsilon$.

$\varepsilon \cdot c = c \subseteq \varepsilon \circ c$.

3) Пусть $c \subseteq \pi$, тогда $c \cdot c' \subseteq \pi \cdot c'$ (т.к. $x \subseteq y \Rightarrow x \cdot z \subseteq y \cdot z$)

$\pi \cdot c' = lL^* \cdot c' \subseteq \pi$. Если также $\pi \cdot c' \subseteq c'$, то $\pi \cdot c' \subseteq \pi \cap c'$.

Докажем, что $\pi \cdot c' \subseteq c'$.

Если $c' = \top$, то $\pi \cdot c' = \pi \cdot \top \subseteq L^* \subseteq \top$.

Иначе $c' = b_1 \cap ... \cap b_n$, $b_i \neq \varepsilon$.

Если для любого i верно $\pi \cdot b_i \subseteq b_i$, то $\pi \cdot c' \subseteq \pi \cdot b_i \subseteq b_i$,

следовательно $\pi \cdot c' \subseteq b_1 \cap ... \cap b_n = c'$.

Осталось доказать, что $\pi \cdot b_i \subseteq b_i$, где $b_i \in \{\sigma, \pi, e\}$:

$\pi \cdot \pi = lL^* \cdot lL^* \subseteq lL^* = \pi$;

$\pi \cdot \sigma = lL^* \cdot L^*l \subseteq L^*l = \sigma$;

$\pi \cdot e = lL^* \cdot L^*llL^* \subseteq lL^*llL^* \subseteq L^*llL^* = e$.

4) Если $c' \subseteq \sigma$, то $c \circ c' = \sigma \cap c$ доказывается аналогично 3.

5) Если $c \subseteq \sigma$ и $c' \subseteq \pi$, тогда $c \cdot c' \subseteq \sigma \cdot \pi = L^*llL^* = e$.

6) Иначе, если $c \subseteq e$, то $c \cdot c' \subseteq c' \cdot e \subseteq T \cdot e = L^*L^*llL^* \subseteq L^*llL^* = e$

7) Иначе, $c \circ c' = \top$. По определению $\top$ верно, что для любого $d \in D$ верно $d \subseteq \top$, следовательно $c \cdot c' \subseteq T$.□

Результат применения оператора $\circ$ для элементов множества B приведён в табл. 1, элементы C могут быть представлены как пересечение элементов B, а элементы D как объединение элементов C.

Табл.1. Операция конкатенации для элементов B.

	π	**σ**	**e**	**ε**
π	**π**	**π⋂σ**	**π⋂e**	**σ**
σ	**e**	**σ**	**e**	**π**
e	**e**	**e⋂σ**	**e**	**e**
ε	**π**	**σ**	**e**	**ε**

3. Выполнение анализа

Выполним анализ потока данных, сопоставив рёбрам графа потока управления элементы домена D. Элемент, сопоставленный ребру выхода из процедуры назовём аннотацией. Введём функцию A, возвращающую аннотацию для каждой процедуры. Будем считать, что $A(lock) = \pi \cap \sigma$, $A(unlock) = T$.

Определим передаточные функции P_i: $D \rightarrow D$ для инструкции i программы:

1. Для инструкции i вызова процедуры p, где $A(p) = d_2$, $P_i(d_1) = d_1 \circ d_2$.
2. Для остальных инструкций $P_i(d) = d$.

Оператор сбора $\sqcup$: D x D → D будет объединением множеств .

Для шага k трассы t находим инструкцию $i = t_k$, конкретная семантика $c = S(t_{1..k})$, абстрактная семантика $d = OUT_i \in D$. В соответствии с определением оператора сбора и передаточной функцией, а также леммой 1, $c \in d$.

Если анализ потока данных сопоставляет некоторому ребру состояние e, то при конкретном выполнении программы в данном месте произойдёт повторная блокировка.

Доказательство: Достаточно доказать, что $S(t_{1..k}) \in d_i \Rightarrow S(t_{1..k+1}) \in d_{i+1}$, где k – шаг трассы для инструкции I, $d_i = P(d_{i-1})$. Если i – инструкция вызова процедуры, то передаточная функция определена через оператор $\circ$, для которого утверждение верно согласно лемме 1.

Для остальных инструкций передаточная функция является тождественной функцией, поэтому $d_{i+1} = d_i$ и $S(t_{1..k}) = S(t_{1..k+1})$.

Т.е. если на некотором шаге k состояние равно e, то $S(t_{1..k}) \in e = L^*llL^*$, что означает ситуацию с повторной блокировкой. □

Как видно из определения передаточной функции P и оператора сбора ⊔, если в программе возможна повторная блокировка, то алгоритм не обязательно найдёт её. Контрпример приведён на рис.2. Аннотация для процедуры foo A(foo) = $\pi \cap \sigma \cup \varepsilon$. Аннотация для lock – $\pi \cap \sigma$. Поэтому после повторного вызова lock состояние $P = \pi \cap \sigma \cup \varepsilon \cdot \pi \cap \sigma = \pi \cap \sigma \cap e \cup \pi \cap \sigma = \pi \cap \sigma$. Таким образом ошибка повторной блокировки не будет найдена. Причина подобной реализации в том, что неизвестно точно о взаимосвязи глобальной переменной glob и параметра flag. Если флаги взаимно исключают друг друга, то мы получим ложное срабатывание. Так как целью работы было создание алгоритма, выдающего предупреждения с низким уровнем ошибок, то мы предпочитаем не выдавать предупреждения для подобных ситуаций. Альтернативный подход может не использовать приближение $e \subseteq T$ и выдавать предупреждения о повторной блокировке в случае условных блокировок. Но так как в существующих библиотеках часто используются функции, которые в зависимости от параметра вызывают блокировку, либо разблокировку семафора, то такой подход приведёт к ложным срабатываниям для всех мест использования подобных функций. Также можно использовать разрешитель булевых формул для анализа взаимосвязей между переменными программы, но в этом случае возрастает сложность и увеличивается время анализа.

```
void foo(MUTEX* mutex) {
        if(glob) { lock(mutex); }
}
void func(int flag, MUTEX* mutex) {
        if(flag) { foo(mutex); }
        lock(mutex);
}
```

Рис. 2. Потенциальная ошибка.

Если программа содержит несколько семафоров, то алгоритм работает с каждым семафором отдельно.

4. Реализация для trylock

Функция trylock тестирует состояние семафора и если он находится в закрытом состоянии, производит блокировку. Таким образом двойной вызов trylock не приводит к ошибке повторной блокировки.

Если семафор находится в открытом состоянии, то после выполнения trylock, он будет в закрытом состоянии. Если же семафор находится в закрытом

состоянии, то функция trylock ничего не будет делать. Т.е. после вызова trylock семафор будет всегда находится в закрытом состоянии.

Формально, если d∈D – состояние перед выполнением trylock, а a – аннотация trylock, то если d = L*l, то d · a = L*l · a = L*l, т.е. a эквивалентно ε;

если d = L*, то d · a = L* · a = L*l, т.е. a эквивалентно σ.

Таким образом для эмуляции поведения trylock, можно принять что A(trylock) = σ. Мы говорим об эмуляции, т.к. A(trylock) = σ означает, что все трассы проходящие через функцию trylock заканчиваются блокировкой строго внутри этой функции, что некорректно.

5. Реализация

Описанный выше алгоритм был реализован в системе статического анализа Svace, разрабатываемой в ИСП РАН.

Svace осуществляет анализ алиасов и сопоставление формальных и фактических параметров при вызове функции. Благодаря этому можно разрабатывать алгоритм поиска повторных блокировок для программы, содержащей только один семафор, а вся остальная работа будет выполнена средой Svace.

Система Svace выполняет неконсервативный анализ, допуская ложные срабатывания. Поэтому алгоритм поиска повторных блокировок реализованный в Svace может выдавать ложные срабатывания. Подробнее система Svace описана в [3], [4], [5].

Для тестирования алгоритма был создан набор тестов, моделирующий различные варианты использования блокировок. Для тестов алгоритм выдал предупреждения о повторной блокировке для всех ожидаемых ситуаций. Пример теста приведён на рис. 3, ожидается что предупреждение о повторной блокировке будет выдано для строки 12.

```
 1: void mutex_unlock(MUTEX*lock) {
 2:    unlock(lock);
 3:}
 4:void lock_section(MUTEX *lock) {
 5:    mutex_unlock(lock);
 6:    lock(lock); //svace: not_emitted DOUBLE_LOCK
 7:}
 8:void test(MUTEX* mut) {
 9:    lock(mut);
10:    if(flag) {
11:       lock_section(mut);
12:       lock(mut); //svace: emitted DOUBLE_LOCK
```

```
13:  }
14:}
```

Рис. 3. Пример теста для проверки алгоритма.

Также был произведён анализ 33 проектов с открытым исходным кодом (включая ядро операционной системы линукс (linux-kernel-3.10.9), binutils-2.20.1, libxml2-2.7.6, nss-3.12.9+ckbi-1.82, pulseaudio-0.9.22). Было выдано одно сообщение о двойной блокировке для pulseaudio-0.9.22, фрагмент кода приведён на рис. 4. Процедура pa_mutex_lock блокирует семафор e->mutex на строке 1068. После чего в цикле вызывается процедура pa_memexport_process_release на строке 1070, которая на строке 1087 вызывает процедуру pa_mutex_lock и повторно блокирует семафор. В данном случае сообщение является ложным, т. к. семафор был создан с флагом PTHREAD_MUTEX_RECURSIVE, позволяющим повторную блокировку. Анализатор выдал сообщение, т. к. не учитывает способ создания семафора. Таким образом с точки зрения анализатора предупреждение является истинным.

```
1065: void pa_memexport_free(pa_memexport *e) {
1066:    pa_assert(e);
1067:
1068:   pa_mutex_lock(e->mutex);
1069:   while (e->used_slots)
1070:       pa_memexport_process_release(e, (uint32_t) (e->used_slots - e->slots));
1071:   pa_mutex_unlock(e->mutex);
1072:
1073:   pa_mutex_lock(e->pool->mutex);
1074:   PA_LLIST_REMOVE(pa_memexport, e->pool->exports, e);
1075:   pa_mutex_unlock(e->pool->mutex);
1076:
1077:   pa_mutex_free(e->mutex);
1078:   pa_xfree(e);
}

1082: int pa_memexport_process_release(pa_memexport *e, uint32_t id) {
1083:   pa_memblock *b;послать
1084:
1085:   pa_assert(e);
```

```
1086:
1087:   pa_mutex_lock(e->mutex);
1088:
1089:   if(id >= e->n_init)
1090:       goto fail;
...
```

Рис. 4. Фрагмент кода для выданного сообщения.

6. Заключение

Был разработан и реализован алгоритм поиска двойных блокировок семафоров. Алгоритм разработан таким образом, чтобы выдавать как можно меньше ложных срабатываний. Для реализации использовалась система статического анализа Svace. Результаты тестирования показали, что алгоритм может быть полезен, но выдаётся слишком мало срабатываний. На наш взгляд причина заключается в большом количестве взаимосвязанных условных выражений, которые используются вместе с семафорами. Текущая реализация не выдаёт предупреждения для условных блокировок семафоров. Поэтому в планах по улучшению анализа – добавление в систему Svace разрешителя булевых формул, после чего алгоритм сможет выдавать предупреждения для более сложных ситуаций. При этом не потребуется менять сам алгоритм поиска двойных блокировок, а только инфраструктуру Svace.

Список литературы

[1]. POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0464 [121]

[2]. Cousot, P., & Cousot, R. (1992). Abstract interpretation and application to logic programs. The Journal of Logic Programming, 13(2), 103-179.

[3]. В.С. Несов. Автоматическое обнаружение дефектов при помощи межпроцедурного статического анализа исходного кода. Материалы XI Международной конференции «РусКрипто'2009».

[4]. А. Аветисян, А. Белеванцев, А. Бородин, В. Несов. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ. Труды ИСП РАН, 2011, том 21.

[5]. Иванников, В. П., Белеванцев, А. А., Бородин, А. Е., Игнатьев, В. Н., Журихин, Д. М., Аветисян А.И., Леонов, М. И. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды Института системного программирования РАН, 2014, 26(1), с. 231-250, doi: 10.15514/ISPRAS-2014-26(1)-7.

Static detection of error of double locking of mutex

Alexey Borodin
<alexey.borodin@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. This paper describes algorithm for static search for error of double locking of mutex. The algorithm allows emitting warnings with low level of false positives. We considered finding errors for abstract library containing functions of mutex lock, unlock and conditional locking. We defined a set of regular languages, which models locks and unlocks during concrete execution of a program. We have defined a domain approximated set of regular languages. The algorithm is implemented in terms of data flow analysis. In the analysis elements of the domain are used as the data flow properties. The algorithm is described for a program that has only one mutex and does not contain any aliases. In that case every emitted warning will correspond to a real error which may occur during program execution. The algorithm is implemented in system of static analysis Svace developed in Institute for System Programming of the Russian Academy of Sciences. Svace performs alias analysis and matching of formal and actual function parameters. This makes it possible to apply the algorithm to search for double locking of a program containing only one mutex, and the rest of the work will be executed by Svace. The search algorithm of the double lock implemented in Svace can emit some number of false positives because Svace performs nonconservative analysis.

Keywords: static analysis; mutex; error of double locking; data-flow analysis; regular languages.

References

[1]. POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0464 [121]
[2]. Cousot, P., & Cousot, R. (1992). Abstract interpretation and application to logic programs. The Journal of Logic Programming, 13(2), 103-179.
[3]. Nesov V. S. Automatically Finding Bugs in Open Source Programs. Proceedings of the Third International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2009).
[4]. Avetisyan A.I., Belevantsev A.A., Borodin A.E., Nesov V. S. Ispol'zovanie staticheskogo analiza dlya poiska uyazvimostej i kriticheskikh oshibok v iskhodnom kode programm [Using static analysis to find vulnerabilities and critical errors in the source code of programs]. Trudy ISP RAN [The Proceedings of ISP RAS], 2011, vol 21 (in Russian).
[5]. Ivannikov V.P., Belevantsev A.A., Borodin A.E., Ignatyev V.N., Zhurikhin D.M., Avetisyan A.I., Leonov M.I. Staticheskij analizator Svace dlya poiska defektov v iskhodnom kode programm [Svace: static analyzer for detecting of defects in program source code]. Trudy ISP RAN [The Proceedings of ISP RAS], 2014, vol 26(1), pp. 231-250 (in Russian), doi: 10.15514/ISPRAS-2014-26(1)-7.

Применение компиляторных преобразований для противодействия эксплуатации уязвимостей программного обеспечения*

А.Р.Нурмухаметов
<oleshka@ispras.ru>
Ш.Ф. Курмангалеев
<kursh@ispras.ru>.
В.В. Каушан
<korpse@ispras.ru>
С.С. Гайсарян
<ssg@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25.

Аннотация. Уязвимости программного обеспечения представляют серьезную угрозу безопасности информационной системы. Любое программное обеспечение, написанное на языках C/C++, потенциально содержит в себе значительное количество уязвимостей, используя которые злоумышленник может с помощью специально подготовленных эксплойтов захватить контроль над системой. Для противодействия эксплуатации таких уязвимостей в данной работе предлагается использовать компиляторные преобразования: перестановка местами функций в модуле, добавление локальных переменных на стек функции, перемешивание локальных переменных на стеке. С помощью этих преобразований предлагается генерировать диверсифицированную популяцию исполняемых файлов компилируемого приложения. Такой подход, например, усложняет планирование ROP-атак на всю популяцию. Злоумышленник, получив в свое распоряжение один исполняемый файл, может сделать ROP-эксплойт, работающий только для этой версии приложения. Остальные исполняемые файлы популяции останутся устойчивыми к данной атаке.

Ключевые слова: уязвимость; компиляторные преобразования; ROP-атака; эксплойт.

1. Введение

Программное обеспечение, написанное на языках C/C++, потенциально содержит в себе значительное количество уязвимостей. Уязвимости могут

* Работа поддержана грантом РФФИ 14-01-00462 А

быть как результатом ошибок программирования и недостатков, допущенных при проектировании, так и результатом внесения специальных закладок в код открытых проектов. Оценить количество ошибок непросто из-за разнообразности исходного кода каждого проекта и неясности точного определения того, что является ошибкой, а что не является. Разные источники дают разные количественные данные, например, в работе [1] авторы говорят об одной ошибке на каждую тысячу строк кода. Не столько важно конкретное количество ошибок, сколько их наличие. Даже одна единственная ошибка, затерявшаяся от глаз разработчиков в миллионах строках исходного кода большого проекта, может быть с успехом использована злоумышленником для перехвата управления системы с последующим исполнением вредоносного кода.

Поиск и устранение ошибок, приводящих к уязвимостям, в исходном коде трудоемкая и дорогостоящая задача. Для ее решения существуют такие продукты, как Coverity Insight, Klocwork K9, Svace [2]. Ни один из них не в состоянии найти в большом проекте все уязвимости. В условиях наличия в программном обеспечении уязвимостей актуальной становится задача воспрепятствования их эксплуатации. Для ее решения существует ряд технологий: предотвращение выполнения данных (DEP) [3], рандомизация адресного пространства (ASLR) [4], протектор стека [5], безопасные функции [6], теневой стек [7, 8, 9], исполняемые файлы без инструкций возврата [10, 11]. Суть каждого метода подробнее будет описана в следующем разделе, кроме того, будут проанализированы их сильные и слабые стороны.

Ошибки программирования или специальные закладки, приводящие к уязвимостям программного обеспечения, особо опасны по той причине, что огромное количество компьютеров одновременно обладают идентичным программным обеспечением и могут быть массово атакованы одним эксплойтом. На данный момент сложилась ситуация программного однообразия. Идентичные исполняемые файлы самого популярного программного обеспечения работают на миллионах компьютеров. Это облегчает масштабное эксплуатирование, потому что одна и та же атака с успехом проходит на множество целей.

Задачу противодействия эксплуатации ошибок в программном обеспечении можно разделить на две подзадачи. Первая задача – противодействие современным средствам статического и динамического анализа, чтобы затруднить поиск ошибки и построение эксплойта. Для решения этой задачи требуется использовать запутывающие преобразования, которые также могут быть реализованы в запутывающем компиляторе [12]. Вторая задача – затруднение эксплуатации имеющихся уязвимостей.

В данной работе предлагается путем использования компиляторных преобразований (перестановка функций, добавление локальных переменных на стек, перемешивание переменных на стеке [13, 14]) генерировать

диверсифицированную популяцию исполняемых файлов компилируемого приложения. Злоумышленник, получив одну из версий исполняемого файла, может создать для него эксплойт, который не будет работать для других версий исполняемого файла в силу их различий. Подробно предлагаемый подход описывается в разделе 3.

Настоящая статья состоит из введения, пяти разделов и заключения. В разделе 2 описываются существующие компиляторные метод защиты. В разделе 3 подробно описывается предлагаемый подход. В разделе 4 описываются реализованные компиляторные преобразования и технические подробности, связанные с ними. Влияние на производительность наглядно представлено в разделе 5. В разделе 6 приводятся оценки защищенности предлагаемой методики защиты.

2. Обзор методов защиты

2.1 Предотвращение выполнения данных

Предотвращение выполнения данных (англ. Data Execution Prevention, DEP) — функция безопасности, которая не позволяет выполнять код из областей памяти, помеченных «только для данных». Присутствует во всех современных операционных системах и опирается на аппаратные возможности современных процессоров (NX-бит). Данная технология позволяет предотвратить атаки, которые сохраняют код для последующего исполнения в такой области. Примером таких атак являются атаки переполнения буфера. Классическая атака, использующая переполнение буфера, хранила вредоносный код в самом буфере на стеке, и управление передавалось на начало буфера. Если секция стека помечена флагом «только для данных», то вредоносный код не выполнится, произойдет обработка исключения. Таким образом, данная технология не устраняет саму уязвимость переполнения буфера, а всего лишь запрещает выполнение кода, например, на стеке. Буфер переполнить, возможно, но вредоносный код необходимо положить в другое место, выполнение кода в котором разрешено. Другой способ обхода такого способа защиты – ROP-атака, которая использует код, уже находящийся в области памяти с возможностью выполнения.

2.2 Рандомизация адресного пространства

Рандомизация адресного пространства (англ. Address Space Layout Randomization, ASLR) – технология, применяемая в операционных системах, при использовании которой случайным образом изменяется расположение в адресном пространстве процесса важных структур, а именно: образа исполняемого файла, подгружаемых библиотек, кучи и стека. Технология значительно усложняет эксплуатацию уязвимостей, приводящих к перехвату потока управления, так как довольно сложно угадать, где будет расположен

стек или куча, в которых можно поместить вредоносный код. Данная технология поддерживается во всех современных операционных системах.

2.3 Протектор стека

Вставка протектора стека (англ. Stack canary) – компиляторный метод защиты от уязвимостей переполнения буфера на стеке. Данный метод позволяет эффективно противостоять атакам переполнения буфера с незначительным падением производительности защищаемой программы. Он реализуется как патч к компилятору, изменяющий пролог и эпилог функции. В прологе функции на стек кладется специальное проверочное значение. Это значение располагается на стеке между локальными переменными и адресом возврата функции, исполняя роль стража адреса возврата. В эпилоге функции происходит проверка проверочного значения на изменение. Если произошло переполнение буфера, то проверочное значение изменится, и проверка приведёт к вызову обработчика исключительной ситуации.

2.4 Безопасные функции

Безопасные функции – это легковесная компиляторная защита от переполнения буфера для некоторых функций работы со строками и памятью. Для этих функций подсчитывается, если это возможно во время компиляции, размер буфера, в который будет производиться запись. Вызовы, для которых размер копируемого буфера известен во время компиляции, проверяются на переполнение непосредственно во время компиляции. Для других вызовов генерируется вызов специальной функции-оболочки, в которой во время исполнения производится проверка на переполнение. В случае если функция не прошла проверку на переполнение во время компиляции, то выбрасывается ошибка компиляции, если функция во время исполнения не пройдет проверку, то вызовется обработчик исключительной ситуации и исполнение прервётся. Использование безопасных функций позволяет контролировать выход за пределы буферов только в том случае, когда компилятор знает их размер. Данная защита не сможет воспрепятствовать переполнению буфера, размер которого неизвестен во время компиляции.

2.5 Теневой стек

В основе этого подхода лежит идея о создании для адресов возврата отдельного стека, который нельзя будет изменить, переполняя какой угодно буфер на стеке с данными. Существуют различные версии реализации теневого стека: аппаратного уровня (SmashGuard) [9], компиляторного уровня (StackShield) [8], уровня исполняемого файла (TRUSS) [7].

На аппаратном уровне семантика инструкций вызова и возврата изменяется таким образом, чтобы при вызове функции копия адреса возврата сохранялась в теневом стеке, а при возврате из функции сравнивалась с текущим адресом

возврата. Если адреса различаются, то происходит прерывание исполнения и запускается обработчик исключительной ситуации. Такой подход не требует никакой модификации программного обеспечения.

Аналогичная логика сохранения копии адреса возврата в теневом стеке используется в защите компиляторного уровня (StackShield). Во время компиляции вставляются инструкции, сохраняющие копию адреса возврата в теневой стек при вызове функции и сравнивающие его с текущим адресом возврата при возврате. Обработчик исключительного события вызывается в том случае, если адреса различаются. Для защиты приложения данным методом нужно иметь доступ к его исходным кодам.

TRUSS использует бинарную инструментацию для проверки адреса возврата до его использования. Каждая инструкция вызова функции инструментируется кодом, который сохраняет копию адреса возврата в теневой стек, а каждая инструкция возврата из функции инструментируется кодом сравнения адресов. В случае их различия вызывается обработчик исключительной ситуации. Преимуществом данного подхода является то, что таким образом можно защищать программы, для которых не предоставлено исходного кода.

2.6 Исполняемые файлы без инструкций возврата

Данный метод защиты направлен против ROP-атак и предлагает исключить абсолютно все инструкции возврата из бинарного кода программы. Машинные коды инструкций возврата могут содержаться не только в самих инструкциях возврата, но и в других инструкциях и их операндах. Для полного их удаления в работе [10] предлагается использовать компиляторные преобразования: перераспределение регистров для удаления машинных кодов возврата из машинных кодов регистровых операндов; оптимизации замены инструкций, содержащих внутри себя или внутри своих нерегистровых операндов машинные коды возврата, на эквивалентные инструкции; методику непрямого возврата для удаления из кода непосредственно инструкций возврата.

Методика непрямого возврата состоит в следующем: при каждом вызове функции вместо адреса возврата на стек записывается некоторый индекс. Этот индекс будет указывать на адрес возврата в некоторой таблице адресов возврата, которая специально выделена в отдельном месте в памяти. При каждом возврате управления из функции адрес возврата находится в таблице адресов возврата по индексу, сохраненному на стеке.

3. Описание предлагаемого подхода

Компилятор является ключевой частью процесса разработки программного обеспечения, в том числе потому, что он транслирует программу, написанную на языке высоко уровня, в низкоуровневую программу, непосредственно исполняемую компьютером. Одним из свойств любого компилятора является

детерминированность процесса трансляции. Это означает, что один и тот же файл исходного кода всегда транслируется в один и тот же исполняемый файл, конечно при условии того, что компиляция производится при одинаковом уровне оптимизации.

Компилятор, оборудованный набором диверсифицирующих преобразований, может генерировать большое количество функционально эквивалентных, но внутренне различных копий компилируемой программы. Поведение программы, специфицированное исходным кодом, у всех копий одинаковое. Разным является поведение в тех случаях, которые не специфицированы напрямую в исходном коде программы, стандартом языка или соглашениями о вызовах и общих интерфейсах. Таким образом, каждая версия программы ведет себя по-разному при попытке эксплуатировать ее неспецифицированное поведение (т.е. уязвимость).

Генерация диверсифицированной популяции исполняемых файлов для одного пакета исходного кода приводит к усложнению и удорожанию разработки эксплойтов. Разработчик, планирующий атаку, должен либо создать эксплойт для каждой копии, либо сделать его достаточно сложным для того, чтобы он мог успешно срабатывать сразу на нескольких копиях. В любом случае ему потребуется больше времени и ресурсов для планирования атаки и разработки эксплойта.

В современном мире большое количество программного обеспечения распространяется через магазины приложений различных платформ (Apple's App Store, Android Marketplace, Windows Market). В такой схеме распространения без труда можно осуществить автоматическую генерацию уникальной копии бинарного кода приложения по запросу пользователя. Злоумышленник, загрузив из магазина приложения одну из копии бинарного пакета, создаст для своей версии приложения эксплойт. При попытке атаки другие копии данного приложения не будут скомпрометированы этим эксплойтом (рис. 1).

Рис. 1. Схема атаки на диверсифицированную популяцию исполняемых файлов.

Для генерации разнообразных программ компилятор должен иметь внутри себя источник недетерминированности. Предлагается использовать линейный

конгруэнтный генератор случайных чисел, в котором последовательность псевдослучайных чисел генерируется рекуррентной формулой:

$$r_n = (A * r_c + C)\% M$$

где A, C, M – некоторые специальным образом подобранные константы, r_c – случайное число на текущей итерации генератора случайных чисел, r_n – случайное число на следующей итерации генератора случайных чисел.

Последовательность генерируемых псевдослучайных чисел однозначно задается по начальной затравке, которая задается флагом компилятора. Таким образом, реализуется воспроизводимая недетерминированность процесса компиляции, которая необходима, например, для отладки программ.

4. Реализованные преобразования

Для диверсификации генерируемых программ были реализованы следующие преобразования: перестановка функции, добавление и перемешивание переменных на стеке. Данные преобразования были реализованы в компиляторе GCC и разрабатываемом в ИСП РАН обфусцирующем компиляторе на базе LLVM [12]. В качестве основной реализации рассматривается GCC, поскольку обладает большей совместимостью (например, собирает ядро Linux). Использование GCC позволяет при необходимости генерировать диверсифицированные образы операционной системы целиком.

4.1 Перестановка функций

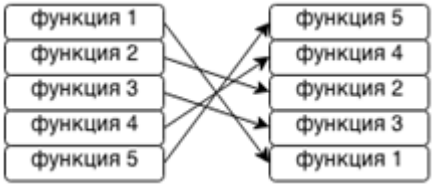

Рис. 2.Перестановка функций местами

Перестановка функций местами меняет структуру памяти процесса. Изменяется относительный порядок функций, меняются адреса точек входа. Данное преобразование особо полезно для противодействия ROP-атакам. При планировании таких атак важно знать местоположения гаджетов. Гаджеты – это небольшие участки машинного кода, оканчивающиеся инструкцией возврата. Правильно передавая управление с одного такого участка кода на другой, можно выполнить вредоносный код. Разработчик эксплойта, подобрав нужную ему последовательность гаджетов для доступной ему копии приложения, столкнется с той проблемой, что адреса гаджетов в другой копии приложения будут другими.

Скажем пару слов о реализации перестановки функций местами (рис. 2). Каждой функции в модуле сопоставляется случайное число, полученное от линейного конгруэнтного генератора случайных чисел. Функции переставляются в соответствии с порядком возрастания, присвоенных им случайных чисел. Количество уникальных перестановок функций местами, следовательно, и количество уникальных копий приложения, равно $n!$.

4.2 Добавление локальных переменных

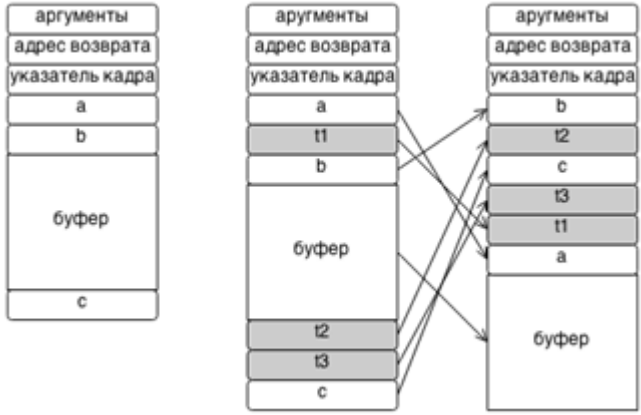

Рис. 3. Добавление локальных переменных и их перемешивание на стеке

Для добавления локальных переменных на стек функции был реализован компиляторный проход над промежуточным представлением GIMPLE. Этот проход добавляет в каждую функцию заданное флагом количество локальных переменных, хранящихся на стеке (рис. 3).

4.3 Перемешивание локальных переменных

Перемешивание переменных на кадре стеке функции (рис. 3) реализовано аналогично перестановке функций местами. Каждой локальной переменной, находящейся на стеке функции, сопоставляется случайное число, полученное от линейного конгруэнтного генератора случайных чисел. Локальные переменные перемешиваются в соответствии с порядком возрастания присвоенных им случайных чисел. Количество уникальных перестановок кадра стека функции равно:

$$(n_f + k)!$$

где n_f – количество локальных переменных на стеке функции, k - количество добавленных локальных переменных. Если рассмотреть количество уникальных перестановок стека глубиной m, то оно задается формулой:

$$\prod_{i=1}^{m}(n_i + k)!$$

где n_i – количество локальных переменных на стеке в функции на глубине вызова i.

5. Влияние на производительность

Тестирование производительности производилось на приложении SQLite. Результаты наглядно представлены в табл. 1. Всего было проведено шесть серий измерений. Первая из них с названием base проводилась при выключенных диверсифицирующих преобразованиях. Серия Peak1 – с включенной перестановкой функций. Серия Peak2 – с включенными перестановками функций и локальных переменных. Серии Peak3, Peak4, Peak5 – с обеими включенными перестановками и количеством добавленных переменных 1, 3, 5 соответственно.

Проведенные диверсифицирующие преобразования отрицательно сказываются на производительности и увеличивают размер кода. Чем больше вставляется локальных переменных на стек, тем больше падает производительность и тем больше становится размер исполняемого файла программы.

Табл. 1. Влияние на производительность реализованных преобразований

серия	Количество добавленных лок. перем.	Перестановка		Увеличение размера, %	Ухудшение производ., %
		лок. перем.	функций		
Peak5	5	+	+	5.16	15.75
Peak4	3	+	+	3.13	7.87
Peak3	1	+	+	1.29	3.94
Peak2	0	+	+	0	1.57
Peak1	0	-	+	0	0.79
Base	0	-	-	0	0

6. Оценка защищенности

Оценка защищенности производилась в режиме полностью контролируемого окружения. Режим полностью контролируемого окружения подразумевает под собой выключенный в системе ASLR и отключенный в исполняемом файле DEP. Для заданного образца защищённого приложения возможна полуавтоматическая генерация эксплойта, приводящего к перехвату управления. Для другой копии, скомпилированной с неизвестным значением затравки для генератора случайных чисел, возможно изменение эксплойта. Для этого нужно подобрать константу, характеризующую масштабы изменений между кадрами стека эксплуатируемых функций. Эта константа

линейно зависит от размера кадра стека незащищённого приложения и количества добавленных локальных переменных. Константу подбирается простым перебором, при этом на каждой итерации перебора требуется запуск целевого приложения. Перебор можно значительно упростить, заполняя свободное пространство эксплойта значением для адреса возврата. В этом случае требуется около десятка попыток. В случае, если на стеке находятся переменные, изменение которых может привести к преждевременному аварийному завершению из-за их использования, то перемешивание локальных переменных может привести как к усилению защиты, так и к её ослаблению.

В случае ROP-атаки возможно построение цепочки ROP-гаджетов по коду копии приложения, имеющегося в наличии. Однако построение ROP-эксплойта для другой копии с неизвестным значением затравки для генератора случайных чисел является довольно сложной задачей, потому что требуется знание адресов ROP-гаджетов для этой копии. Подбор этих адресов потребует неприемлемое количество времени.

Необходимо отметить возможность атаки на затравку для генератора случайных чисел. Предсказуемость этого значения может значительно упростить задачу перебора адресов ROP-гаджетов. Поэтому следует генерировать достаточно случайные значения для затравки генератора, чтобы их было сложно предсказать или подобрать.

Для обеспечения максимального уровня защиты рекомендуется использовать предлагаемый подход совместно с ASLR и DEP. Указанные технологии затрудняют прямую эксплуатацию уязвимостей, запрещая исполнять код на стеке и обеспечивая случайное перераспределение адресов загрузки исполняемых модулей. Тем не менее, сочетание ASLR и DEP зачастую недостаточно, поскольку в системе присутствуют файлы с выключенным ASLR, анализируя которые, можно создать набор ROP-гаджетов (или использовать для этого модуль атакуемого приложения). Применение диверсификации решает данную проблему, поскольку найденные адреса ROP-гаджетов, смещения функций, системных структур оказываются непригодны для построения эксплойта на нескомпроментированных системах.

7. Заключение

В представленной работе предложены и описаны диверсифицирующие преобразования, реализованные в рамках обфусцирующего компилятора на базе GCC. За счет этих преобразований в условиях современной модели распространения программного обеспечения через магазины приложений становится возможным предоставить каждому клиенту свою оригинальную версию бинарного приложения. Внутренняя структура приложений у различных клиентов разная, что повышает уровень защиты, а в случае некоторых типов атак (ROP-атак) противодействует эксплуатации уязвимостей. Реализованные преобразования отрицательно сказываются на

производительности приложений. По результатам тестов было зафиксировано ухудшение производительности на 15% и увеличение размера кода на 5% на тесте SQLite. С применением диверсифицирующих преобразований производилась сборка полного образа системы CentOS, включая ядро Linux и большое количество пакетов приложений. Это говорит о возможности генерации диверсифицированных образов операционной системы целиком.

В дальнейшие планы развития обфусцирующего компилятора входит как увеличение количества диверсифицирующих преобразований, так и оснащение его другими методами защиты от эксплуатации уязвимостей разными типами атак. Теневой стек выглядит одним из самых перспективных методов защиты от перехвата управления.

Список литературы

[1]. Dazhi Z., Detecting Program Vulnerabilities Using Trace-Based Security Testing, Ph. D. Dissertation, University of Texas at Arlington, Arlington, TX, USA, Advisor(s) Donggang L, AAI3474008, 2011.

[2]. А. Аветисян, А. Белеванцев, А. Бородин, В. Несов. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ. Труды Института Системного Программирования РАН, том 21, 2011 г, стр. 23-38.

[3]. N. Stojanovski, M. Gusev, D. Gligoroski, S. Knapskog. Bypassing Data Execution Prevention on MicrosoftWindows XP SP2. Proceedings of the The Second International Conference on Availability, Reliability and Security, ARES '07, 2007, p. 1222-1226. doi:10.1109/ARES.2007.54

[4]. H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, D. Boneh. On the Effectiveness of Address-space Randomization. Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04, 2004, p. 298-307. doi:10.1145/1030083.1030124

[5]. P. Wagle, C. Cowan. Stackguard: Simple stack smash protection for GCC. Proc. of the GCC Developers Summit, 2003, p. 243-255.

[6]. J. Jelinek. Object size checking to prevent (some) buffer overflows, 2004 https://gcc.gnu.org/ml/gcc-patches/2004-09/msg02055.html

[7]. S. Sinnadurai, Q. Zhao, W. Wong. Transparent runtime shadow stack: Protection against malicious return address modifications, 2008.

[8]. StackShield: A “stack smashing” technique protection tool for Linux. (http:/www.angelfire.com/sk/stackshield)

[9]. H. Ozdoganoglu, T. N. Vijaykumar, C. E. Brodley, A. Jalote, B. A. Kuperman. “SmashGuard: A Hardware Solution to Prevent Security Attacks on the Function Return Address.” Technical Report TR-ECE 03-13, Purdue University, February 2004.

[10]. K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti, E. Kirda. G-Free: Defeating Return-oriented Programming Through Gadget-less Binaries. Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC '10, 2010, p. 49-58. doi:10.1145/1920261.1920269

[11]. J. Li, Z. Wang, X. Jiang, M. Grace, S. Bahram. Defeating Return-oriented Rootkits with "Return-Less" Kernels. Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, 2010, p. 195-208. doi:10.1145/1755913.1755934

[12]. В.Иванников, Ш. Курмангалеев, А. Белеванцев, А. Нурмухаметов, В. Савченко, Р. Матевосян, А. Аветисян. Реализация запутывающих преобразований в компиляторной инфраструктуре LLVM. руды Института Системного Программирования РАН, том 26, 2014 г, выпуск 1 стр. 327-342. doi: 10.15514/ISPRAS-2014-26(1)-12

[13]. M Stewart. Algorithmic Diversity for Software Security. (http://arxiv.org/abs/1312.3891)

[14]. M. Franz. E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism. In Proceedings of the 2010 Workshop on New Security Paradigms, NSPW ’10, p. 7–16. doi:10.1145/1900546.1900550

Compiler protection techniques against software vulnerabilities exploitation*

A. Nurmukhametov
<oleshka@ispras.ru>
Sh. Kurmangaleev
<kursh@ispras.ru>
V. Kaushan
<korpse@ispras.ru>
S. Gaissaryan
<ssg@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. Software vulnerabilities are critical for security. All C/C++ programs contain significant amount of vulnerabilities. Some of them can be successfully exploitable by attacker to gain control of the execution flow. In this article we propose several compiler protection techniques against vulnerability exploitation: function reordering, insertion of additional dummy variables into stack, local variables permutation on the stack. These transformations were implemented in GCC. It successfully diversifies whole operational system including Linux kernel. We suggest to generate diversified population of binary application files with these transformations. Diversified applications can be easily distributed via the application stores. Every client downloads the unique copy of application. The proposed method complicates and increases the cost of ROP-attacks. After downloading of the binary copy attacker can create ROP-exploit for this copy but it would not be exploitable for another application copy. The diversified transformations decrease application performance about 15% and increase code size about 5%.

Keywords: vulnerability; compiler transformations; ROP-attacks; exploit.

References

[1]. Dazhi Z., Detecting Program Vulnerabilities Using Trace-Based Security Testing, Ph. D. Dissertation, University of Texas at Arlington, Arlington, TX, USA, Advisor(s) Donggang L, AAI3474008, 2011.

[2]. A. Avetisyan, A. Belevantsev, A. Borodin, V. Nesov. Ispol'zovanie staticheskogo analiza dlya poiska uyazvimostej i kriticheskikh oshibok v iskhodnom kode program [The usage of static analysis for searching vulnerabilities and critical errors in source code]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 21, 2011. p. 23-38.

[3]. N. Stojanovski, M. Gusev, D. Gligoroski, S. Knapskog. Bypassing Data Execution Prevention on MicrosoftWindows XP SP2. Proceedings of the The Second International

* The paper is supported by RFBR grant 14-01-00462 A

Conference on Availability, Reliability and Security, ARES '07, 2007, p. 1222-1226. doi:10.1109/ARES.2007.54

[4]. H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, D. Boneh. On the Effectiveness of Address-space Randomization. Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04, 2004, p. 298-307. doi:10.1145/1030083.1030124

[5]. P. Wagle, C. Cowan. Stackguard: Simple stack smash protection for GCC. Proc. of the GCC Developers Summit, 2003, p. 243-255.

[6]. J. Jelinek. Object size checking to prevent (some) buffer overflows, 2004 https://gcc.gnu.org/ml/gcc-patches/2004-09/msg02055.html

[7]. S. Sinnadurai, Q. Zhao, W. Wong. Transparent runtime shadow stack: Protection against malicious return address modifications, 2008.

[8]. StackShield: A "stack smashing" technique protection tool for Linux. (http:/www.angelfire.com/sk/stackshield)

[9]. H. Ozdoganoglu, T. N. Vijaykumar, C. E. Brodley, A. Jalote, B. A. Kuperman. "SmashGuard: A Hardware Solution to Prevent Security Attacks on the Function Return Address." Technical Report TR-ECE 03-13, Purdue University, February 2004.

[10]. K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti, E. Kirda. G-Free: Defeating Return-oriented Programming Through Gadget-less Binaries. Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC '10, 2010, p. 49-58. doi:10.1145/1920261.1920269

[11]. J. Li, Z. Wang, X. Jiang, M. Grace, S. Bahram. Defeating Return-oriented Rootkits with "Return-Less" Kernels. Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, 2010, p. 195-208. doi:10.1145/1755913.1755934

[12]. V.Ivannikov, SH. Kurmangaleev, A. Belevantsev, A. Nurmukhametov, V. Savchenko, R. Matevosyan, A. Avetisyan. Realizatsiya zaputyvayushhikh preobrazovanij v kompilyatornoj infrastrukture LLVM [Implementing Obfuscating Transformations in the LLVM Compiler Infrastructure]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 26, iss.1, 2014. p. 327-342. doi: 10.15514/ISPRAS-2014-26(1)-12

[13]. M Stewart. Algorithmic Diversity for Software Security. (http://arxiv.org/abs/1312.3891)

[14]. M. Franz. E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism. In Proceedings of the 2010 Workshop on New Security Paradigms, NSPW '10, p. 7–16. doi:10.1145/1900546.1900550

Автоматизированный метод построения эксплойтов для уязвимости переполнения буфера на стеке

В.А. Падарян
<vartan@ispras.ru>
В.В. Каушан
<korpse@ispras.ru>
А.Н. Федотов
<fedotoff@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. В статье рассматривается метод автоматизированного построения эксплойтов для уязвимости переполнения буфера на стеке и его применение к задаче оценки критичности ошибок в программном обеспечении. Метод, на основе динамического анализа и символьное выполнения, применяется к бинарным файлам программ без дополнительной отладочной информации. Описанный метод был реализован в виде инструмента для построения эксплойтов. Возможности инструмента были продемонстрированы на примере 8 уязвимостей в приложениях, работающих под управлением ОС Windows и Linux, 3 из которых не были исправлены на момент написания статьи.

Ключевые слова: классификация ошибок; эксплуатация уязвимостей; бинарный код; динамический анализ; символьное выполнение.

1. Введение

С развитием информационных технологий становятся все более значимыми как сама безопасность программ, так и средства ее обеспечивающие. Сложное программное обеспечение активно используется в работе критической инфраструктуры: контролирует движение транспорта, управляет в больницах различным медицинским оборудованием, регулирует работу электростанций. Рабочие сбои в функционировании такого ПО приводят к серьезным последствиям, а целенаправленная эксплуатация ошибок злоумышленниками способна привести к еще большему ущербу. Ошибки, использование которых может привести к намеренному нарушению целостности системы и нарушению её работы, называют уязвимостями. Многие крупные IT-компании (Microsoft, Google и т.д.) не только поддерживают исследования в области

поиска ошибок и уязвимостей ПО, но и внедряют самые передовые технологии в жизненный цикл разработки программ, с целью сокращения ошибок в выпускаемом ПО и снижения издержек, связанных с обнаружением и исправлением ошибок.

Искать ошибки и уязвимости можно как на уровне исходных текстов, так и анализируя бинарный код. Второй способ является более предпочтительным, так как абстракции языка высокого уровня скрывают особенности работы программы, важные для выявления ошибок и оценки их важности. Кроме того, исходные тексты зачастую недоступны, поэтому специалистам, оценивающим безопасность ПО, неизбежно приходится иметь дело с исполняемым (бинарным) кодом и применять соответствующие методы анализа [1]. В последние годы активно развивается подход к поиску ошибок на основе символьного выполнения.

Символьное выполнение было предложено в конце семидесятых годов двадцатого столетия для тестирования программного обеспечения [2]. Под символьным выполнением понимается процесс выполнения исследуемой программы, в которой конкретные значения переменных заменяются символьными значениями. Как правило, символьными значениями являются входные данные программы. Операции над символьными значениями порождают формулы, описывающие последовательность операций над символьными переменными и константами. Каждое условное ветвление, зависящее от символьных данных, добавляет в общую систему уравнение, описывающее прохождение управления по определенной ветке. Строящуюся систему уравнений называют предикатом пути, т.к. она описывает сценарий работы программы, в рамках которого была построена. Построенная система уравнений подается на вход программе-решателю, где символьные переменные выступают в качестве неизвестных величин в формулах. Результатом решения является конкретный набор значений для символьных переменных.

Первоначальная идея применения символьных вычислений была нацелена на улучшение покрытия программы тестами. Но в последнее время эту технику начали применять для направленного поиска определенного состояния программы. Перед вызовом решателя предикат пути дополняют уравнениями, описывающими искомое состояние программы, в рассматриваемом случае – это ситуация, когда срабатывает уязвимость. Ввиду большого количества классов уязвимостей и многочисленных факторов, влияющих на проявление той или иной уязвимости, попытки решить задачу формального описания уязвимостей на уровне бинарного кода предпринимались только для отдельных, частных случаев.

Уязвимость, как правило, обусловлена программной ошибкой. Однако не каждая программная ошибка порождает уязвимость. Современные средства фаззинга, применяемые в промышленном программировании, вырабатывают тысячи наборов входных данных, приводящих к аварийной остановке [3].

Актуален вопрос расстановки приоритетов для найденных ошибок. Целесообразно в первую очередь исправлять ошибки, которые могут быть проэксплуатированы. Наиболее опасными для пользователя и желаемыми для нарушителя являются ошибки, которые предоставляют возможность выполнить произвольный код.

В рамках данной работы предложен метод оценки найденных ошибок, основанный на символьном выполнении бинарного кода программ. Для заданного набора входных данных, приводящих к аварийной остановке программы, выполняется построение эксплойта (набора входных данных, эксплуатирующих уязвимость) для распространенного вида уязвимостей: переполнения буфера на стеке. Те ошибки, для которых удалось построить эксплойт, должны быть отнесены к категории критических и исправлены как можно скорее. Предложенный в работе метод автоматизируем; был разработан программный инструмент, реализующий этот метод. Он позволяет генерировать для соответствующих ошибок эксплойты, приводящие к выполнению кода полезной нагрузки, заданного пользователем.

Статья организована следующим образом. Во втором разделе рассмотрены методы, положенные в основу решения поставленной задачи. В третьем разделе представлены основные сведения об особенностях эксплуатации уязвимости переполнения буфера на стеке. В четвертом разделе описан сам метод, а отдельные аспекты его реализации и результаты практического применения приведены в пятом разделе. В последнем, шестом, разделе обсуждаются полученные результаты и дальнейшие направления исследований.

2. Используемые методы анализа

Анализ бинарного кода можно проводить с помощью методов статического и динамического анализа [1]. Символьное выполнение в рамках статического анализа ограничено применимо из-за высокой сложности получаемой системы уравнений, практический успех был заявлен только в рамках динамического [4] или смешанного анализа [5].

Представленные в данной статье исследования опираются на возможности среды анализа бинарного кода [6]. Основным предметом анализа являются трассы машинных инструкций, полученные в результате работы полносистемного эмулятора [7,8]. Трассы содержат состояния регистров, информацию о прерываниях и взаимодействии с периферийными устройствами, что позволяет восстанавливать статико-динамическое представление для всех работавших в системе программ и эффективно исследовать его свойства. Основным назначением среды анализа является автоматизация метода выделения алгоритмов из бинарного кода [9] и повышение уровня представления этих выделенных алгоритмов.

Поскольку известен набор входных данных, приводящий к аварийному завершению программы, можно получить трассу выполнения, которая

содержит аварийное завершение программы. Для генерации эксплойта достаточно рассматривать только те инструкции, которые относятся к обработке входных данных с момента их получения до момента аварийного завершения программы. Для отбора таких инструкций используется алгоритм динамического слайсинга трассы, дополненный анализом помеченных данных.

Современные процессорные архитектуры содержат множество различных инструкций со сложной семантикой и побочными эффектами. Распространенным приемом, обеспечивающим поддержку различных архитектур, является использование промежуточного представления. В рамках разрабатываемого метода используется промежуточное представление Pivot [10], позволяющее унифицировано описывать операционную семантику инструкций различных архитектур. Промежуточное представление отвечает требованием SSA-формы, что позволяет значительно упростить анализ. Основные операторы Pivot, используемые при построении предиката пути, следующие.

- Оператор `NOP` – не имеет никакого эффекта.
- Оператор `INIT` – инициализирует локальную переменную константным значением.
- Оператор `APPLY` – применяет одну из модельных операций. В качестве параметров и результата используются локальные переменные.
- Оператор `BRANCH` – осуществляет передачу управления.
- Оператор `LOAD` – производит загрузку значения из одного из адресных пространств.
- Оператор `STORE` – записывает значение в одно из адресных пространств.

Для описания памяти и регистров в Pivot используется модель единообразных адресных пространств. С точки зрения этой модели все адресуемые операнды целевой архитектуры, с которыми может работать машина (регистры, память, порты ввода-вывода), размещаются в линейных адресных пространствах, обращение к ним использует пару (идентификатор пространства, смещение). Для учета побочных эффектов работы операций используется модельное слово состояния, которое аналогично регистру флагов в x86.

3. Эксплуатация уязвимости переполнения буфера на стеке

Рассматривается ситуация, когда размер данных, записываемых в буфер на стеке, превышает размеры этого буфера. На рис. 1А показан кадр стека некоторой функции, в которой происходит переполнение буфера. Традиционно, в архитектуре x86 стек растет от старших адресов к младшим

(на рисунке – сверху вниз). Параметры размещаются на стеке в обратном порядке, далее вызов функции приводит к размещению адреса возврата на стеке, после чего функция может сохранить старое значение регистра `ebp` и выделить память для локальных переменных (в том числе для буфера). Данные в буфер записываются в порядке возрастания адресов (на рисунке – снизу вверх).

Если в буфер записать больше данных, чем размер этого буфера, это приведет к перезаписи ячеек выше по стеку, в том числе может привести к перезаписи старого значения `ebp`, адреса возврата и аргументов функции. Если злоумышленник может контролировать значения, записываемые в буфер, он может добиться того, что после переполнения адрес возврата будет содержать указатель на произвольный код, который может быть сформирован самим злоумышленником. Выполнение этого кода может привести к серьезным последствиям, в том числе к компрометации данных приложения, пользователя или операционной системы. Обычно, в качестве такого кода используется код, приводящий к вызову оболочки, называемый шелл-кодом. В реальности, использование такого кода настолько популярно, что часто любой код полезной нагрузки называют шелл-кодом. Злоумышленник может поместить код как ниже адреса возврата, так и выше него (рис. 1Б).

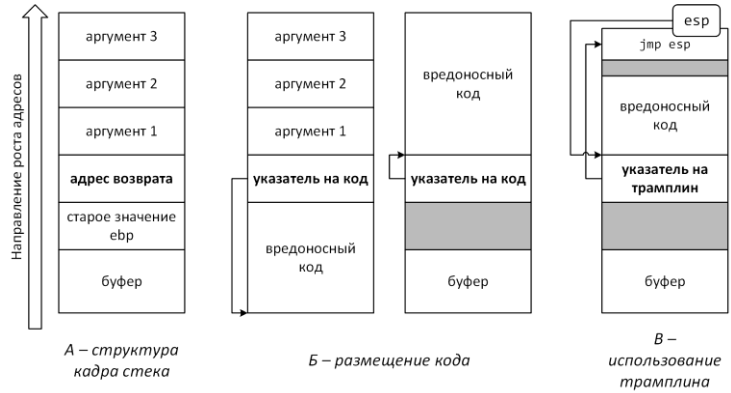

Рис. 1. Организация стека и способы размещения на нем внедряемого кода.

Следует отметить, что область памяти, выделенная под стек, может быть защищена от выполнения кода, и подобная попытка выполнения вредоносного кода может привести к аварийному завершению программы. В этом случае можно воспользоваться методикой Return-oriented programming [11], позволяющей составлять полезную нагрузку из уже имеющихся участков кода. Помимо того, при использовании механизма рандомизации адресного пространства, вредоносный код будет загружаться по разным адресам для разных попыток эксплуатации уязвимости, что не позволяет заранее

определить значение для записи на место адреса возврата. Но достаточно часто в момент выхода из функции один из регистров указывает на некоторую область памяти на стеке. Если эта область памяти доступна для размещения кода, то можно передать управление на этот код с помощью инструкции вида `jmp <reg>` или `call <reg>`, расположенной по заранее известному адресу. Инструкции такого вида в данной работе называются «трамплинами». При использовании трамплинов, после возврата из функции управление будет передано на трамплин, а оттуда – в размещенный на стеке исполняемый код (рис. 1В). Следует отметить, что использование трамплинов полезно не только в случае рандомизации, но и когда начальный адрес размещаемого шелл-кода содержит нулевой байт. Довольно часто переполнение буфера происходит во время копирования нуль-терминированной строки, которая не может содержать нулевые символы, а, следовательно, не позволяет переписать ячейку адреса возврата нужным значением.

4. Схема работы

Процесс генерации эксплойта разделен на четыре последовательных этапа, один из них является необязательным (рис. 2).

Сначала происходит выделение подтрассы, состоящей только из инструкций обрабатывающих входные данные. Для этого используется алгоритм слайсинга, а также информация о точке получения входных данных и точке аварийного завершения. Для полученной подтрассы строится предикат пути. В качестве опционального шага проводится поиск трамплинов. После этого происходит собственно построение эксплойта для заданного пользователем шелл-кода.

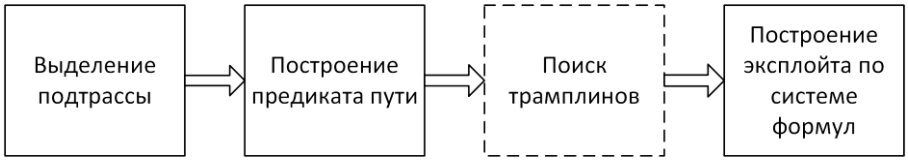

Рис. 2. Декомпозиция метода на четыре этапа

4.1 Выделение подтрассы

Выделение подтрассы делается для ограничения числа рассматриваемых машинных команд. В подтрассу отбираются только те команды, которые прямо или косвенно обрабатывают введенные в программу данные; ее построение выполняется алгоритмом слайсинга трассы [12]. Для работы слайсинга необходимы следующие данные: диапазон шагов трассы, на котором алгоритм отслеживает данные и начальный набор отслеживаемых данных. Начальному шагу соответствует точка получения входных данных, а конечному – точка аварийного завершения.

Начальный шаг и буфер с входными данными

Искать буфер с входными данными и шаг трассы, на котором буфер оказывается заполненным, аналитик может одним из нескольких способов.

Многие программы для получения данных используют библиотечные функции. Зная источник данных, например, сеть, файл и т.п., аналитик может найти в трассе вызовы соответствующих функций: например, для получения данных из сети чаще всего используется функция `recv`, а для чтения данных из файла – `ReadFile`. Входные данные также могут передаваться в качестве параметров командной строки, в таком случае, аналитику нужно найти вызов функции `main`.

Поиск шага аварийного завершения и ячейки перезаписи адреса возврата

Данная задача разделяется на две подзадачи:

- поиск ячейки, в которой хранится адрес возврата, перезаписываемый во время переполнения буфера;
- поиск шага трассы, на котором произойдет переход по перезаписанному адресу.

Для поиска ячейки используется алгоритм слайсинга, критерием является найденный буфер с входными данными. Для каждой отобранной инструкции адрес операнда-приемника сравнивается с адресами ячеек, хранящих адреса возврата функций для текущего стека вызовов. Если эти адреса пересекаются, это означает, что произошла перезапись адреса возврата одной из функций в стеке вызовов. В этом случае ячейка памяти, хранящая перезаписанный адрес возврата, считается искомой ячейкой.

Далее с помощью описанного ниже алгоритма выполняется поиск шагов трассы, в которых, вероятно, произошло аварийное завершение. Для каждого из этих шагов сравнивается значение регистра `ESP` с адресом ячейки, в которой хранится адрес возврата (полученной на предыдущем этапе). Если значения совпадают, то данный шаг считается шагом, на котором происходит аварийное завершение исследуемой программы.

Информация о возможных аварийных завершениях

Понятие аварийного завершения программы привязано к операционной системе. Так как предложенный метод рассчитан на работу в произвольном операционном окружении, без привязки к конкретной процессорной архитектуре, поиск аварийного завершения использует обобщенную модель процессора общего назначения и исходит только из найденных сценариев работы в рамках этой модели. Можно отметить, что аварийное завершение программы почти всегда происходит в результате исключительной ситуации, произошедшей в этой программе. В свою очередь, исключительная ситуация

приводит к возникновению прерывания процессора. Прерывания являются сущностями, не привязанными к конкретной платформе и архитектуре, поэтому работа алгоритма поиска аварийного завершения построена на анализе обработки прерываний. Необходимо различать прерывания, возникающие из-за исключительной ситуации в программе от прерываний, возникающих из-за работы устройств ввода/вывода или других событий процессора. Кроме того, так как трасса отражает выполнение всех программ в исследуемой системе, она содержит не только инструкции и исключительные ситуации, относящиеся к исследуемому приложению, но и для всех остальных приложений.

Используемый в данной работе алгоритм находит множество точек трассы, в которых, скорее всего, происходили исключительные ситуации. Так как аварийное завершение программы происходит в результате исключительной ситуации и при этом исключительная ситуация порождает прерывание, разумно искать точки аварийного завершения программы среди точек трассы, в которых возникают прерывания. Кроме того, исключительная ситуация нарушает естественный вход выполнения программы: вместо следующей инструкции после выхода из прерывания будет выполнен код обработчика исключений или завершение процесса. Это наблюдение позволяет значительно сократить множество точек трассы, рассматриваемых в дальнейшем.

Для каждого прерывания в трассе определяют адреса инструкций в точке входа в прерывание и в точке выхода из него. Далее рассматривается инструкция, выполняемая перед прерыванием. Здесь возможны два варианта: инструкции передачи управления и все остальные. Для инструкций передачи управления необходимо дополнительно вычислить адрес назначения: следующей должна выполниться инструкция по этому адресу (либо по адресу, следующему за инструкцией передачи управления в случае инструкций условного перехода). Для всех остальных инструкций сравнивается адрес после прерывания и адрес, следующий за выполненной инструкцией. Если они равны – выполнение не было нарушено. Аналогичные проверки проводятся для инструкций передачи управления, только сравнение идет с адресом назначения.

Результатом является множество точек трассы, в которых потенциально возникли исключительные ситуации. С помощью приведенного выше способа, среди этих точек перебором находится точка аварийного завершения исследуемой программы.

4.2 Построение предиката пути

Для построения предиката пути используется слайсинг трассы. Критерием слайсинга является уже найденный буфер с входными данными и шаг трассы, на котором происходит их получение. Просмотр трассы ограничен шагом, на котором происходит аварийное завершение исследуемой программы.

Результатом является слайс трассы и набор отслеживаемых ячеек на каждом шаге этого слайса. По этой информации строится предикат пути.

В данном случае, предикат пути представляет собой уравнения на языке SMT [13], описывающие преобразования, которые происходили с входными данными, на пути до определенной точки программы. Предикат пути необходим для того, чтобы описать ограничения на входные данные, полученные на пути к точке аварийного завершения.

В процессе построения предиката пути происходит двойная трансляция: сначала машинная инструкция транслируется в промежуточное представление, затем это представление переводится в соответствующие ему SMT-уравнения. Пространство памяти и регистров представляется в виде двух массивов битовых векторов. Так выглядит объявление массивов памяти и регистров на языке SMT:

```
(declare-const r_0 (Array (_ BitVec 16)  (_ BitVec 8)));
(declare-const v_0 (Array (_ BitVec 64)  (_ BitVec 8)));
```

Здесь размер адреса в пространстве регистров (r) составляет 16 бит, а в пространстве виртуальной памяти (v) – 64 бита. Гранулярность данных в обоих пространствах составляет 8 бит.

Процесс формирования уравнений предлагается рассмотреть на конкретном примере двух последовательных машинных команд `CMP BYTE PTR [80553450h], 00h` и `JB 806EE97Ch` архитектуры Intel x86. На рис. 3 представлен результат бинарной трансляции.

Трансляция Pivot-инструкций происходит последовательно, начиная с первой. Оператор `INIT` транслируется в константное выражение SMT. Оператор `APPLY` в эквивалентную операцию SMT-решателя. Например, для второго оператора `APPLY` инструкции `CMP` будет сгенерировано следующее выражение:

```
((_ sign_extend 64) #x80553450)
```

Оператор `Load` производит загрузку значения элемента адресного пространства (ячейка памяти, регистр) в локальную переменную. Необходимо определить, является ли элемент отслеживаемым или нет. Для этого происходит поиск элемента в наборе отслеживаемых элементов на момент отбора инструкции.

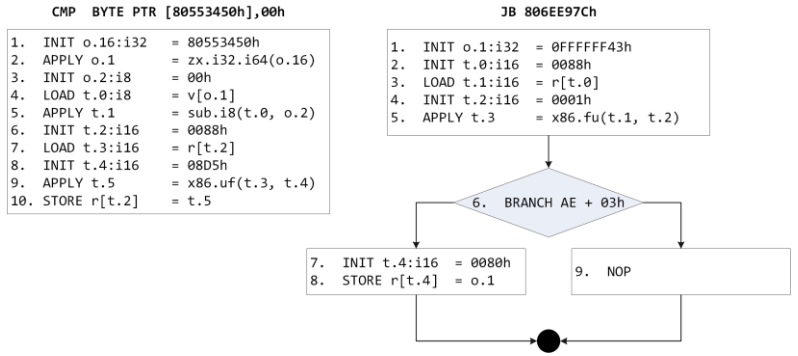

Рис. 3. Трансляция инструкций CMP и JB архитектуры Intel x86

Если элемент присутствует в наборе, то этот элемент считается символьным, в противном случае подставляется его конкретное значение. Для получения конкретных значений используется алгоритм восстановления буфера. Существуют ситуации, когда восстановить значение не удалось, в таком случае, элемент считается символьным. Пусть байт памяти по адресу `0x80553450` является символьным, тогда для оператора `Load` будет сгенерировано следующее выражение:

```
(select v_0 ((_ sign_extend 64)  #x80553450))
```

В итоге, для пятого оператора `APPLY` будет получено:

```
(bvsub (select v_0 ((_ sign_extend 64)  #x80553450)) #x00)
```

Инструкции с шестой по десятую не обрабатываются, поскольку они отвечают за обновление регистра флагов. Для того чтобы не создавать избыточность в уравнениях, происходит ленивое вычисление флагов. Результат трансляции машинной инструкции `CMP` – это выражение для пятого оператора и некоторая дополнительная информация необходимая для выставления флагов.

Теперь рассмотрим формирование уравнений для инструкции `JB 806EE97Ch`. Инструкции с 2-ой по 5-ую не обрабатываются, т.к. регистр флагов `r:[0x88]` не является символьным. Инструкции с номерами 1,7,8,9 не обрабатываются, т.к. регистр счетчика команд `r:[0x80]` также не является символьным. Таким образом, обрабатывается единственная шестая инструкция `Branch`.

Для условия оператора `Branch` выполняется вычисление нужных флагов и загрузка их на модельное слово состояния. В данном случае условие – `AE` (больше или равно для беззнаковых чисел). Вычисляется флаг `CF`, который загружается в модельное слово состояния. Затем составляется уравнение на выполнение инструкции `Branch`. В данном случае необходимо, чтобы значение флага `CF` было равно нулю. Затем добавляется уравнение, отражающее выполнение или невыполнение машинной инструкции условного перехода в трассе. Пусть условный переход был выполнен в трассе. Тогда для выполнения перехода `JB`, нужно добавить уравнение вида `(CF=0) = false`,

тем самым указав, что инструкция `Branch AE` не выполняется. Построенное уравнение становится частью предиката пути.

4.3 Поиск трамплинов

Для поиска трамплинов необходимо указать модули, которые относятся к адресному пространству приложения, и будут размещены по заведомо известным адресам. В коде указанных модулей происходит поиск инструкций-трамплинов. Эти инструкции имеют вид `jmp Reg` или `call Reg`. Они передают управление по адресу, записанному в регистре. Сначала стоит определить области памяти, в которых потенциально может быть размещен шелл-код. Из набора отслеживаемых (помеченных) ячеек памяти в момент аварийного завершения программы, отбираются ячейки, образующие непрерывные буферы достаточного размера для размещения шелл-кода. Если такие ячейки нашлись, то далее отбираются регистры, значения которых указывают на эти ячейки. Если таких регистров нет, то поиск трамплинов на этом завершается. В случае успеха, для отобранных регистров происходит поиск инструкций-трамплинов. Код операции и операнда-регистра таких инструкций занимает два байта, которые должны быть найдены в любых исполняемых секциях модулей. Следует отметить, что требуемые байты могут быть расположены, в том числе, и на границе двух разных инструкций. С ростом объема просматриваемых секций вероятность найти, как минимум, один трамплин растет достаточно быстро: для 500Кб она достигает 0.999.

4.4 Описание эксплойта и решение системы уравнений

Для получения эксплойта к предикату пути необходимо добавить уравнения описывающие этот эксплойт. Уравнения для перехвата управления после переполнения буфера на стеке можно разделить на два вида:

- уравнения для размещения шелл-кода в области памяти, находящейся под контролем нарушителя;
- передача управления в область памяти, находящейся под контролем нарушителя.

Для областей памяти, являющихся отслеживаемыми на момент аварийного завершения программы, выберем такие, размер которых больше или равен размеру шелл-кода. Если таких областей нет, то считаем, что данная уязвимость не эксплуатируема, в противном случае составим следующее уравнение для одной области. Пусть шелл-код представляет собой строку `"ABCD"`, диапазон адресов: от `1000` до `1003` включительно. Уравнение имеет вид:

`v(1000,1) =0x41` $\wedge$ `v(1001,1)=0x42` $\wedge$ `v(1002,1)=0x43` $\wedge$ `v(1003,1) =0x44`, где $\wedge$ - логическое "и".

Для описания передачи управления в область памяти, находящейся под контролем нарушителя, необходимо составить уравнение, описывающее то,

что в области памяти, где хранится адрес возврата из функции, будет храниться адрес шелл-кода.

Пусть `x` – это адрес, по которому хранится адрес возврата из функции , `l` – адрес шелл-кода или адрес трамплина. Тогда формула имеет вид: `v(x,1) = l[0]` $\wedge$ `v(x+1,1) = l[1]` $\wedge$ `v(x+2,1) = l[2]` $\wedge$ `v(x+3,1)= l[3]`, где $\wedge$ - логическое "и".

Объединяя уравнения размещения и передачи управления с предикатом пути, получим набор уравнений, которых достаточно, чтобы описать процесс формирования рабочего эксплойта. Далее уравнения подаются на вход решателю, и если система уравнений совместна ее решение является рабочим эксплойтом.

5. Реализация метода и результаты практического применения

Предложенный метод был реализован в виде модуля-расширения среды анализа бинарного кода, он использует такие ее возможности, как повышение уровня представления, модель процессора общего назначения, слайс трассы.

Для решения системы используется сторонний SMT-решатель, который был интегрирован в разработанный модуль-расширение. На сегодняшний день существует большое количество доступных решателей: MiniSat, OpenSMT, STP, Yices, Z3 и др. В разработанном инструменте используется SMT-решатель Z3, в силу следующих достоинств:

- инкрементальный подход к решению уравнений;
- поддержка большого количества типов данных, в том числе и машинных;
- доступно API на языке Си, которое позволяет работать с уравнениями в виде объектов в памяти, что гораздо эффективней, нежели работа с текстом;
- исходный код доступен по лицензии MSR-LA;
- высокая скорость работы по сравнению с другими решателями.

Разработанный инструмент был опробован на нескольких примерах. В качестве гостевых ОС использовались 32-битные версии Windows XP SP2, Windows XP SP3, Arch Linux (по состоянию на апрель 2014) и МСВС 3.0. Использовались приложения с известными уязвимостями, а также приложения из актуального дистрибутива Arch Linux, ошибки в которых были найдены с помощью фаззинга. Список анализируемых приложений приведен в табл. 1.

Табл. 1. Список анализируемых приложений.

Операционная система	Приложение	Уязвимость
Linux	corehttp 0.5.4	CVE:2007-4060

MCBC	libpng (konqueror)	CVE:2004-0597
Windows XP SP3	SuperPlayer 3500	EDB-ID:27041
Linux	iwconfig v26	CVE:2003-0947
Windows XP SP2	lhhtpd 0.1	CVE:2002-1549
Linux	get_driver (sysfsutils)	
Linux	mkfs.jfs (jfsutils)	
Linux	alsa_in (jack)	

Для фаззинга использовался метод, описанный в [14]. Фаззер запускает исследуемое приложение со всеми возможными однобуквенными параметрами командной строки (от `-a` до `-z` и от `-A` до `-Z`) и еще одним параметром длиной 6676 байт. Для 6607 установленных приложений фаззер получил 748 аварийных завершений для 42 различных приложений. Из этих 42 приложений были выбраны 3, распространяемые в рамках популярных пакетов программного обеспечения: `sysfsutils`, `jfsutils`, `jack`.

В табл. 2 приведены результаты работы алгоритма построения эксплойта: размер слайса обработки входных данных, размер блока входных данных, время генерации системы уравнений и время ее решения.

Табл. 2. Результаты работы алгоритма построения эксплойта.

Операционная система	Приложение	Размер слайса	Размер данных, байт	Время решения, с	Суммарное время работы, с
Linux	corehttp 0.5.4	18293	565	1024	1367
MCBC	libpng (konqueror)	2493	536	8	128
Windows XP SP3	SuperPlayer 3500	4855	594	<1	66
Linux	iwconfig v26	124	80	<1	7
Windows XP SP2	lhhtpd 0.1	20174	320	18	245
Linux	get_driver	152	272	2	41
Linux	mkfs.jfs	209	407	3	23
Linux	alsa_in	241	58	<1	40

Следует отметить, что для успешной эксплуатации уязвимости переполнения буфера на стеке с использованием современного дистрибутива ОС Linux были предприняты меры по отключению различных механизмов защиты.

- Отключена рандомизация адресного пространства.
- Исследуемые приложения компилированы с флагами `-fno-stack-protector` и `–U_FORTIFY_SOURCE`, что привело к отключению механизма защиты стека «канарейкой» и использования безопасных функций, предоставляемых компилятором `gcc`.
- С помощью утилиты `execstack` для исследуемых приложений было выставлено разрешение на выполнение кода на стеке.

Для других операционных систем (Windows XP и МСВС) дополнительные действия по отключению механизмов защиты не предпринимались. При построении эксплойтов для приложений из табл. 1, работающих под Windows XP, использовались трамплины из-за того, что адреса памяти, выделенные под стек, содержали нулевой байт, и поэтому адрес шелл-кода не мог быть напрямую записан в адрес возврата. Для приложений, работающих под Linux, эта проблема отсутствует, но использование трамплинов для обхода рандомизации не принесло результатов, так как единственные нерандомизированные участки кода принадлежали самим приложениям: из-за малого размера приложений алгоритм поиска трамплинов не дал результатов.

В список результатов не были включены приложения, для которых не удалось сгенерировать работающий эксплойт. В этих приложениях в кадре стека, помимо переполняемого буфера, были расположены другие переменные, перезапись которых приводила к преждевременному аварийному завершению без выполнения полезной нагрузки. Чаще всего, такие переменные содержали указатели на некоторую область памяти, после перезаписи таких переменных разыменование расположенных там указателей приводило к ошибке доступа к памяти. В других случаях перезаписывалась переменная, содержащая переменную цикла, что приводило к чтению данных по некорректному адресу. Для корректной работы эксплойта в этих случаях требуется перезапись соответствующих ячеек памяти корректными значениями.

6. Заключение

В статье представлен метод построения эксплойтов для обнаруженных ошибок. Метод основан на символьном выполнении бинарного кода, он позволяет преодолевать рандомизацию адресного пространства с помощью трамплинов, автоматизируются действия, которые не могут быть проведены без участия пользователя. Метод был реализован в виде программного инструмента, являющегося частью среды анализа бинарного кода. Его применение помогает разработчику понять, какие ошибки следует исправлять в первую очередь, как представляющие наибольшую угрозу для безопасности ПО.

Наиболее близкие результаты были показаны в работах коллектива из университета Карнеги – Меллон. В работе AEG [14] была представлена первая система для автоматической генерации эксплойтов. Поиск потенциально

эксплуатируемых уязвимостей ведется в исходном коде, а построение эксплойта происходит на уровне бинарного кода. Система MAYHEM [4] является развитием системы AEG для поиска уязвимостей и использует в своей работе только бинарный код. Их возможности позволяют перебирать состояния программы во время ее динамического символьного выполнения и обнаруживать срабатывание программных дефектов. К сожалению, все разработанные этим коллективом инструменты недоступны. Также следует отметить, что в приведенных публикациях авторы решают задачи в аналогичных ограничениях: отключаются некоторые механизмы защиты кода.

Помимо того, известны и другие системы символьного выполнения, работающие с бинарным кодом: BitFuzz [15], FuzzBall [16], S2E [17], SAGE [18], Avalanche [19] и др. Большинство из них в первую очередь нацелены на перебор состояний программы и не располагают средствами построения эксплойтов.

Представленные в данной работе результаты предлагают законченный метод, позволяющий оценить найденные ошибки. Построение эксплойта гарантирует безошибочную классификацию ошибки как критической. Дальнейшие работы предполагают построение эксплойта с использованием ROP-компиляции, позволяющей успешно преодолевать защиту в виде неисполняемого стека, эксплуатацию других типов уязвимостей, и практическое апробирование метода на других процессорных архитектурах.

Список литературы

[1]. А.Ю.Тихонов, А.И. Аветисян. Комбинированный (статический и динамический) анализ бинарного кода. // Труды Института системного программирования РАН, том 22, 2012 г. стр. 131-152.

[2]. King J.C. Symbolic execution and program testing. //*Commun. ACM.* – 1976. – №19.

[3]. Miller, C., Caballero, J., Johnson, N. M., Kang, M. G., McCamant, S., Poosankam, P., Song, D. Crash analysis with BitBlaze // *at BlackHat USA, 2010*.

[4]. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley. Unleashing MAYHEM on Binary Code // IEEE Symposium on Security and Privacy. – 2012.

[5]. Avgerinos, T., Rebert, A., Cha, S. K., & Brumley, D. (2014, May). Enhancing symbolic execution with veritesting. // *In ICSE*, May 2014, pp. 1083-1094.

[6]. В.А. Падарян, А.И. Гетьман, М.А. Соловьев, М.Г. Бакулин, А.И. Борзилов, В.В. Каушан, И.Н. Ледовских, Ю.В. Маркин, С.С. Панасенко. Методы и программные средства, поддерживающие комбинированный анализ бинарного кода. // Труды Института системного программирования РАН, том 26, 2014 г. Выпуск 1. Стр. 251-276. DOI: 10.15514/ISPRAS-2014-26(1)-8.

[7]. П.М. Довгалюк, Н.И. Фурсова, Д.С. Дмитриев. Перспективы применения детерминированного воспроизведения работы виртуальной машины при решении задач компьютерной безопасности. // Материалы конференции РусКрипто’2013. Москва, 27 — 30 марта 2013.

[8]. Довгалюк П.М., Макаров В.А., Падарян В.А., Романеев М.С., Фурсова Н.И. Применение программных эмуляторов в задачах анализа бинарного кода. // Труды

Института системного программирования РАН, том 26, 2014 г. Выпуск 1. Стр. 277-296. DOI: 10.15514/ISPRAS-2014-26(1)-9.

[9]. Андрей Тихонов, Арутюн Аветисян, Вартан Падарян. Методика извлечения алгоритма из бинарного кода на основе динамического анализа. // Проблемы информационной безопасности. Компьютерные системы. №3, 2008. Стр. 66-71

[10]. Падарян В. А., Соловьев М. А., Кононов А. И. Моделирование операционной семантики машинных инструкций. // Программирование, № 3, 2011 г. Стр. 50-64.

[11]. E. J. Schwartz, T. Avgerinos, and D. Brumley. Q: Exploit hardening made easy. // In Proc. of the USENIX Security Symposium, 2011.

[12]. Андрей Тихонов, Вартан Падарян. Применение программного слайсинга для анализа бинарного кода, представленного трассами выполнения. // Материалы XVIII Общероссийской научно-технической конференции «Методы и технические средства обеспечения безопасности информации». 2009. стр. 131

[13]. Silvio Ranise and Cesare Tinelli. The SMT-LIB Format: An Initial Proposal // Proceedings of PDPAR'03, July 2003

[14]. T. Avgerinos, S. K. Cha, Alexandre Rebert, Edard J. Schwartz, Maverick Woo, and D. Brumley. AEG: Automatic exploit generation // *Commun. ACM*. – 2014.– №2.

[15]. J. Caballero, P. Poosankam, S. McCamant, D. Babic, and D. Song. Input generation via decomposition and re-stitching: Finding bugs in malware. // In Proc. of the ACM Conference on Computer and Communications Security, Chicago, IL, October 2010.

[16]. L. Martignoni, S. McCamant, P. Poosankam, D. Song, and P. Maniatis. Path-exploration lifting: Hi-fi tests for lo-fi emulators. // In Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems, London, UK, Mar. 2012.

[17]. G. C. Vitaly Chipounov, Volodymyr Kuznetsov. S2E: A platform for in-vivo multi-path analysis of software systems. // In Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems, 2011, pp. 265–278.

[18]. P. Godefroid, M. Levin, and D. Molnar. Automated whitebox fuzz testing. // In Proc. of the Network and Distributed System Security Symposium, Feb. 2008.

[19]. Исаев, И. К., Сидоров, Д. В., Герасимов, А. Ю., Ермаков, М. К. (2011). Avalanche: Применение динамического анализа для автоматического обнаружения ошибок в программах использующих сетевые сокеты. Труды Института системного программирования РАН, том 21, 2011 г., стр. 55-70.

Automated exploit generation method for stack buffer overflow vulnerabilities

V. A. Padaryan
<vartan@ispras.ru>
V. V. Kaushan
<korpse@ispras.ru>
A. N. Fedotov
<fedotoff@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. In this paper automated method for exploit generation is presented. This method allows to construct exploits for stack buffer overflow vulnerabilities and also to prioritize software bugs. It is applied to program binaries, without requiring debug information. The method is based on dynamic analysis and symbolic execution. We present a tool implementing the method. We used this tool to generate exploits for 8 vulnerabilities in both Linux and Windows programs, 3 of which were undocumented at the time this paper was written.

Keywords: bug classification; vulnerability exploitation; binary code; dynamic analysis; symbolic execution.

References

[1]. Tikhonov A.YU., Avetisyan A.I. Kombinirovannyj (staticheskij i dinamicheskij) analiz binarnogo koda. [Combined (static and dynamic) analysis of binary code]. *Trudy ISP RAN [The Proceedings of ISP RAS]*, vol. 22, 2012, pp. 131-152 (in Russian).
[2]. King J.C. Symbolic execution and program testing. *Commun. ACM.* – 1976. – №19.
[3]. Miller, C., Caballero, J., Johnson, N. M., Kang, M. G., McCamant, S., Poosankam, P., Song, D. Crash analysis with BitBlaze. *at BlackHat USA, 2010*.
[4]. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley. Unleashing MAYHEM on Binary Code. *IEEE Symposium on Security and Privacy*, 2012.
[5]. Avgerinos, T., Rebert, A., Cha, S. K., & Brumley, D. (2014, May). Enhancing symbolic execution with veritesting. *In ICSE*, May 2014, pp. 1083-1094.
[6]. V.A. Padaryan, A.I. Getman, M.A. Solovyev, M.G. Bakulin, A.I. Borzilov, V.V. Kaushan, I.N. Ledovskich, U.V. Markin, S.S. Panasenko. Metody i programmnye sredstva, podderzhivayushhie kombinirovannyj analiz binarnogo koda [Methods and software tools for combined binary code analysis]. *Trudy ISP RAN [The Proceedings of ISP RAS]*, 2014, vol. 26, no. 1, pp. 251-276 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-8
[7]. Dovgalyuk P.M., Fursova N.I., Dmitriev D.S. Perspektivyi primeneniya determinirovannogo vosproizvedeniya rabotyi virtualnoy mashinyi pri reshenii zadach kompyuternoy bezopasnosti. [Prospects of using virtual machine deterministic replay in

solving computer security problems]. *Materialyi konferentsii RusKripto'2013 [The Proceedings RusCrypto'2013]*, 2014 (In Russian).

[8]. Dovgalyuk P.M., Makarov V.A., Padaryan, M.S. Romaneev, V.A., Fursova N.I. Primenenie programmnykh ehmulyatorov v zadachakh analiza binarnogo koda [Application of software emulators for the binary code analysis]. *Trudy ISP RAN [The Proceedings of ISP RAS]*, 2014, vol. 26, no. 1, pp. 277-296 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-9.

[9]. Tikhonov A.YU., Avetisyan A.I., Padaryan V.A., Metodika izvlecheniya algoritma iz binarnogo koda na osnove dinamicheskogo analiza [Methodology of exploring of an algorithm from binary code by dynamic analysis]. *Problemy informatsionnoj bezopasnosti. Komp'yuternye sistemy [Informations security aspects. Computer sistems]*, 2008, №3. pp. 66-71 (in Russian)

[10]. Padaryan V.A., Solov'ev M.A., Kononov A.I. Modelirovanie operatsionnoy semantiki mashinnyih instruktsiy. [Simulation of operational semantics of machine instructions]. *Programming and Computer Software*, May 2011, Volume 37, Issue 3, pp 161 – 170 , DOI 10.1134/S0361768811030030 (In Russian)

[11]. E. J. Schwartz, T. Avgerinos, and D. Brumley. Q: Exploit hardening made easy. *In Proc. of the USENIX Security Symposium*, 2011.

[12]. Tikhonov A.YU., Padaryan V.A., Primenenie programmnogo slaysinga dlya analiza binarnogo koda, predstavlennogo trassami vyipolneniya.[Using program slicing for bynary code represented by execution traces] *Materialyi XVIII Obscherossiyskoy nauchno-tehnicheskoy konferentsii «Metodyi i tehnicheskie sredstva obespecheniya bezopasnosti informatsii». [The Proceedings of XVIII Russian science technical conference "Methods and technical information security tools"]* 2009. pp 131 (In Russian).

[13]. Silvio Ranise and Cesare Tinelli. The SMT-LIB Format: An Initial Proposal. *Proceedings of PDPAR'03*, July 2003

[14]. T. Avgerinos, S. K. Cha, Alexandre Rebert, Edard J. Schwartz, Maverick Woo, and D. Brumley. AEG: Automatic exploit generation. *Commun. ACM*. – 2014.– №2.

[15]. L. Martignoni, S. McCamant, P. Poosankam, D. Song, and P. Maniatis. Path-exploration lifting: Hi-fi tests for lo-fi emulators. *In Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, London, UK, Mar. 2012.

[16]. J. Caballero, P. Poosankam, S. McCamant, D. Babic, and D. Song. Input generation via decomposition and re-stitching: Finding bugs in malware. *In Proc. of the ACM Conference on Computer and Communications Security*, Chicago, IL, October 2010.

[17]. G. C. Vitaly Chipounov, Volodymyr Kuznetsov. S2E: A platform for in-vivo multi-path analysis of software systems. *In Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011, pp. 265–278.

[18]. P. Godefroid, M. Levin, and D. Molnar. Automated whitebox fuzz testing. *In Proc. of the Network and Distributed System Security Symposium*, Feb. 2008.

[19]. Isaev, I. K., Sidorov, D. V., Gerasimov, A. YU., Ermakov, M. K. (2011). Primenenie dinamicheskogo analiza dlya avtomaticheskogo obnaruzheniya oshibok v programmakh ispol'zuyushhikh setevye sokety [Using dynamic analysis for automatic bug detection in software that use network sockets]. *Trudy ISP RAN [The Proceedings of ISP RAS]*, 2011, vol. 21, pp. 55-70 (In Russian).

Полиномиальный алгоритм проверки эквивалентности в модели программ с перестановочными и подавляемыми операторами

[1] *В.В. Подымов*
<valdus@yandex.ru>
[2] *В.А.Захаров*
<zakh@cs.msu.su>
[1] *Факультет ВМК, МГУ имени М.В. Ломоносова,*
119991 ГСП-1 Москва, Ленинские горы, 2-й учебный корпус
[2] *ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25*

Аннотация. В статье исследована задача проверки эквивалентности последовательных программ, некоторые операторы которых обладают свойствами перестановочности и подавления. Два оператора считаются перестановочными, если результат их последовательного выполнения не зависит от порядка, в котором эти операторы выполняются. Считается, что оператор $\boldsymbol{b}$ подавляет оператор $\boldsymbol{a}$, если последовательное выполнение операторов $\boldsymbol{a}$ и $\boldsymbol{b}$ дает такой же результат, что и выполнение одного лишь оператора $\boldsymbol{b}$. Разрешимость проблемы эквивалентности в исследуемой модели программ была установлена в 1971 г. А.А. Летичевским. Однако с тех пор вопрос о сложности проверки эквивалентности таких программ оставался открытым. Основной результат статьи – алгоритм, осуществляющий проверку эквивалентности программ с перестановочными и подавляемыми операторами за время, полиномиально зависящее от размеров анализируемых программ.

Ключевые слова: программа, перестановочные операторы, подавляемые операторы, эквивалентность программ, конечный автомат, разрешающий алгоритм, полиномиальная сложность.

1. Введение

На протяжении своего жизненного пути, от начальной разработки до заключительной модернизации, программа претерпевает разнообразные преобразования. В некоторых случаях, как, например, при компиляции или декомпиляции, преобразования программ проводятся автоматически. В других случаях, как, например, при модернизации программ, такие преобразования проводятся вручную. На каждом этапе, когда программа

подвергается изменению, необходимо следить за тем, чтобы применяемые преобразования не изменяли функций, вычисляемых программой, и не ухудшали вычислительные характеристики программы, такие как быстродействие, объем используемой памяти, размер и др.

Большим подспорьем здесь служат автоматические средства проверки эквивалентности и оптимизации программ. Оптимизирующие компиляторы снабжены эффективными процедурами, которые позволяют обнаруживать и устранять недостижимые и бесполезные фрагменты кода, оптимизировать структуру графов потока управления и линейных участков программ, экономить используемую память и пр.[1] Но все эти средства нуждаются в дальнейшем совершенствовании, поскольку во многих случаях они не способны провести глубокую оптимизацию даже простых фрагментов кода. Например, фрагмент программы π_1:

$$if\ P(z)\ then\ \{x = f(x); y = y + z\}\ else\ \{x = g(x); x = v + z\};$$

$$if\ !P(z)\ then\ \{y = y + z\}\ else\ \{x = v + z\}$$

эквивалентен линейному участку $\pi_2 : y = y + z; x = v + z$, однако современные компиляторы не способны провести оптимизирующие преобразования подобного вида. Поэтому разработка эффективных алгоритмов глубокого семантического анализа и оптимизации программ по-прежнему остается актуальной задачей теории программирования.

Для разработки таких алгоритмов выбирается и строится подходящая математическая модель программы таким образом, чтобы интересующая задача анализа или оптимизация программ сводилась некоторой общеизвестной математической задаче, наподобие задачи выполнимости логической формулы [2], вычисления хроматического числа графа [3] или минимизации автомата [4]. Алгоритмы решения этой задачи служат основой для построения процедуры решения соответствующей задачи системного программирования.

Этот подход был впервые предложен работах А.А. Ляпунова и Ю.И. Янова [5,6]. В них была введена и исследована первая математическая модель программ, получившая название схем Ляпунова-Янова. Используя систему эквивалентных преобразований, предложенную в [6] и усовершенствованную в [7], а также взаимосвязь схем Ляпунова-Янова с конечными автоматами, А.П. Ершов впервые применил оптимизирующие преобразования в трансляторе программ на языке Алгол [8], показав плодотворность такого подхода и придав тем самым мощный импульс развития теории схем программ.

Значительно более развитая алгебраическая модель программ, обобщающая схемы программ Ляпунова-Янова, была предложена в статье В.М. Глушкова и А.А. Летичевского [9]. В рамках этой модели можно, в частности, осуществлять эквивалентные преобразования фрагментов программы π_1 и π_2,

приведенных выше. В статье [9] был также предложен эффективный метод оптимизации программ, в котором задача минимизации программы (как по размеру, так и по быстродействию) сводится к задаче проверке эквивалентности схем программ. К сожалению, рекурсивный алгоритм проверки эквивалентности, предложенный в той же статье для некоторых классов моделей программ, имел экспоненциальную оценку сложность и был непригоден для применения на практике. Отсутствие быстрых и практичных алгоритмов проверки эквивалентности схем программ в моделях программ, учитывающих алгебраические свойства операторов, было главным препятствием дальнейшего внедрения результатов и методов теории схем программ в практику системного программирования [10].

Долгое время задача построения быстрых (полиномиальных по времени) алгоритмов проверки эквивалентности схем программ в алгебраических моделях программ не имела успешного решения. Первый результат в этом направлении был получен в статье [11]. Вскоре были предложены общие методы построения полиномиальных по времени алгоритмов проверки эквивалентности схем программ в моделях программ, учитывающих различные алгебраические свойства операторов [12,13]. Два свойства программных операторов оказались в центре внимания – перестановочность и подавляемость. Операторы a и b перестановочны, если результат вычисления программы не зависит от порядка выполнения этих операторов (т.е. $a; b = b; a$); оператор a подавляет оператор b, если результат выполнения оператора b несущественен, если вслед за ним выполняется оператор a (т.е. $a; b = b$). Интерес к этим алгебраическим свойствам программных операторов обусловлен двумя обстоятельствами.

Во-первых, отношения перестановочности и подавляемости операторов выявляются на основе простого синтаксического анализа операторов. Для каждого оператора a легко вычисляются два множества переменных: множество $Used(a)$, состоящее из всех тех переменных, значения которых используются при выполнении оператора, и множество $Mod(a)$, состоящее из всех тех переменных, значения которых изменяются в результате выполнения оператора. Нетрудно убедиться, что для перестановочности операторов a и b достаточно выполнимости равенств $Used(a) \cap Mod(b) = \emptyset$, $Used(b) \cap Mod(a) = \emptyset$, $Mod(a) \cap Mod(b) = \emptyset$, а для того чтобы оператор a подавлял оператор b, достаточно выполнимости равенств $Used(a) \cap Mod(b) = \emptyset$, $Mod(b) \subseteq Mod(a)$. На основании этих соотношений можно быстро проверить, что операторы $y = y + z$ и $x = v + z$ из примера, приведенного в начале этого раздела, перестановочны, и при этом оператор $x = v + z$ подавляет операторы $x = f(x)$ и $x = h(y)$.

Во-вторых, для многих оптимизирующих преобразований программ достаточно лишь привлечения отношений перестановочности и подавляемости операторов. Так, например, фрагменты программ, π_0 и π_1, приведенные в качестве примеров в начале этого раздела, эквивалентны в

алгебраической модели программ, в которой учитываются только два рассматриваемых свойства операторов.

Проблема эквивалентности программ в алгебраических моделях, учитывающих каждое из этих свойств по отдельности, уже была исследована ранее. В статье [13] описан алгоритм сложности $O(n^3)$, проверяющий эквивалентность программ с частично перестановочными операторами. Если в модели программ учитывается только отношение подавляемости операторов, то, как показано в работе [14], задача проверки эквивалентности принадлежит классу NL. Однако к комбинированным моделям программ общего вида, в которых учитываются оба эти свойства операторов, известные методы построения быстрых алгоритмов проверки эквивалентности оказались неприменимы (за исключением одного специального случая, исследованного в статье [15]).

Основной результат данной работы – описание полиномиального по времени алгоритма проверки эквивалентности произвольных программ с перестановочными и подавляемыми операторами. Статья состоит из 9 разделов. В начале статьи вводится модель пропозициональных последовательных программ. В разделе 2 приведены основные определения синтаксиса этой модели программ, а в разделе 3 определяется семантика этого класса абстрактных программ. Семантика программ определяется в алгебраических терминах на основе моноидов (полугрупп) с определяющими соотношениями (тождествами) перестановочности и подавления. В следующем разделе 4 исследованы наиболее важные для решения задачи проверки эквивалентности программ алгебраические свойства моноидов с тождествами перестановочности и подавления. В разделе 5 введен граф совместных вычислений – основная структура, при помощи которой проводится проверка эквивалентности программ. В этом же разделе доказана теорема 1 о необходимых и достаточных условиях эквивалентности программ; эти условия представлены в терминах устройства графа совместных вычислений анализируемых программ. В разделе 6 исследованы особенности применения тождеств подавления операторов $a; b = b$ в контексте перестановочности некоторых операторов. Показано, что эффект подавления операторов в операторных цепочках может быть описан при помощи конечных автоматов. Соответствующее утверждение (лемма 8) является ключевым в построении разрешающего алгоритма. В разделе 7 приводятся утверждения, обеспечивающие полиномиальный обход графа совместных вычислений. Существование такого обхода совместно с упомянутой теоремой 1 обеспечивает полиномиальную разрешимость проблемы эквивалентности программ в рассматриваемой модели вычислений. Описанию этого алгоритма и обоснованию его корректности посвящен раздел 8. В заключительном разделе 9 обсуждаются перспективы практического использования результата этой статьи для решения задач системного программирования, а также возможности дальнейших исследований, развивающих полученный результат.

Описание разрешающего алгоритма и Теорема 2 – основные результаты статьи, – являются заслугой первого из авторов статьи.

2. Основные определения

Далее считаем заданными конечные алфавиты операторов $\mathfrak{A}$ и логических условий $\mathfrak{C}$. Слова в алфавите операторов $\mathfrak{A}$ будем называть *термами*. Запись h^- будет обозначать зеркальный образ терма h.

Последовательной программой, или просто программой, будем называть систему $\pi = (L, l^{in}, l^{out}, T, B)$, состоящую из конечного множества состояний управления L, входа $l^{in} \in L$, выхода $l^{out} \in L$, функции переходов $T: (V \setminus \{l^{out}\}) \times \mathfrak{C} \to L$ и привязки операторов $B: L \to \mathfrak{A}$. Под размером $|\pi|$ программы π будем понимать число ее состояний управления. Состояние управления будем называть завершаемым, если из него достижим выход программы.

Пара состояний управления l_1, l_2 программы π образуют шаг вычисления $l_1 \overset{\delta}{\to} l_2$, если $T(l_1, \delta) = l_2$. Путем в программе π назовем всякую последовательность шагов вычисления $pt = l_1 \overset{\delta_1}{\to} l_2 \overset{\delta_2}{\to} \ldots$. Путь назовем полным, если он бесконечен или оканчивается в выходе. Путь, начинающийся во входе программы, назовем ее трассой. Полную трассу назовем вычислением. Записью $B(pt)$ обозначим терм $B(l_1)B(l_2)\ldots$; для простоты выкладок полагаем $B(l^{in}) = B(l^{out}) = \lambda$, где λ — пустое слово.

Определим семантику программ, основываясь на детерминированных динамических шкалах и моделях [16]. *Шкала* $\mathcal{F} = (S, s^0, R)$ задает множество *состояний данных* S, *начальное состояние* $s^0 \in S$ и *интерпретации операторов* $R: S \times \mathfrak{A} \to S$. Введем следующее обобщение R^* функции R с множества $\mathfrak{A}$ на $\mathfrak{A}^*$: $R^*(s, \lambda) = s$, $R(s, ah) = R^*(R(s, a), h)$. Для краткости вместо $R^*(s^0, h)$ будем писать $[h]$.

Модель $\mathcal{M} = (\mathcal{F}, \xi)$ задается шкалой $\mathcal{F} = (S, s^0, R)$ и *оценкой логических условий* $\xi: S \to \mathfrak{C}$. Будем говорить, что *трасса реализуется в модели* $\mathcal{M}$, если для любого ее префикса $tr \overset{\delta}{\to} l$ верно равенство $\xi([tr]) = \delta$. Такую трассу для краткости будем называть *трассой в модели* $\mathcal{M}$. Результатом конечного вычисления cp в модели $\mathcal{M}$ будем называть состояние данных $[cp]$. Бесконечные вычисления считаем безрезультатными. Заметим, что в любой модели $\mathcal{M}$ всегда реализуется ровно одно вычисление программы.

Трассы tr_1, tr_2 в произвольных программах π_1, π_2 будем называть $\mathcal{F}$-*совместными*, где $\mathcal{F}$ — произвольная шкала, если они реализуются в какой-либо модели $\mathcal{M} = (\mathcal{F}, \xi)$. Программы π_1, π_2 назовем $\mathcal{F}$-*эквивалентными* ($\pi_1 \sim_{\mathcal{F}} \pi_2$), если все их $\mathcal{F}$-совместные вычисления имеют одинаковый результат.

Свойства перестановочности и подавляемости операторов естественным образом описываются соотношениями перестановочности ($ab = ba$, где $a, b \in \mathfrak{A}$) и подавления ($ab = b$, где $a, b \in \mathfrak{A}$). Рассматриваемые далее шкалы задаются посредством моноидов $\mathfrak{M} = (S_{\mathfrak{M}}, *)$, которые порождаются конечным множеством образующих $\mathfrak{A}$ и определяются разнообразными семействами соотношений перестановочности и подавления. Элементом $\langle h \rangle$ моноида $\mathfrak{M}$, порожденным термом $h \in \mathfrak{A}^*$, является множество всех термов, которые можно получить из h , применив конечное число раз определяющие соотношения этого моноида. Операция умножения элементов моноида определяется следующим образом: $\langle h \rangle * \langle g \rangle = \langle hg \rangle$. Указанные моноиды с тождествами перестановочности и подавления будем называть *СА-моноидами*. Терм h , имеющий наименьшую в классе $\langle h \rangle$ длину, будем называть *минимальным представителем* (класса $\langle h \rangle$). *Нормой* $\|s\|$ элемента s моноида будем называть длину минимальных представителей этого элемента.

Будем говорить, что шкала $\mathcal{F} = (S, s^0, R)$ задается моноидом $\mathfrak{M} = (S_{\mathfrak{M}}, *)$, если верны следующие соотношения: $S = S_{\mathfrak{M}}$; $s^0 = \langle \lambda \rangle$; $R(s, a) = s * \langle a \rangle$. Шкалу назовем *упорядоченной*, если $[h_1 h_2] \neq [h_1]$. Далее будут рассматриваться только упорядоченные шкалы $\mathcal{F}(C, A)$, задаваемые моноидами $\mathfrak{M}(C, A)$ с определяющими соотношениями $C \cup A$, где C содержит только соотношения перестановочности и A – только соотношения подавления. На множестве состояний данных упорядоченной шкалы введем частичный порядок: состояние s_1 предшествует состоянию s_2 (обозначение $s_1 \leq s_2$), если $R(s_1, h) = s_2$ для некоторого терма $h \in \mathfrak{A}^*$. Несравнимость состояний s_1, s_2 по отношению порядка $\leq$ обозначим записью $s_1 \perp s_2$.

Утверждение 1 [13]. Пусть $\mathcal{F}$ - произвольная упорядоченная шкала. Тогда трассы tr_1, tr_2 являются $\mathcal{F}$-совместными в том и только в том случае, если для любых их префиксов $tr_1' \xrightarrow{\delta_1} l_1$, $tr_2' \xrightarrow{\delta_2} l_2$ верно следующее: если $[tr_1] = [tr_2]$, то $\delta_1 = \delta_2$.

В работе [13] предложен метод решения проблема эквивалентности пропозициональных последовательных программ на упорядоченных шкалах $\mathcal{F}$: для заданных программ π_1 , π_2 и заданной упорядоченной шкалы $\mathcal{F}$ выяснить, верно ли соотношение $\pi_1 \sim_{\mathcal{F}} \pi_2$. Этот метод будет использован в настоящей статье для построения полиномиального по времени работы (относительно размеров программ) разрешающего алгоритма для упорядоченных шкал $\mathcal{F}(C, A)$, задаваемых моноидами $\mathfrak{M}(C, A)$.

3. Алгебраические свойства СА-моноидов

В данном разделе сформулированы основные свойства моноидов $\mathfrak{M}(C, A)$, которые понадобятся далее для построения алгоритма проверки эквивалентности программ с перестановочными и подавляемыми операторами. Обоснование утверждений этого раздела проводится с

использованием традиционной для теории полугрупп техники доказательств; в связи с ограничениями на объем статьи эти доказательства опущены.

Утверждение 2 [17]. CA-моноид $\mathfrak{M}(C,A)$ с множеством определяющих соотношений коммутативности C и подавления A упорядочен тогда и только тогда, когда не существует ни одной пары операторов $a,b \in \mathfrak{A}$, для которой выполняется как равенство $ab = ba$ в моноиде $\mathfrak{M}(C)$ с соотношениями коммутативности C, так и равенство $ab = b$ в моноиде $\mathfrak{M}(A)$ с соотношениями подавления A.

Утверждение 3. Пусть CA-моноид $\mathfrak{M}(C,A)$ упорядочен, и h — такой минимальный представитель, для которого верно $[ah] \neq [h]$ для некоторого оператора $a \in \mathfrak{A}$. Тогда терм ah — также минимальный представитель.

Будем говорить, что моноид $\mathfrak{M}$ обладает свойством левого сокращения, если для любых его элементов s, s_1, s_2 верно следующее: если $s * s_1 = s * s_2$, то $s_1 = s_2$.

Утверждение 4. Всякий упорядоченный CA-моноид $\mathfrak{M}(C,A)$ обладает свойством левого сокращения.

Утверждение 5. Пусть термы h, g являются минимальными представителями некоторого элемента упорядоченного CA-моноида $\mathfrak{M}(C,A)$. Тогда терм g может быть получен из терма h применением только соотношений перестановочности.

Автоматом будем называть систему $\mathbb{A} = (Q, q^0, T, L_{\mathbb{A}}, B_{\mathbb{A}})$, где Q – конечное множество состояний, q^0 – инициальное состояние, $T: Q \times \mathfrak{A} \to Q$ – функция переходов, $L_{\mathbb{A}}$ – множество меток и $B_{\mathbb{A}}$ – разметка состояний. Далее будем использовать следующие обозначения:

- $\mathbb{A}(q,h)$ - состояние, в которое автомат $\mathbb{A}$ переводится словом h из состояния q;
- $\mathbb{A}(h) = \mathbb{A}(q^0,h)$; $B_{\mathbb{A}}(h) = B_{\mathbb{A}}(\mathbb{A}(h))$.

Будем говорить, что автомат $\mathbb{A}$ распознает подавление операторов, если выполнены следующие условия: $L_{\mathbb{A}} = 2^{\mathfrak{A}}$; если $\langle h_1 \rangle = \langle h_2 \rangle$, то $\mathbb{A}(h_1^-) = \mathbb{A}(h_2^-)$; $B_{\mathbb{A}}(h^-) = \{a \mid \langle ah \rangle = \langle h \rangle\}$.

Утверждение 6. Пусть моноид $\mathfrak{M}(C,A)$ упорядочен. Тогда существует автомат, распознающий подавление операторов.

4. Граф совместных вычислений

Пусть заданы упорядоченная шкалу $\mathcal{F}(C,A) = (S, s^0, R)$ и программы $\pi_i = \left(L_i, l_i^{in}, l_i^{out}, T_i, B_i\right)$, $i \in \{1,2\}$. Рассмотрим пошаговое построение совместных вычислений программ π_1, π_2, организованное следующим образом. В начале вычисления программ трассы пусты. Предположим, что для программ уже построены трассы tr_1, tr_2. Если $[tr_1] = [tr_2]$, то выбирается произвольное логическое условие δ, и обе программы делают шаг вычисления согласно этому условию. Если программа π_2 «отстает» от программы π_1, т. е. $[tr_2] <$

$[tr_1]$, то программа π_2 делает шаг вычисления. Иначе программа π_1 делает шаг вычисления.

В графе Г совместных вычислений программ π_1, π_2 сохранена информация о такой совместной работе программ, достаточная для исследования проблемы эквивалентности. Вершинами *графа совместных вычислений* Г являются четверки вида (l_1, l_2, s_1, s_2), где $l_i \in L_i$ и $s_i \in S$, $i \in \{1,2\}$. Корнем графа Г объявляется вершина $\left(l_1^{in}, l_2^{in}, s^0, s^0\right)$. Вершину $v = (l_1, l_2, s_1, s_2)$ назовем *опровергающей*, если выполнено одно из трех условий:

1. $l_1 = l_1^{out}, s_2 \neq s^0$;
2. $l_2 = l_2^{out}, s_1 \neq s^0$;
3. $l_1 = l_1^{out}, l_2 = l_2^{out}, (s_1, s_2) \neq (s^0, s^0)$.

Опровергающие вершины и вершину $(l_1^{out}, l_2^{out}, s^0, s^0)$ будем называть терминальными.

Дуги графа Г помечены парами $(d_1, d_2) \in (\mathfrak{C} \cup \{\varepsilon\})^2$. Из терминальных вершин не исходит ни одной дуги. Различные дуги, исходящие из нетерминальной вершины, помечены различными парами. Метки дуг, исходящих из вершины $v = (l_1, l_2, s_1, s_2,)$, образуют множество: $\mathfrak{C} \times \{\varepsilon\}$, если $l_2 = l_2^{out}$ или $s_2 \neq s^0$; $\{(\delta, \delta) \mid \delta \in \mathfrak{C}\}$, если $l_1 \neq l_1^{out}$, $l_2 \neq l_2^{out}$ и $s_1 = s_2 = s^0$; $\{\varepsilon\} \times \mathfrak{C}$ в остальных случаях. Вершина (l_1', l_2', s_1', s_2'), достижимая из v по дуге с меткой (d_1, d_2), определяется следующим образом:

- если $d_i = \varepsilon$, то $v_i' = v_i$ и $s_i'' = s_i$, иначе $v_i \xrightarrow{d_i} v_i'$ и $s_i'' = s_i * [B_i(v_i')]$, $i \in \{1,2\}$,
- если $s_1'' = s_2''$, то $s_1' = s_2' = s^0$;
- если $s_1'' \perp s_2''$, то $s_1' = s_1''$ и $s_2' = s_2''$;
- если $s_i'' < s_{3-i}''$, то $s_i' = s^0$ и $s_{3-i}'' = s_i'' * s_{3-i}'$.

Корректность этого описания следует из свойства левого сокращения (утверждение 4).

Корневым маршрутом графа Г назовем маршрут, начинающийся в его корне. Корневой маршрут будем называть *опровергающим* в следующих двух случаях:

1. этот маршрут оканчивается в опровергающей вершине;
2. существуют натуральное число N и индекс i, такие что при отбрасывании N первых вершин маршрута все i-е компоненты меток дуг равны ε и все i-е компоненты вершин завершаемы.

Далее мы покажем, что существование опровергающих маршрутов в графе совместных вычислений Г равносильно неэквивалентности программ.

Чтобы установить соответствие между корневыми маршрутами графа Г и совместными трассами программ π_1, π_2, введем понятие *проекции* $pr(\Omega)$ корневого маршрута Ω графа Г:

- $pr(\Omega) = \left(pr_1(\Omega), pr_2(\Omega)\right)$;
- $pr_i\left(l_1^{in}, l_2^{in}, s^0, s^0\right) = l_i^{in}$;
- $pr_i\left(\Omega \xrightarrow{d_1,d_2} (l_1, l_2, s_1, s_2)\right) = pr_i(\Omega)$, если $d_i = \varepsilon$;
- $pr_i\left(\Omega \xrightarrow{d_1,d_2} (l_1, l_2, s_1, s_2)\right) = pr_i(\Omega) \xrightarrow{d_i} l_i$, если $d_i \neq \varepsilon$.

Лемма 1. Пусть Ω — корневой маршрут графа Г. Тогда:

1. $pr_1(\Omega)$ и $pr_2(\Omega)$ — совместные трассы программ π_1, π_2;
2. если маршрут Ω оканчивается в вершине (l_1, l_2, s_1, s_2), то:
   1. для некоторого состояния данных s верны соотношения $[pr_i(\Omega)] = s * s_i$, $i \in \{1,2\}$;
   2. $s_2 \in \{[a] \mid a \in \mathfrak{A}\} \cup \{s^0\}$;
   3. если $s_1 \neq s^0$ и $s_2 \neq s^0$, то $s_1 \perp s_2$.

Лемма 1 легко обосновывается индукцией по длине маршрута Ω с привлечением утверждения 1. Если маршрут Ω бесконечен, то совместность его проекций вытекает из совместности любых конечных префиксов его проекций.

Лемма 2. Пусть cp_1, cp_2 — совместные вычисления программ π_1, π_2. Тогда в графе совместных вычислений Г существует корневой маршрут Ω, который либо является бесконечным, либо оканчивается в терминальной вершине, такой что $pr_i(\Omega)$ – это префикс вычисления cp_i для $i \in \{1,2\}$.

Для обоснования леммы 2 достаточно построить маршрут согласно описанию совместной работы программ и убедиться, что ни на каком шаге построения не нарушается совместность трасс (утверждение 1, лемма 1). Если процесс построения маршрута бесконечен, то в качестве результата берется бесконечный маршрут, префиксами которого являются маршруты на всех шагах построения.

Лемма 3. Пусть Ω — корневой маршрут графа Г с проекцией (tr_1, tr_2) и для некоторого $i \in \{1,2\}$ трасса tr_i является префиксом трассы tr. Тогда трассы tr и tr_{3-i} совместны.

Доказательство. Без ограничения общности считаем $i = 1$. Предположим, что трассы tr и tr_2 несовместны. По утверждению 1 равенство $[tr_1'] = [tr_2']$ выполнено для некоторых собственных префиксов tr_1', tr_2' трасс tr, tr_2. По утверждению 1 трассы tr_1 и tr_2 совместны, а значит, $tr_1' = tr_1 \to pt$. Рассмотрим кратчайший префикс Ω' маршрута Ω, для которого $pr_2(\Omega') = tr_2'$. Привлекая утверждение 1, лемму 1 и описание совместной работы программ, легко убедиться в невозможности достроить маршрут Ω' до Ω.

Теорема 1. $\pi_1 \sim_{\mathcal{F}} \pi_2$ тогда и только тогда, когда в графе Г не содержится ни одного опровергающего маршрута.

Доказательство. Достаточность. Рассмотрим опровергающий маршрут Ω. По определению он либо бесконечен, либо оканчивается в опровергающей вершине. Если он оканчивается в вершине, содержащей выходы программ π_1, π_2, то по лемме 1 получим совместные конечные вычисления с различными результатами. Если он оканчивается в вершине, содержащей выход l_i^{out} ровно одной программы, то, продолжив трассу $pr_{3-i}(\Omega)$ произвольным образом до вычисления и используя утверждение 4 и леммы 1, 3, получим либо совместные конечные вычисления с различными результатами, либо совместные конечное и бесконечное вычисление. Пусть теперь маршрут Ω бесконечен. По лемме 1 его проекцией являются бесконечное вычисление программы π_i и конечная трасса программы π_{3-i}, где $i \in \{1,2\}$, причем по определению опровергающего маршрута и лемме 3 трассу $pr_{3-i}(\Omega)$ можно продолжить до конечного вычисления программы π_{3-i}, совместного с вычислением $pr_i(\Omega)$.

Необходимость. Программы π_1, π_2 имеют либо совместные конечные вычисления с различными результатами, либо совместные конечное и бесконечное вычисления. Пусть это вычисления cp_1, cp_2. Рассмотрим корневой маршрут Ω графа Γ, подходящий под условие леммы 2. Если маршрут Ω конечен, то он оканчивается в опровергающей вершине (иначе по лемме 1 результаты вычислений cp_1, cp_2 совпадают). Если маршрут Ω бесконечен, то, привлекая определение проекции корневого маршрута, получаем, что Ω — бесконечный опровергающий маршрут. Доказательство теоремы завершено.

5. Эффекты подавления

Все дальнейшие рассуждения строятся на основе существования функции $æ_s$, где $s \in S$, определяемой следующим образом: $æ_s(s_1) = s_2$, где $s_1 * s = s_2 * s$ и s_2 имеет наименьшую возможную норму. Функцию $æ_s$ будем называть *эффектом подавления*, индуцированным состоянием данных s, или, коротко, -подавлением. Существование -подавления для рассматриваемой нами шкалы вытекает из следующего утверждения.

Лемма 4. Если уравнение $X * s = s'$ имеет решение относительно X, то ровно одно его решение имеет наименьшую среди всех решений норму.

Доказательство. Предположим, что s_1 и s_2 — два различных решения уравнения $X * s = s'$, имеющие наименьшую среди всех решений норму. Пусть h_i — минимальный представитель класса s_i, $i \in \{1,2\}$, и h — минимальный представитель класса s. По утверждению 3 термы h_1h, h_2h являются минимальными представителями одного состояния данных. По утверждению 5 терм h_2h может быть получен из h_1h применением только соотношений перестановочности. Взяв зеркальные образы двух последних термов, применив утверждение 4 и применив повторно зеркальное

преобразование, получим вывод терма h_2 из h_1. Доказательство леммы завершено.

Множество всевозможных эффектов подавлений далее будем обозначать записью Æ. Введем порядок на множестве Æ: $æ_1 \le æ_2$ тогда и только тогда, когда существуют состояния данных s, s', такие что $s \le s'$, $æ_1 = æ_s$ и $æ_2 = æ_{s'}$.

Лемма 5. Отношение $\le$ на множестве Æ является нестрогим частичным порядком.

Доказательство. Предположим, что существуют два различных эффекта подавления $æ_1$, $æ_2$, такие что $æ_1 \le æ_2$ и $æ_2 \le æ_1$. Заметим, привлекая утверждение 3, что если $æ(s) = s'$ и h — минимальный представитель класса s, то можно получить минимального представителя h' класса s вычеркиванием некоторых символов из h, причем вычеркиваемые символы однозначно определяются эффектом подавления æ. Из определения порядка на множестве Æ следует: если $æ \le æ'$, $æ(s) = s_1$ и $æ'(s) = s_2$ то существуют минимальные представители h_1, h_2 классов s_1 и s_2 соответственно, такие что h_2 получается из h_1 вычеркиванием некоторых вхождений. Используя последний факт, получаем, что если $æ_1 \le æ_2$ и $æ_2 \le æ_1$, то $æ_1 = æ_2$.

Лемма 6. $æ_s(s_1) = æ_s(s_2)$ тогда и только тогда, когда $s_1 * s = s_2 * s$.

Лемма 7. Если $æ(s_1) = æ(s_2)$ и $æ \le æ'$, то $æ'(s_1) = æ'(s_2)$.

Лемма 6 следует из леммы 4 и определения эффекта подавления, лемма 7 следует из леммы 6.

Будем говорить, что автомат $\mathbb{A}$ с множеством меток Æ распознает эффекты подавления, если выполнены следующие два условия:

автомат $\mathbb{A}$ согласован с моноидом: для любой пары термов h_1, h_2 если $[h_1] = [h_2]$, то $\mathbb{A}(h_1) = \mathbb{A}(h_2)$;

автомат $\mathbb{A}$ согласован с эффектами подавления: $B_{\mathbb{A}}(h) = æ_h$ для любого терма h.

Для автомата $\mathbb{A}$, согласованного с моноидом, будем использовать записи $\mathbb{A}([h])$, $B_{\mathbb{A}}([h])$ наравне с $\mathbb{A}(h)$, $B_{\mathbb{A}}(h)$.

Лемма 8. Существует автомат, распознающий эффекты подавления.

Доказательство. Приведем явное описание автомата $\mathbb{A}$, распознающего эффекты подавления. Начнем построение с автомата $\mathbb{A}_0$, распознающего подавление операторов (утверждение 6). Состояния q_1, q_2, этого автомата объявим эквивалентными, если равенство $B_{\mathbb{A}_0}(q_1, h) = B_{\mathbb{A}_0}(q_2, h)$ выполнено для всех термов $h \in \mathfrak{U}^*$. Переберем классы эквивалентности состояний автомата $\mathfrak{U}_0$ и назначим i-му классу новую метку $æ_{\mathrm{i}}$. В результате такой замены получим автомат $\mathbb{A}_1$. Заметим (утверждение 3), что $\mathbb{A}_1(h_1) = \mathbb{A}_2(h_2)$ тогда и только тогда, когда $æ_{g_1} = æ_{g_2}$, где g_i есть терм h_i, записанное наоборот, $i \in \{1,2\}$. Таким образом, можно считать, что $L_{\mathbb{A}_1} = Æ$. Отбросив

метки вершин и назначив вершины с меткой $æ_i$, получим автомат-распознаватель $\mathbb{A}_2^i$. Используя известные результаты теории автоматов, можно построить по автомату $\mathbb{A}_2^i$ автомат $\mathbb{A}_3^i$. Можно легко убедиться, что автомат $\mathbb{A}_3^i$ принимает язык $\{h \mid æ_{[h]} = æ_i\}$. Рассмотрим декартово произведение автоматов $\mathbb{A}_3^i$, назначив метку $æ_i$ состоянию $(q_1, \dots, q_k)$, где q_i — финальное состояние автомата $\mathbb{A}_3^i$ (заметим, что по определению эффекта подавления индекс i однозначно определен для любого состояния декартова произведения). Описанное декартово произведение и будет искомым автоматом $\mathbb{A}$. Доказательство леммы завершено.

В дальнейших рассуждениях считаем заданным автомат $\mathbb{A} = (Q, q^0, T_{\mathbb{A}}, L_{\mathbb{A}}, B_{\mathbb{A}})$, распознающий эффекты подавления.

Пусть $s \in S$ и $h = a_1 \dots a_k \in \mathfrak{A}^*$. Тогда *развитием s-подавления вдоль терма h* (коротко, (s,h)-*подавлением*) назовем последовательность $æ_s^h$, получаемую из последовательности $(æ_s, æ_{s*[a_1]}, \dots, æ_{s*[h]})$ удалением соседних повторяющихся элементов. (s,pt)-подавлением, где $s \in S$ и pt – конечный путь в произвольной программе, будем называть $(s, B(pt))$-подавлением. Если путь pt бесконечен, то (s,pt)-*подавлением* назовем наибольшее по длине (s,pt')-подавление, где pt' — конечный префикс пути pt. Существование (s,pt)-подавления для бесконечного пути pt следует из очевидного факта: любое (s,pt)-подавление является цепью в конечном (лемма 8) частично упорядоченном (лемма 5) множестве $Æ$.

Конечный путь pt в программе назовем $\dot{s}$-*приведенным*, где $s \in S$, если для любых его различных префиксов pt_1, pt_2 они оканчиваются в различных состояниях управления или $\mathbb{A}(s * [pt_1]) \neq \mathbb{A}(s * [pt_2])$. Бесконечный путь pt будем называть $\dot{s}$-*приведенным*, если он представим в виде $pt_1(\to pt_2)^\omega$, где $pt_1 \to pt_2$ — конечный $\dot{s}$-приведенный путь.

Лемма 9. Пусть l — состояние управления программы π, pt — путь в программе π, начинающийся в состоянии l, и $s \in S$. Тогда существует $\dot{s}$-приведенный путь pt' в программе π, начинающийся в состоянии управления l и такой что: $\dot{æ}_s^{pt} = \dot{æ}_s^{pt'}$. При этом путь pt' конечен тогда и только тогда, когда путь pt конечен; если путь pt' конечен, то он завершается в том же состоянии управления, что и pt.

Доказательство. Опишем алгоритм построения пути pt' по pt. Пусть путь pt конечен. Если он не является $\dot{s}$-приведенным, то существуют два различных префикса pt_1 и $pt_1 \to pt_2$ этого пути, оканчивающиеся в одном состоянии управления и такие что $\mathbb{A}(s * [pt_1]) = \mathbb{A}(s * [pt_1 \to pt_2])$. Удалим из пути pt подпуть pt_2, и если результат не является $\dot{s}$-приведенным, то повторим процедуру. Путь pt конечен, а следовательно, за конечное число удалений будет получен искомый $\dot{s}$-приведенный путь pt'.

Допустим, что путь pt бесконечен. Рассмотрим произвольный конечный префикс $pt'' \to l$ пути pt, такой что $\dot{\text{æ}}_{\text{s}}^{\text{pt}''} \neq \dot{\text{æ}}_{\text{s}}^{pt}$ и $\dot{\text{æ}}_{\text{s}}^{pt''\to l} = \dot{\text{æ}}_{\text{s}}^{\text{pt}}$. Применим к нему процедуру удаления подпути. Добавим к результирующему пути бесконечный «хвост», отброшенный перед удалением. Найдем в этом хвосте первое повторение состояния управления и заменим хвост на бесконечное повторение найденного цикла. Результатом будет искомый путь pt'. Доказательство завершено.

Лемма 10. Пусть $s \in S$ и pt — $\dot{s}$-приведенный путь в программе π, представимый в виде $pt = pt'(\to pt'')^{\omega}$ (здесь путь pt'' считается пустым в том и только в том случае, если путь pt конечен). Тогда $|pt'| + |pt''| \leq |\pi| \cdot |Q|$.

Справедливость леммы 10 объясняется тем, что существует ровно $|\pi| \cdot |Q|$ различных пар (l, q), где l – состояние управления программы π и q – состояние автомата $\mathbb{A}$.

Пусть $s \in S$ и $h = a_1 \dots a_k \in \mathfrak{A}^*$. Тогда *обобщенным развитием s-подавления вдоль терма h*, или, коротко, $[\![s, h]\!]$-подавлением, будем называть последовательность $\ddot{\text{æ}}_{\text{s}}^{\text{h}}$, получаемую из последовательности $\left((\text{æ}_{\text{s}^0}, \text{æ}_s), (\text{æ}_{[a_1]}, \text{æ}_{s*[a_1]}), \dots, (\text{æ}_{[\text{h}]}, \text{æ}_{s*[\text{h}]})\right)$ удалением соседних повторяющихся элементов. $[\![s, pt]\!]$-подавлением, где $s \in S$ и pt — конечный путь в произвольной программе, будем называть $[\![s, [pt]]\!]$-подавление. Если путь pt бесконечен, то $[\![s, pt]\!]$-подавлением будем называть наибольшее по длине $[\![s, pt']\!]$-подавление, где pt' — конечный префикс пути pt. Существование $[\![s, pt]\!]$-подавления для бесконечного пути pt следует из очевидного факта: любое $[\![s, pt]\!]$-подавление является цепью в декартовом квадрате конечного частично упорядоченного множества *Æ*.

Конечный путь pt (в какой-либо программе) будем называть *$\ddot{s}$-приведенным*, где $s \in S$, если для любых его различных префиксов pt_1, pt_2, оканчивающихся в состояниях управления l_1, l_2 соответственно, выполнено: $\left(l_1, \mathbb{A}([pt_1]), \mathbb{A}(s * [pt_1])\right) \neq \left(l_2, \mathbb{A}([pt_2]), \mathbb{A}(s * [pt_2])\right)$. Бесконечный путь pt будем называть *$\ddot{s}$-приведенным*, если он представим в виде $pt_1(\to pt_2)^{\omega}$, где $pt_1 \to pt_2$ — конечный $\ddot{s}$-приведенный путь. $\ddot{s}$-приведенный путь будем также называть $\mathbb{A}(s)$-приведенным.

Лемма 11. Пусть l — состояние управления программы π, pt — путь в программе π, начинающийся в состоянии l, и $s \in S$. Тогда существует $\ddot{s}$-приведенный путь pt' в программе π, начинающийся в состоянии управления l и такой что: $\ddot{\text{æ}}_{\text{s}}^{\text{pt}} = \ddot{\text{æ}}_{\text{s}}^{\text{pt}'}$. При этом путь pt' конечен тогда и только тогда, когда путь pt конечен; если путь pt' конечен, то он оканчивается в том же состоянии управления, что и путь pt.

Лемма 12. Пусть $s \in S$ и pt — $\ddot{s}$-приведенный путь в программе π, представимый в виде $pt = pt_1(\to pt_2)^{\omega}$ (здесь путь pt_2 считается пустым в

том и только в том случае, если путь pt конечен). Тогда $|pt_1| + |pt_2| \le |\pi| \cdot |Q|^2$.

Обоснования лемм 11, 12, по существу, повторяют обоснования лемм 9, 10 с незначительными изменениями и потому опущены.

6. Анализ графа совместных вычислений

В данном разделе приводятся утверждения, обеспечивающие полиномиальный обход графа Г. Существование такого обхода совместно с упомянутой теоремой 1 обеспечивает полиномиальную разрешимость проблемы эквивалентности программ в рассматриваемой модели вычислений.

Лемма 13. Пусть $q \in Q$, $l_1 \in L_1$, $l_2 \in L_2$, $s_2 \in S$, $m = |Q|^3 \cdot |\pi_1| \cdot |\pi_2|^2 + 2$ и:

- $l_1 \to pt_1$ и $l_2 \to pt_2$ — полные пути программ π_1, π_2, и хотя бы один из этих путей конечен;
- $\ddot{æ}_{q}^{pt_1} = \left((æ_1^1, æ_1^2), \dots, (æ_k^1, æ_k^2)\right)$;
- $\dot{æ}_{s_2}^{pt_2} = (æ_1, \dots, æ_s)$;
- $1 \le k' \le k$;
- из корня графа Г достижимы -вершины $v_i = \left(l_1, l_2, s_1^i, s_2\right)$, где $i \in \{1, \dots, m\}$;
- состояния данных $æ_{k'}^1\left(s_1^i\right)$, $i \in \{1, \dots, m\}$ попарно различны;
- $\{æ_1, \dots, æ_s\} \cap \left\{æ_{k'+1}^2, \dots, æ_k^2\right\} = \emptyset$.

Тогда $\pi_1 \nsim \pi_2$.

Доказательство. Пусть Ω_i — корневой маршрут, оканчивающийся в вершине v_i, $i \in \{1,2\}$, и $pr(\Omega_i) = \left(tr_1^i \to l_1, tr_2^i \to l_2\right)$. Из леммы 1 получим представления $\left[tr_1^i \to l_1\right] = s^i * s_1^i$ и $\left[tr_2^i \to l_2\right] = s^i * s_2$. Приведем подробное обоснование только для одного случая: состояние управления l_2 завершаемо, $k' = k$, путь pt_1 бесконечен. Остальные случаи обосновываются схожим образом, дополнительно привлекая леммы 10, 11, и потому опущены. Без ограничения общности полагаем, что путь pt_1 q-приведен (лемма 9). Пусть $l_2 \to pt$ – кратчайший полный путь в программе π_2. Достаточно показать, что хотя бы одна пара вычислений $cp_1^i = tr\_1\text{^}i \to pt_1$, $cp_2^i = tr_2^i \to pt$ совместна. Предположим, что это не так. Тогда по утверждению 1 для любого индекса $j \in \{1, \dots, m\}$ найдутся собственные префиксы $\widehat{tr}_1^j \xrightarrow{\delta_1} l_1'$, $\widehat{tr}_2^j \xrightarrow{\delta_2} l_2'$ вычислений cp_1^j, cp_2^j, такие что $\left[\widehat{tr}_1^j\right] = \left[\widehat{tr}_2^j\right]$ и $\delta_1 \ne \delta_2$. Привлекая утверждение 1 и лемму 3, получим неравенства $\left|\widehat{tr}_1^j\right| > \left|tr_1^j\right|$, $\left|\widehat{tr}_2^j\right| > \left|tr_2^j\right|$. Тогда по утверждению 4 для некоторых префиксов $\widehat{pt}_1^j$, $\widehat{pt}_2^j$ путей pt_1, pt верно равенство $s_1^j * \left[\widehat{pt}_1^j\right] = s_2 * \left[\widehat{pt}_2^j\right]$. Из неравенства $|pt| \le |\pi_2|$ получаем следующие соотношения:

$\left\|s_1^j * \left[\widehat{pt}_1^j\right]\right\| = \left\|s_2 * \left[\widehat{pt}_2^j\right]\right\| \le |\pi_2|$. По лемме 12 получим неравенство $\left|\widehat{pt}_1^j\right| \le |Q|^2 \cdot |\pi_1| \cdot |\pi_2|$. Следовательно, пара длин $\left(\left|\widehat{pt}_1^j\right|, \left|\widehat{pt}_2^j\right|\right)$ может принимать лишь $(m-2)$ различных значений, а значит, найдутся различные индексы p, q, такие что $\left(\widehat{pt}_1^p, \widehat{pt}_2^p\right) = \left(\widehat{pt}_1^q, \widehat{pt}_2^q\right)$. Привлекая леммы 6, 7, получим равенство $æ_k\left(s_1^p\right) = æ_k\left(s_1^q\right)$, что противоречит условию леммы. Полученное противоречие завершает доказательство леммы.

Лемма 14. Пусть $q \in Q$, $l_1 \in L_1$, $l_2 \in L_2$, $s_2 \in S$, $m = |Q|^3 \cdot |\pi_1| \cdot |\pi_2|^2 + 2$ и:

- $l_1 \to pt_1$ и $l_2 \to pt_2$ — полные пути программ π_1, π_2;
- $ä_q^{pt_1} = \left((æ_1^1, æ_1^2), \dots, (æ_k^1, æ_k^2)\right)$;
- $ȧ_{s_2}^{pt_2} = (æ_1, \dots, æ_s)$;
- из корня графа Г достижимы -вершины $v_i = \left(l_1, l_2, s_1^i, s_2\right)$, где $i \in \{1, \dots, m\}$;
- $1 \le k' < k$, $1 \le s' < s$;
- состояния данных $æ_{k'}^1\left(s_1^i\right)$, $i \in \{1, \dots, m\}$, попарно различны;
- состояния данных $æ_{k'+1}^1\left(s_1^i\right)$, $i \in \{1, \dots, m\}$, совпадают;
- $\{æ_1, \dots, æ_{s'}\} \cap \{æ_{k'+1}^2, \dots, æ_k^2\} = \emptyset$;
- $æ_{s'+1} \in \{æ_{k'+1}^2, \dots, æ_k^2\}$;
- Ω — корневой маршрут графа Г, оканчивающийся в вершине v_m;
- $pr_1(\Omega) \to pt_1$ и $pr_2(\Omega) \to pt_2$ — совместные вычисления программ π_1, π_2, имеющие различные результаты.

Тогда для некоторого $j \in \{1, \dots, m\}$ из вершины v_j исходит опровергающий маршрут.

Доказательство. Пусть Ω_i — корневой маршрут графа Г, оканчивающийся в вершине v_i, $i \in \{1,2\}$, и $pr(\Omega_i) = \left(tr_1^i \to l_1, tr_2^i \to l_2\right)$. По лемме 1 получим представления $\left[tr_1^i \to l_1\right] = s^i * s_1^i$ и $\left[tr_2^i \to l_2\right] = s^i * s_2$. Рассмотрим альтернативные случаи.

Случай 1: одно из состояний управления l_i незавершаемо. Без ограничения общности полагаем $i = 1$. Тогда вычисление $pr_1(\Omega) \to pt_1$ бесконечно, и из условия леммы получаем, что путь pt_2 конечен. По лемме 3 трасса $tr_1^1 \to l_1$ и вычисление $cp_2 = tr_2^1 \to l_2 \to pt_2$ совместны. Используя утверждение 1, можно продолжить трассу $tr_1^1 \to l_1$ до бесконечного вычисления, совместного с конечным вычислением cp_2.

Случай 2: состояние управления l_1 завершаемо и существует бесконечный путь $l_2 \to pt_2'$, такой что $ȧ_{s_2}^{pt_2'}$ есть префикс $ȧ_{s_2}^{pt_2}$ длины менее s'. Рассуждая так же, как и в доказательстве леммы 13, можно получить совместные вычисления $tr_1^j \to l_1 \to \overline{\overline{pt}}_1$, $tr_2^j \to l_2 \to \overline{\overline{pt}}_2$ программ π_1, π_2. Достроим

маршрут Ω_j наидлиннейшим образом до маршрута Ω' так же, как и в обосновании леммы 2. Используя лемму 1, можно легко показать, что маршрут Ω' является опровергающим.

Случай 3: состояние управления l_2 завершаемо и существует бесконечный путь $l_1 \to pt_1'$, такой что $ä_q^{pt_1'}$ есть префикс $ä_q^{pt_1}$ длины не более k'. Рассуждения данного случая аналогичны случаю 2.

Случай 4: состояния управления l_1, l_2 завершаемы и не существует бесконечных путей, удовлетворяющих условиям случаев 2, 3. Пусть $pt_1' \to l_1'$, $pt_2' \to l_2'$ — префиксы путей pt_1, pt_2, такие что $\left|ä_q^{pt_1'}\right| = k' \neq \left|ä_q^{pt_1' \to l_1'}\right|$ и $\left|ȧ_{s_2}^{pt_2'}\right| = s' \neq \left|ȧ_{s_2}^{pt_2' \to l_2'}\right|$. По леммам 10, 12 имеем неравенства $|pt_1'| \leq |Q|^2 \cdot |\pi_1|$, $|pt_2'| \leq |Q| \cdot |\pi_2|$. Рассуждая так же, как в доказательстве леммы 13, получим индекс $j \in \{1, \dots, m-1\}$, такой что трассы $tr_1^j \to l_1 \to pt_1' \to l_1'$ и $tr_2^j \to l_2 \to pt_2' \to l_2'$ совместны. Достаточно показать, что вычисления $cp_1^j = tr_1^j \to l_1 \to pt_1$ и $cp_2^j = tr_2^j \to l_2 \to pt_2$ совместны и имеют различные результаты. Различие результатов может быть легко получено с использованием утверждения 4, леммы 7, представлений $\left[tr_1^i \to l_1\right] = s^i * s_1^i$ и $\left[tr_2^i \to l_2\right] = s^i * s_2$ и того факта, что результаты вычислений $cp_1^m = tr_1^m \to l_1 \to pt_1$, $cp_2^m = tr_2^m \to l_2 \to pt_2$ различны. Совместность вычислений cp_1^j, cp_2^j может быть легко получена из совместности вычислений cp_1^m, cp_2^m с использованием утверждений 1, 6 и лемм 3, 8.

7. Разрешающий алгоритм

Далее полагаем $n = \max(|\pi_1|, |\pi_2|)$. Опишем алгоритм α, разрешающий проблему эквивалентности последовательных программ на упорядоченной шкале $\mathcal{F}(C, A)$. Рассмотрим произвольный обход графа Г (например, обход в глубину или в ширину). Модифицируем этот обход следующим образом. Разобьем рассмотренные при обходе вершины на $O(n^2)$ групп, определяя в одну группу вершины (l_1, l_2, s_1, s_2), (l_1', l_2', s_1', s_2'), если $l_1 = l_1'$, $l_2 = l_2'$, $\mathbb{A}(s_1) = \mathbb{A}(s_1')$ и $s_2 = s_2'$. При рассмотрении очередной вершины v производятся следующие действия. Если v – опровергающая вершина, то алгоритм останавливается и констатирует неэквивалентность программ. Если через вершину v и остальные рассмотренные вершины проходит цикл, являющийся «хвостом» бесконечного опровергающего маршрута, то алгоритм останавливается и констатирует неэквивалентность программ. Если при добавлении вершины v в свою группу число вершин в ее группе оказывается $(|Q|^3 \cdot n^3 + 2)$ вершин, то согласно леммам 13, 14 либо программы признаются неэквивалентными, либо дуги, исходящие из вершины v, игнорируются в дальнейшем обходе. Если обход завершен и программы не признаны неэквивалентными, то они признаются эквивалентными.

Лемма 15. Равенство состояний $[h_1]$, $[h_2]$ шкалы $\mathcal{F}(C,A)$ может быть проверено за время $O(m^2)$, где $m = \max(|h_1|, |h_2|)$.

Доказательство. По утверждениям 3, 6 минимальные представители g_1, g_2 классов $[h_1]$, $[h_2]$ могут быть построены за время $O(m)$. По утверждению 5 достаточно проверить, можно ли получить g_2 из g_1, применяя только соотношения перестановочности. Если $|g_1| \neq |g_2|$, то этого сделать нельзя. Пусть теперь $|g_1| = |g_2|$. Если $g_1 = \lambda$, то достаточно проверить равенство $g_2 = \lambda$. Иначе рассмотрим самый левый символ a терма g_1 и самое левое вхождение того же символа в терм g_2. Если такое вхождение не найдено, то $[h_1] \neq [h_2]$. Если это вхождение найдено, но слева от него есть вхождение символа b, такое что a и b не перестановочны, то $[h_1] \neq [h_2]$. Иначе вычеркнем левый символ терма g_1 и рассматриваемое вхождение в терм g_2 и повторим описанную процедуру проверки для полученных более коротких термов. Всего производится не более m процедур проверки, каждая из которых завершается за время $O(m)$.

Лемма 16. Алгоритм α завершает свою работу за время $O(n^{18})$.

Доказательство. Алгоритм α исследует $O(n^5)$ вершин графа Г (леммы 13, 14). Для хранения состояний данных в каждой вершине достаточно термов длины $O(n^5)$. При исследовании вершины производится не более $O(n^3)$ сравнений состояний данных (леммы 13, 14). Сравнение двух состояний данных может быть произведено за время $O(n^{10})$ (лемма 15).

Лемма 17. $\pi_1 \sim \pi_2$ тогда и только тогда, когда алгоритм α признает программы эквивалентными.

Справедливость леммы 17 следует из лемм 16, 13, 14 и теоремы 1. Леммы 16, 17 подводят итог исследованию, проведенному в данной работе, и позволяют считать обоснованной следующую теорему.

Теорема 2. Пусть шкала $\mathcal{F} = \mathcal{F}(C,A)$ упорядочена. Тогда задача проверки $\mathcal{F}$-эквивалентности программ π_1, π_2 разрешима за полиномиальное относительно размера программ время.

8. Заключение

Основной результат статьи – теорему 2 – можно оценивать двояко. В работе [9] показано, каким образом, используя алгоритм проверки эквивалентности, можно построить эффективный алгоритм минимизации программ, операторы которых являются образующими элементами упорядоченного левосократимого моноида. Таким образом, полиномиальный алгоритм проверки эквивалентности программ с перестановочными и подавляемыми операторами, предложенный в разделе 8, свидетельствует о том, что оптимизировать программы можно столь же эффективно и просто, как и конечные автоматы. Однако высокая степень полинома, оценивающего сложность этого алгоритма, пока не дает оснований говорить о том, что этот метод оптимизации программ является практически пригодным. Поэтому цель

дальнейших исследований – совершенствование предложенного алгоритма в стремлении понизить его сложность до величины порядка $n^2 \div n^4$. Результаты работ [13-15] позволяют полагать, что такие ожидания небеспочвенны.

Список литературы

[1]. Ахо А., Лам М., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструментарий. — 2-е издание. — М.: «Вильямс», 2008. — 1184 с.

[2]. Hoare C.A.R. An axiomatic basis for computer programming // Communications of the ACM. – 1969. – v. 12, N 10, p. 576-580.

[3]. Ершов А.П. Сведение задачи экономии памяти при составлении программ к задаче раскраски вершин графа // Доклады АН СССР. – 1962. – т. 142, N 4. – с. 785–787.

[4]. Глушков В.М. Теория автоматов и формальные преобразования микропрограмм // Кибернетика. – 1965. – N 5. – с. 1–9.

[5]. Ляпунов А.А. О логических схемах программ // Проблемы кибернетики, вып. 1. – М.:Физматгиз, 1958. – с. 46-74.

[6]. Янов Ю.И. О логических схемах алгоритмов // Проблемы кибернетики, вып. 1. – М.:Физматгиз, 1958. – с. 75-127.

[7]. Ершов А.П. Операторные схемы (Об операторных схемах Янова) // Проблемы кибернетики, вып. 20. – М.:Физматгиз, 1967. – с. 181-200.

[8]. Ershov A.P. Alpha – an automatic programming system of high efficiency // Journal of the Association for Computing Machinary. – 1966. – v. 13, N 1. – p. 17-24.

[9]. Глушков В.М., Летичевский А.А. Теория дискретных преобразователей. // Избранные вопросы алгебры и логики: сб.статей. – Новосибирск: Наука, 1973. – с. 5-39.

[10]. Ершов А.П. Современное состояние теории схем программ // Проблемы кибернетики, вып. 27. – М.:Наука, 1973. – с. 87-110.

[11]. Подловченко Р.И., Захаров В.А. Полиномиальный по сложности алгоритм, распознающий коммутативную эквивалентность схем программ // Доклады РАН, серия Информатика. – 1998. – т. 362, N 6. – с. 27-31.

[12]. Zakharov V.A. An efficient and unified approach to the decidability of equivalence of propositional program schemes // Lecture Notes in Computer Science. – 1998. – v. 1443. – p. 247-258.

[13]. Захаров В.А. Быстрые алгоритмы разрешения эквивалентности операторных программ на уравновешенных шкалах // Математические вопросы кибернетики, вып.7. – М.:Физматлит, 1998. – с. 303-324.

[14]. Захаров В.А. Проверка эквивалентности программ при помощи двухленточных автоматов // Кибернетика и системный анализ. – 2010. – N 4. – с. 39-48.

[15]. Захаров В.А., Щербина В.Л. Об эквивалентности программ с операторами, обладающими свойствами коммутативности и подавления // Материалы 9-го Международного семинара «Дискретная математика и ее приложения», Москва, 2007. – 2007. – с. 191-194.

[16]. Harel D., Kozen D., Tiuryn J. Dynamic logic. MIT Press, Cambridge, MA, USA. – 2000. – 450 p.

[17]. Подымов В.В., Захаров В.А. Об одной полугрупповой модели программ, определяемой при помощи двухленточных автоматов // Научные ведомости Белгородского государственного университета. Серия История, экономика, политология, информатика, том 14. – № 7. – с. 94-101.

A polynomial algorithm for checking the equivalence in models of programs with commutation and vast operators

[1] *V.V. Podymov*
<*valdus@yandex.ru*>
[2] *V.A. Zakharov*
<zakh@cs.msu.su>
[1] *Lomonosov Moscow State University, Faculty CMC,*
2nd Education Building, GSP-1, Leninskie Gory, Moscow
[2] *ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation*

Abstract. In this paper we study the equivalence problem in the model of sequential programs which assumes that some instructions are commutative and absorbing. Two instructions are commutative if the result of their executions does not depend on an order of their execution. An instruction $\boldsymbol{b}$ absorbs an instruction $\boldsymbol{a}$ if the sequential composition $\boldsymbol{a};\boldsymbol{b}$ yields the same result as the single instruction $\boldsymbol{b}$. A.A. Letichevskij in 1971 proved the decidability of equivalence checking problem in this model of programs. Nevertheless a possibility of building polynomial time equivalence checking procedures remains an open problem till nowadays. The main result of this paper is the description of a polynomial time algorithm for checking the equivalence of sequential programs with commutative and absorbing instructions. The paper includes 9 sections. In Section 1 we introduce informally the model of programs under consideration, the equivalence checking problem, and give a brief overview of the preceding results in the study of equivalence checking problem for this model. In Sections 2 and 3 the syntax and a semigroup-based semantics of propositional model of sequential programs are formally defined. The algebraic properties of semigroups of commutative and absorbing program instructions are studied in Section 4. In Section 5 we introduce a graph of joined computations which is the key structure in designing our equivalence checking techniques. In Section 6 and 7 we show that the cumulative absorbing effect of finite sequences of commutative instructions can be specified by means of finite automata; this is another key step in building the decision procedure. In Section 8 we show that equivalence checking of two programs is reducible to a traversal of a bounded fragment of the graph of joint computations of these programs; the latter can be performed in time polynomial of the size of programs to be checked.

Keywords: program, commuting instructions, absorbing instructions, program equivalence, finite automata, decision procedure, polynomial time complexity.

References

[1]. Aho A., Lam M., Sethi R, Ullman J. D. Compilers: Principles, Techniques, and Tools. Pearson Education, Ltd., 2014, 940 p.

[2]. Hoare C.A.R. An axiomatic basis for computer programming. Communications of the ACM. 1969, v. 12, N 10, p. 576-580.

[3]. Ershov, A. P. Axiomatics for Memory Allocation, *Acta Inform.*, Vol. 6, 1976, pp. 61-75.

[4]. Glushkov V.M. Teoriya avtomatov i formalnyie preobrazovaniya mikroprogramm [Automata theory and formal transformations of microprograms]. Kybernetika [Cybernetics], 1965, N 5.

[5]. Lyapunov A.A. O logicheskih shemah programm [On the logical program schemata]. Problemy kibernetiki [Problems of cybernetics], 1958, vol. 1, p. 46-74.

[6]. Yanov Ju. I. O logicheskih shemah algoritmov [On the logical algorithm schemata]. Problemy kibernetiki [Problems of cybernetics], 1958, vol. 1, p. 75-127.

[7]. Ershov A.P. Operatornye skhemy (Ob operatornykh skhemakh Yanova) [Operating schemata (On Yanov operator shemata)]. Problemy kibernetiki [Problems of cybernetics], 1967, v. 20, p. 181-200.

[8]. Ershov A.P. Alpha – an automatic programming system of high efficiency // Journal of the Association for Computing Machinary. – 1966. – v. 13, N 1. – p. 17-24.

[9]. Glushkov V.M., Letichevsky A.A. Teoriya diskretnykh preobrazovateley [Theory of discrete transducers]. Izbrannye voprosy algebry i logiki: sb.statey.. Izbrannyje voprosy algebry I logiki [Selected problems in algebra and logics: Proceedings], 1973, Novosibirsk: Nauka, p. 5-39.

[10]. Ershov, A. P., Theory of Program Schemata. *Proc. IFIP 1971*, North-Holland, Amsterdam, pp. 144-163.

[11]. Podlovchenko R.I., Zakharov V.A. Polinomial'niy po slojnosti algoritm raspoznayushchiy ekvivalentost skhem program [Polynomial equivalence checking algorithm for program schemata], Doklady Mathematics *(Doklady Akademii Nauk)*, 1998, vol. 362, N 6, p. 27-31.

[12]. Zakharov V.A. An efficient and unified approach to the decidability of equivalence of propositional program schemes. Lecture Notes in Computer Science, 1998, v. 1443, p. 247-258.

[13]. Zakharov V.A. Byistryie algoritmyi razresheniya ekvivalentnosti operatornyih programm na uravnoveshennyih shkalah [Swift algorithms deciding equivalence of operator schemata on length-preserving frames]. Matematicheskie voprosyi kibernetiki [Mathematical Problems of Cybernetics], vol.7. – Moscow PhysMathLit, 1998, p. 303-324.

[14]. Zakharov V.A. Program equivalence checking by two-tape automata. Cybernetics and Systems Analysis. 46, № 4, p. 554-562.

[15]. Zakharov V.A., Shcherbina V.L. Ob ekvivalentnosti programm s operatorami, obladayushchimi svoystvami kommutativnosti i podavleniya [On the equivalence of programs with commutative and absorbing operators]. Materialy 9-go Mezhdunarodnogo seminara «Diskretnaya matematika i eye prilozheniya» [Proceedings of the 9-th International Workshop "Discrete mathematics and its application"], Mosocw, 2007, p. 191-194.

[16]. Harel D., Kozen D., Tiuryn J. Dynamic logic. MIT Press, Cambridge, MA, USA. – 2000. – 450 p.

[17]. Podymov V.V., Zakharov V.A. Ob odnoy polugruppovoy modeli programm, opredelyaemoy pri pomoshchi dvukhlentochnykh avtomatov [On a semigroup model of

programs specified by two-tape automata]. Nauchnye vedomosti Belgorodskogo gosudarstvennogo universiteta. Seriya Istoriya, ekonomika, politologiya, informatika [Scientific Bulletin of Belgorod State University: History, Economics, Politology, Informatics], vol 14. N 7, p. 94-101.

Современное состояние исследований в области обфускации программ: определения стойкости обфускации

[1] *Н.П. Варновский*
[2] *В.А.Захаров <zakh@cs.msu.su>*
[3] *Н.Н. Кузюрин <nnkuz@ispras.ru>*
[3] *А.В. Шокуров <shok@ispras.ru>*
[1] *Институт проблем информационной безопасности, 119192, г. Москва, Мичуринский проспект, 1, офис 10*
[2] *Факультет ВМК, МГУ имени М.В. Ломоносова, 119991 ГСП-1 Москва, Ленинские горы, 2-й учебный корпус*
[3] *ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25*

Аннотация. Обфускацией программ называется такое эквивалентное преобразование программ, которое придает программе форму, затрудняющую понимание алгоритмов и структур данных, реализуемых программой, и препятствующую извлечению из текста программы определенной секретной информации, содержащейся в ней. Поскольку обфускация программ может найти широкое применение при решении многих задач криптографии и компьютерной безопасности, задаче оценки стойкости обфускации придается очень большое значение, начиная с самых первых работ в этой области. В этой статье приводится обзор различных определений стойкости обфускации программ и результатов, устанавливающих возможность или невозможность построения стойкой обфускации программ в тех или иных криптографических предположениях.

Ключевые слова: программа, обфускация, стойкость обфускации, сложность, машина Тьюринга, модель черного ящика

1. Введение

Обфускацией программы называется всякое ее преобразование, которое сохраняет вычисляемую программой функцию (эквивалентное преобразование), но при этом придает программе такую форму, что извлечение из текста программы (программного кода) ключевой информации об алгоритмах и структурах данных, реализованных в этой программе, становится трудоемкой задачей. Обфускация программ в противоположность реорганизации (рефакторингу) преследует цель затруднить понимание программ и воспрепятствовать целенаправленной их модификации. Поэтому задачу обфускации программ можно счесть одной из задач системного

программирования, подобной другим задачам преобразования программ – трансляции, оптимизации, реорганизации, распараллеливания. С другой стороны, обфускацию можно также рассматривать как особую разновидность шифрования программ. В отличие от традиционных видов шифрования обфускация не предполагает построения эффективных алгоритмов расшифрования, т.е. восстановления исходного текста программы, но зато требует сохранения смысла зашифрованного сообщения – функции, вычисляемой обфускируемой программой. Поэтому задача обфускации программ может быть также отнесена к области криптографии и криптоанализа. Именно двойственность этой задачи и объясняет тот факт, что ее исследование вот уже более 15 лет проводится по двум направлениям – со стороны системного программирования и со стороны криптографии, – которые очень мало взаимодействуют друг с другом. Цель настоящей статьи – ознакомить специалистов в области системного программирования с теми результатами изучения задачи обфускции программ, которые были достигнуты к настоящему времени в области математических основ криптографии. Эта статья открывает цикл работ, посвященных вопросам обфускации программ, рассматриваемых с точки зрения математики. При изучении математической проблемы обфускации программ начинать нужно с определения стойкости обфускации. Требования стойкости существенно зависят от тех приложений, в которых используется обфускация. И поэтому в первой статье этого цикла мы рассмотрим и проанализируем определения понятия стойкости обфускации программ. В последующей работе планируется провести исследование тех задач системного программирования и криптографии, для решения которых можно было бы использовать обфускацию программ, и оценить, в какой мере современные методы обфускации приблизились к их решению. И, наконец, в последней статье цикла мы расскажем о новейших достижениях в решении задачи построения стойких обфускаторов программ.

Вероятно, задача обфускации была впервые упомянута (без явного употребления термина «обфускация») в 1976 году в основополагающей работе Диффи и Хеллмана [1]. Желая проиллюстрировать концепцию шифрования с открытым ключом, они предложили следующую простую схему ее реализации. Выбирается произвольная криптосистема с секретным ключом, в процедуру шифрования вставляется секретный ключ, и затем инициализированная этим ключом программа шифрования запутывается так, чтобы извлечение из ее текста секретного ключа было очень трудной задачей. Таким образом, модифицированная процедура шифрования становится открытым ключом новой криптосистемы. Запутывание процедуры шифрования с целью предотвращения извлечения из ее текста секретного ключа является одним из возможных применений обфускации программ для решения некоторых задач криптографии и компьютерной безопасности.

В явном виде понятие обфускации программ было введено в 1997 году в работе Коллберга, Томборсона и Лоу [2]. Авторы этой работы рассматривали обфускацию программ, в первую очередь, как средство защиты прав интеллектуальной собственности на алгоритмы, которые реализуются в программах с открытым кодом. В работе [2] были предложены простейшие виды обфускирующих преобразований программ, проведена их систематическая классификация и прослежена взаимосвязь задачи обфускации программ с некоторыми известными задачами системного программирования.

Если вкратце подытожить результаты исследования проблемы обфускации программ за прошедшие 18 лет, то можно сделать следующие выводы.

1. Спектр задач, для решения которых можно было бы использовать алгоритмы программной обфускации, весьма обширен, и цели применения обфускации могут быть противоположны. Обфускацию можно использовать как для защиты программ от вирусных атак [3], так и для маскировки компьютерных вирусов [4,5]. При обфускации программ для нужд криптографии целью маскировки является сокрытие данных (секретного ключа), но не алгоритмов. Но когда методы обфускации применяются для обеспечения компьютерной безопасности, то целью маскировки является сокрытие алгоритмов, но не обрабатываемых данных. Таким образом, “проблема обфускации программ” включает в себя целое семейство задач маскировки программ, для каждой из которых вводятся специальные требования стойкости обфускации.
2. Имеется большой разрыв между теоретическими требованиями стойкости обфускации программ и применяемыми на практике методами и средствами решения этой задачи. Известно немало работ, в которых предлагаются различные практические методы обфускации программ; подробное описание многих из них представлено в монографии [6]. Некоторые из этих методов были реализованы в коммерческих программных продуктах (см., например, [7-12]). Однако влияние фундаментальных теоретических результатов (см. [13-16]) на эту ветвь развития программного обеспечения минимально: требования безопасности, исследуемые в контексте криптографических приложений, либо являются слишком сильными, либо неадекватны тем задачам защиты программного обеспечения, которые возникают на практике.
3. Между положительными и отрицательными результатами решения задачи обфускации образовался большой разрыв. В статье [13] доказано, что для некоторых строго формализованных определений стойкости обфускации (стойкость в модели виртуального «черного ящика») существуют такие семейства эффективно вычислимых функций, которые не допускают стойкой обфускации. Стойкую обфускацию удалось построить для существенно более слабых

требований стойкости и лишь для очень простых функций - точечных функций и близких к ним семейств функций [17-21]. Несмотря на то, что положительные результаты были обобщены в статье [22-24] для более широких классов функций, вопрос о (не)возможности эффективной обфускации для общих криптографических протоколов или же для любого значимого класса программ (например, для конечных автоматов) при стандартных криптографических предположениях остается открытым.

Приведенные выводы показывают, что проблема обфускации программ – это очень сложная и многогранная задача, для которой вряд ли удастся найти единый универсальный метод решения. Мы надеемся, что дальнейший прогресс в исследовании этой проблемы позволит сформировать математические основы для создания широкого многообразия формальных концепций и методов обфускации программ в контексте различных приложений. Создание такого математического аппарата следует начать с разработки различных определений стойкости обфускации и исследования взаимосвязи между предложенными определениями и подходящими понятиями и моделями дискретной математики, математической криптографии, теории сложности вычислений. Это поможет нам открыть наиболее важные свойства для всех типов обфускации программ. Располагая спектром различных определений стойкости обфускации, исследователям будет легче понять, какие требования безопасности обеспечивают те или иные обфускирующие преобразования и оценить, насколько эти преобразования удовлетворяют заявленным целям. Многообразие новых формальных определений стойкости может прояснить решение поставленных задач. Наконец, введение новых формальных требований стойкости обфускации программ откроет новые возможности адаптации формальных методов теоретической информатики к решению задач защиты программ.

Статьи [1] и [2] положили начало двум направлениям исследований задачи обфускации программ, а именно, обфускации программ как средства решения некоторых криптографических задач и обфускации программ как одного из методов обеспечения компьютерной безопасности. Мы уделим главное внимание первому направлению исследований. Однако для полноты картины в следующем разделе статьи мы коротко опишем основные достижения в решении задачи обфускации программ и некоторых смежных с ней задач в области системного программирования. Далее, в разделе 3 мы рассмотрим наиболее известные формальные определения стойкости обфускации программ и взаимосвязь между этими определениями.

2. Обфускация программ с позиции системного программирования

Программная обфускация могла бы быть очень полезной для решения многих задач системного программирования и компьютерной безопасности. Уже в

ранних работах, посвященных обфускации программ, было показано, что обфускирующие преобразования могут быть использованы для защиты интеллектуальной собственности в качестве средства, препятствующего восстановлению исходных алгоритмов на основе открытого программного кода [2,25-27] и удалению из программ водяных знаков (watermarkings) и «отпечатков пальцев» (fingerprintings) [6,8,28-31], для защиты программного обеспечения от атак со стороны вредоносных программ (компьютерных вирусов) и обеспечения безопасности мобильных агентов в информационных сетях [32-36], для проведения безопасного поиска в потоках данных [37], защиты баз данных [38], защиты проектных решений при проектировании микроэлектронных схем [39,40]. Оборотной стороной полезных достоинств обфускации является возможность ее использования для затруднения обнаружения вредоносных программ [4,5,41], а также создания уязвимостей в системах защиты компьютеров [42].

Впервые простейшие приемы обфускации программ были перечислены и систематизированы в работах [2,43]. Авторы этих работ пришли к ряду выводов, которые в дальнейшем определили несколько основных линий исследования задачи обфускации программ методами и средствами системного программирования и теории анализа программ.

- Целью обфускации программ является противодействие алгоритмам статического и динамического анализа программ. Поэтому качество обфускации можно оценивать экспериментально в зависимости от того, насколько результативными оказывается применение программно-инструментальных средств анализа программ к программам, подвергшимся обфускации. Таким образом, разработку средств обфускации программ целесообразно проводить в тесной взаимосвязи с совершенствованием средств деобфускации программ по принципу состязания щита и меча. Развитие этого состязания можно проследить в серии работ [44-60] и нашло воплощение в ежегодно проводимом конкурсе обфускированных программ.
- Для предварительной теоретической оценки качества обфускации можно использовать метрические характеристики, описывающие сложность устройства программ: обфускация программы должна приводить к значительному увеличению этих характеристик. Анализ качества обфускирующих преобразований на основании метрик сложности программ проводился в работах [61-65].
- Поскольку целью обфускации является противодействие алгоритмам статического анализа, то стойкость обфускирующих преобразований можно оценивать также относительно сложности тех моделей программ, которые используются в алгоритмах статического анализа, направленных на выявление и устранение последствий маскировки программного кода: обфускация считается тем более стойкой, чем более сложные модели программ необходимы для проведения

деобфускирующего статического анализа программ. Исследования стойкости обфускирующих преобразований относительно методов статического анализа было проведено в работах [66-73]. Результаты этих исследований, в частности привели к созданию специализированных алгоритмов для обнаружения полиморфных компьютерных вирусов, использующих процедуры обфускации в процессе репликации [74-76].

- В статье [2] было отмечено, что ключевым элементом многих приемов обфускации программ являются т.н. непроницаемые предикаты (opaque predicates) – предикаты, постусловия которых трудно вычислить посредством алгоритмов статического анализа. В нескольких работах [29,31,77-79] было проведено изучение способов построения и оценки стойкости непроницаемых предикатов. Авторы статьи [2] также обратили внимание на то, что для повышения стойкости обфускирующих преобразований целесообразно использовать вычислительно трудные комбинаторные задачи, конструируя на их основе непроницаемые предикаты таким образом, чтобы раскрытие поведения такого предиката было бы равносильно решению указанной задачи. Этот прием был использован в работах [25,80-83] для демонстрации того, что в некоторых случаях задача деобфускации программ может оказаться вычислительно трудной.
- Задачу обфускации программ можно решать и в такой постановке, когда лишь некоторая часть программы доступна противнику для анализа и модификации, в то время как все остальные компоненты программы выполняются на защищенном вычислительном устройстве. Впервые в такой постановке задача защиты программ была рассмотрена в статье [84]; в этой статье рассматривалась модель вычислений, состоящая из защищенного устройства памяти и общедоступного процессора. Успешное решение этой задачи привело к разработке некоторых способов обфускации программ, опирающихся на аппаратную поддержку [85,86].

Результаты проведенных теоретических и экспериментальных исследований задачи обфускации программ воплотились в нескольких десятках программно-инструментальных средств защиты программ путем применения обфускирующих преобразований того или иного вида. Большая часть предложенных подходов представляют собой эвристики (некоторые из которых весьма изощренны), предназначенные для усложнения программ анализа алгоритмов. Общим их недостатком является отсутствие обоснования гарантированной стойкости. Некоторые из этих средств обфускации довольно успешно противостоят автоматическим инструментам статического анализа и декомпиляции программных кодов. Однако в случае применения методов динамического анализа программ и привлечения квалифицированных экспертов в области системного программирования стойкости существующих

средств обфускации программ оказывается уже недостаточно. Таким образом, разработанные к настоящему времени практические методы и алгоритмы обфускации программ способны затруднить (порой, весьма значительно) понимание и модификацию программ, но не могут рассматриваться в качестве средств защиты секретной информации, содержащейся в программном коде, подобных системам шифрования.

3. Обфускация программ с позиции математической криптографии

Обфускация программ имеет важные приложения в области криптографии. Уже в ранних работах [1,13] было отмечено, что обфускация программ позволяет преобразовывать криптосистемы с секретным ключом в криптосистемы с открытым ключом: в качестве открытого ключа выступает обфускированная процедура шифрования с вставленным в нее секретным ключом. При помощи обфускации программ можно также конструировать гомоморфные системы шифрования [13], функциональные системы шифрования [87], доверенные схемы перешифрования [23] и электронно-цифровой подписи [88], избавляться от модели случайного оракула при доказательстве стойкости криптографических протоколов [2], осуществлять случайную перетасовку зашифрованных сообщений в схемах тайного голосования [89], создавать схемы дезавуируемого (двусмысленного) шифрования [90,91] и односторонние функции с секретом [91]. Более подробно об этих приложениях обфускации программ в следующей статье нашего цикла. Однако для того, чтобы каждое из перечисленных приложений обладало определенной криптографической стойкостью, используемая для его построения обфускация программ также должна удовлетворять некоторым требованиям стойкости. Поэтому при исследовании проблемы обфускации программ с позиции математической криптографии требование стойкости выдвигается на первый план. В этом разделе статьи мы рассмотрим различные определения стойкости обфускации, известные в современной математической литературе и приведем некоторые основные результаты, обосновывающие возможность или невозможность построения стойких обфускирующих преобразований.

3.1. Обфускация программ в модели виртуального «черного ящика»

Впервые строгое определение обфускации программ было сформулировано в статье [13], авторы которой опирались на результаты более ранней статьи [92]. К обфускации программ предъявляются три главных требования – сохранение функциональности программы, незначительное изменение ее размера и быстродействия и требование стойкости. Обфускация программ считается стойкой в модели виртуального «черного ящика», если противник, имеющий неограниченный доступ к тексту обфускированной программы, может извлечь

из этого текста только ту информацию об исходной программе, которую можно было бы получить, проводя одни лишь тестовые эксперименты с программой без доступа к ее тексту. Для строгого математического определения обфускации программ в модели виртуального «черного ящика» нужно ввести ряд вспомогательных понятий.

В литературе по теории сложности и математической криптографии используются две общепринятые формализации понятия «вычислительная программа». Согласно одному из них программы представляются в виде машин Тьюринга, а согласно второму программы представляются в виде схем из функциональных элементов (логических схем). В тех случаях, когда сложность вычислений и размеры программ оцениваются с точностью до полиномиальных преобразований, эти два определения равноправны (см. [93]). Для единообразия мы ограничимся рассмотрением представления программ в виде машин Тьюринга. В криптографии в качестве модели противника чаще всего используются вероятностные алгоритмы, подразумевающие возможность проведения вычислений с использованием случайных величин (датчиков случайных чисел, случайных двоичных строк и т.п.). При этом вычисления противника должны завершаться за время, полиномиально зависящее от размера тех данных, которыми он располагает. Таким образом, в качестве модели противника также используются машины Тьюринга, но при этом вероятностные и ограниченные полиномиальным временем. Для обозначения этого класса машин используется аббревиатура PPT. В теории сложности вычислений наряду с обычными и вероятностными машинами Тьюринга также используются машины Тьюринга с оракулом. В качестве оракула могут выступать любые функции или предикаты, которые рассматриваются как вспомогательные средства вычисления. ТМ с оракулом F может обратиться к оракулу для вычисления значения функции $F(y)$ для произвольной строки y, записанной на специальной ленте оракула. Это значение вычисляется за один шаг независимо от сложности функции F.

Для машины Тьюринга (ТМ) π и двоичной строки x, используемой в качестве входных данных для ТМ, условимся использовать запись $|\pi|$ для обозначения размера ТМ π, запись $\pi(x)$ для обозначения результата вычисления ТМ π на входных данных x, и запись $time(\pi(x))$ для обозначения времени (числа шагов) вычисления ТМ π на входных данных x. ТМ π_1 и π_2 считаются эквивалентными, если $\pi_1(x) = \pi_2(x)$ для любых входных данных x. Функция $\nu : N \to [0,1]$ считается пренебрежимо малой, если она убывает быстрее, чем любая функция, обратная многочлену, т.е. для любого $k \in N$ существует такое $n_0 \in N$, что для всех $n > n_0$

выполняется неравенство $\nu(n) < 1/n^k$. Для обозначения произвольного полинома и произвольной пренебрежимо малой функции традиционно используются обозначения $poly(\cdot)$ и $neg(\cdot)$ соответственно.

Определение 1 (модель виртуального «черного ящика») [13]

Обфускатором в модели виртуального «черного ящика» для семейства ТМ M называется такая РРТ O, которая удовлетворяет следующим трем условиям:

1. **Функциональная эквивалентность.** Для любой ТМ $\pi, \pi \in M,$ в результате применения обфускатора O к π строится ТМ, эквивалентная исходной машине π.
2. **Полиномиальные издержки.** Для любой ТМ $\pi, \pi \in M,$ размер и быстродействие любой ТМ $O(\pi)$ отличается от размера и быстродействия ТМ π не более чем полиномиально, т.е. $|O(\pi)| = poly(|\pi|)$ и $time(O(\pi(x))) = poly(time(\pi(x)))$.
3. **Свойство виртуального «черного ящика».** Для любой РРТ A (противника) существует такая РРТ S (симулятор), для которой соотношение

$$|\Pr[A(O(\pi)) = 1] - \Pr[S^{\pi}(1^{|\pi|}) = 1]| \le neg(|\pi|)$$

выполняется для любой ТМ $\pi, \pi \in M,$ причем первая вероятность вычисляется для случайных величин, используемых обфускатором O и противником A, а вторая вероятность вычисляется для случайных величин, используемых симулятором S.

Один из основных результатов работы [13] таков.

Теорема 1. [13]. Существуют такие семейства ТМ, для которых нельзя построить обфускатор в модели виртуального «черного ящика».

Идея доказательства этой теоремы проста. Рассмотрим пару ТМ $\pi_{a,b}$ и $\pi_{c,d}$. Первая из них для каждой входной строки x вычисляет на выходе строку b, если $x = 1$, и выдает на выходе нулевую строку в противном случае. Вторая из этих ТМ рассматривает входную строку x как описание некоторой программы и применяет эту программу к строке c и выдает на выходе 1, если это вычисление завершается за полиномиальное время (относительно длины x) с результатом d; в противном случае ТМ $\pi_{c,d}$ выдает на выходе 0. Из этих двух ТМ можно сформировать ТМ $\pi_{a,b,c,d}(x, y)$, которая в случае $y = 0$ проводит вычисление так же, как

ТМ $\pi_{a,b}(x)$, а в противном случае проводит вычисление так же, как ТМ $\pi_{c,d}(x)$. Целью (угрозой) противника является выяснение для заданной ТМ $\pi_{a,b,c,d}(x,y)$, верно ли, что $a=c$ и $b=d$. Этой цели легко достигает детерминированная ТМ A, которая, получив на входе обфускированную программу $\pi'=O(\pi_{a,b,c,d})$, подставляет в эту программу в качестве второй компоненты входных данных 0 и 1, а затем вычисляет значение $\pi'(\pi'(\cdot,0),1)$. Очевидно, что равенства $a=c$ и $b=d$ выполняются тогда и только тогда, когда указанное значение, вычисленное противником A, равно 1. В то же время всякий симулятор S, завершающий работу за полиномиальное время и имеющий лишь оракульный доступ к программе $\pi_{a,b,c,d}$, может правильно проверить выполнимость условий $a=c$ и $b=d$ лишь за счет случайного угадывания подходящих запросов к оракулу, т.е. с пренебрежимо малой вероятностью.

Авторы работы [13] смогли построить *наследственно необфускируемое семейство функций,* т.е. такое множество функций, что всякая программа вычисления любой из функций этого множества не допускает обфускации в модели виртуального «черного ящика». Интересно отметить, что в класс наследственно необфускируемых семейств функций включаются некоторые семейства псевдослучайных функций, функций шифрования с секретным ключом, схемы электронно-цифровой подписи. Кроме того, в статье [13] было показано, что обфускация в модели виртуального «черного ящика» невозможна для некоторых семейств программ, представимых логическими схемами из класса TC_0. Схемы этого класса строятся из пороговых элементов, и глубина схем ограничена некоторой константой.

В целом, результаты статьи [13] показывают, что задача обфускации программ не имеет простого решения: существуют такие семейства программ, вычисляющие сравнительно простые функции, для которых нельзя построить стойкую обфускацию в модели виртуального «черного ящика». В связи с этим дальнейшие исследования проблемы обфускации были сосредоточены на получении ответов на следующие вопросы. Существуют ли другие определения стойкости обфускации, которые являются менее требовательными, нежели стойкость в модели виртуального «черного ящика», но при этом позволяют находить для обфускации подходящие криптографические приложения? Для каких классов программ можно построить обфускаторы, удовлетворяющие тем или иным разумным требованиям стойкости? Каковы необходимые и достаточные условия необфускируемости программ?

3.2. Вариации модели виртуального «черного ящика»

Уже из самого определения обфускации в модели виртуального «черного ящика» видно, что оно допускает некоторые вариации за счет изменения ограничений, налагаемых на противника A и симулятор S.

Например, можно предполагать, что вычислительные возможности противника неограниченны. Тогда удается полностью описать класс программ, допускающих обфускацию. Семейство программ M называется эффективно выведываемой (learnable) [94], если существует такая PPT L, что для любой программы $\pi, \pi \in M,$ ТМ L^{π} выдает на выходе программу π', эквивалентную программе π. В статье [20] было установлено, что семейство программ допускает обфускацию в модели виртуального «черного ящика» с неограниченным противником в том и только том случае, когда это семейство программ эффективно выведываемо.

Можно рассмотреть такой вариант определения 1, в котором вычислительные возможности симулятора неограничены, но при этом число обращений симулятора к оракулу ограничено величиной, полиномиально зависящей от размера обфускируемой программы π. Такая разновидность обфускации была введена в статье [95]; будем называть ее обфускацией со слабо ограниченным симулятором. В этой же работе было показано, что требования стойкости обфускации в модели со слабо ограниченным симулятором существенно слабее требования стойкости обфускации в модели виртуального «черного ящика».

Теорема 2. [95]. Если существуют односторонние перестановки, то существует такое семейство ТМ, которое допускает обфускацию в модели со слабо ограниченным симулятором, но не допускает обфускации в модели виртуального «черного ящика».

Доказательства этой теоремы основывается на той же идее, что и доказательство теоремы 1.

Авторам статьи [95] удалось также показать существование таких семейств ТМ, для которых невозможно построить обфускации в модели со слабо ограниченным симулятором.

Возможности симулятора можно усилить и другим способом. Поскольку противник, имеющий доступ к обфускированной программе, может наблюдать не только пары вход-выход, но отслеживать трассы вычислений, целесообразно проверить, нельзя ли добиться того, чтобы в результате преобразования обфускации программы, ее вычислительные трассы составляли бы единственную полезную информацию, которую может извлечь противник проводя эксперименты с программой.

Для достижения этой цели модифицируем определение обфускации в модели виртуального «черного ящика». Обфускация нового вида, введенная и

исследованная в статье [14], была названа обфускацией в модели виртуального «серого ящика». Модель виртуального «серого ящика» отличается от модели виртуального «черного ящика» в двух аспектах.

Во-первых, оракул симулятора в ответ на запрос x выдает не только результат вычисления обфускируемой программы $\pi(x)$, но также и трассу вычисления программы π для входных данных x. Таким образом, здесь важно знать, какая именно из эквивалентных программ выбрана в качестве оракула. Мы требуем, чтобы такой программой была исходная программа π. Это означает, что обфускатор не обязан скрывать ни одно из тех свойств программы π, которые можно эффективно установить на основе трасс вычисления ТМ π.

Во-вторых, вместо завершающихся программ мы рассматриваем реагирующие программы. В отличие от завершающихся программ, предназначенных для вычисления отношений между входными и выходными данными, реагирующие программы осуществляют преобразования потоков событий-запросов, поступающих на вход программы, в поток откликов-реакций на эти события, вырабатываемых на выходе программы. Таким образом, реагирующая программа вычисляет функцию, отображающую бесконечную последовательностей входных данных $x_1, x_2, \ldots, x_n, \ldots$ (последовательность запросов) в бесконечную последовательность выходов $y_1, y_2, \ldots, y_n, \cdots$ (последовательность откликов), так что каждый выход y_n зависит только от входов $x_1, x_2, \ldots, x_n$. Примерами реагирующих программ могут служить сетевые протоколы (в том числе криптографические), встроенные системы, операционные системы и т.п. Реагирующую программу можно определить формально как обычную ТМ, которая использует входную и выходную ленты, а также некоторое количество вспомогательных лент так, что чтение очередного входного слова x_{n+1} не осуществляется до тех пор, пока не будет полностью завершена запись выходного слова y_n. Для обозначения таких машин будем использовать аббревиатуру RTM, чтобы отличить их от обычных ТМ.

Чтобы отличать оракул из определения обфускатора в модели виртуального «черного ящика» (Определение 1) от оракула, используемого в новом определении, условимся обозначать последний записью $Tr(\pi)$. В ответ на каждый очередной запрос x_n оракул $Tr(\pi)$ выдает пару $\langle y_n, tr(x_n)\rangle$, где y_n - результат работы машины π на входе x_n (отметим, что y_n зависит не только от x_n, но и на всех предыдущих входов, которые ранее

использовались в качестве запросов к оракулу) и $tr(x_n)$ - трасса выполнения π на этом входе.

Определение 2 (модель виртуального «серого ящика») [14]

Обфускатором в модели виртуального «серого ящика» для семейства RTM M называется такая PPT O, которая удовлетворяет условиям функциональной эквивалентности и полиномиальных издержек, а также следующему требованию стойкости:

Свойство виртуального «серого ящика». Для любой PPT A (противника) существует такая PPT S (симулятор), для которой соотношение

$$|\Pr[A(O(\pi))=1]-\Pr[S^{Tr(\pi)}(1^{|\pi|})=1]|\le neg(|\pi|)$$

выполняется для любой RTM $\pi, \pi \in M$, причем первая вероятность вычисляется для случайных величин, используемых обфускатором O и противником A, а вторая вероятность вычисляется для случайных величин, используемых симулятором S.

Теорема 3. [14]. Если существуют односторонние функции, то существуют такие семейства RTM, обфускация которых в модели виртуального «серого ящика» невозможна.

Еще один вариант определения обфускации, близкий к модели виртуального «черного ящика», был предложен в статье [96]. На практике противник может проводить анализ обфускированной программы, располагая некоторыми дополнительными данными, которые могут иметь или не иметь отношение к этой программе. Например, противник может располагать наряду с обфускированной программой π также и демонстрационной упрощенной версией этой программы π'. Тогда модель виртуального «черного ящика» может быть соответствующим образом модифицирована.

Определение 3 (модель виртуального «черного ящика» с дополнительным входом) [96]

Обфускатором в модели виртуального «черного ящика» с дополнительным входом для семейства TM M называется PPT O, удовлетворяющая условиям функциональной эквивалентности и полиномиальных издержек, а также следующему условию:

Свойство виртуального «черного ящика» с дополнительным входом. Для любой PPT A (противника) существует такая PPT S (симулятор), для которой соотношение

$$|\Pr[A(O(\pi),z)=1]-\Pr[S^{Tr(\pi)}(1^{|\pi|},z)=1]|\le neg(|\pi|)$$

выполняется для любой ТМ $\pi, \pi \in M,$ и двоичной строки (дополнительного входа) Z, причем первая вероятность вычисляется для случайных величин, используемых обфускатором O и противником A, а вторая вероятность вычисляется для случайных величин, используемых симулятором S.

В этом определении допускается возможность того, что дополнительные входные данные Z могут зависеть или не зависеть от обфускируемой программы. Поэтому, есть два варианта обфускации с дополнительным входом, и для обоих была установлена их взаимосвязь с введенными выше разновидностями обфускации.

Теорема 4. [95,96]. Если для семейства ТМ M существуют обфускатор в модели виртуального «черного ящика», то для этого семейства ТМ существует обфускатор с дополнительным входом, не зависящим от обфускируемой программы. Если для семейства ТМ M существуют обфускатор в модели виртуального «черного ящика» со слабо ограниченным симулятором, то для этого семейства ТМ существует обфускатор с дополнительным входом, зависящим от обфускируемой программы.

В статье [96] был также описан класс логических схем – схемы со сверхполиномиальной энтропией, вычисляющие функции с NP-полным фильтром, – для которых невозможно построить обфускаторов с зависимым или независимым дополнительным входом. Авторы упомянутой статьи, в частности, показали, что класс схем со сверхполиномиальной энтропией включает в себя всякое семейство логических схем, вычисляющих псевдослучайные функции, а также всякое семейство логических схем, реализующих криптосистемы, в которых используются псевдослучайные функции.

Полученные результаты показывают, что простые ослабления требования стойкости обфускации в модели виртуального «черного ящика» не влекут за собой существенного упрощения задачи обфускации программ. Кроме того, выяснилось, что для многих криптографических функций никакая программа их вычисления не может быть полностью обфускирована. Поэтому целесообразно рассмотреть другие виды обфускации программ, менее требовательные, нежели обфускации в модели виртуального «черного ящика».

3.3. Обфускация алгоритмов

Наиболее очевидное применение обфускации для защиты программных продуктов предлагает следующий сценарий. Предположим, что некто изобрел быстрый алгоритм решения сложной и практически важной задачи. Целью обфускации в этом случае, является такое преобразование программы, реализующий этот алгоритм, которое не позволяет извлечь его с легкостью из обфускированной программы. Обфускация в модели виртуального «черного

ящика» здесь непригодна: вряд ли найдется клиент, готовый купить «черный ящик», как часть программного обеспечения. Напротив, любой программный продукт на рынке должен иметь руководство пользователя, с указанием свойств функциональности. Если известна функция, вычисляемая программой, то цель обфускации в этом случае состоит в том, чтобы воспрепятствовать пониманию оригинальных особенностей алгоритма, на основе которого простроена данная программа. Такая разновидность обфускации программ более соответствует целям защиты алгоритмов, нежели тотальная обфускация в модели виртуального «черного ящика». Самый простой способ формализации понятия обфускации алгоритмов состоит в том, чтобы предоставить противника в качестве дополнительной информации некоторую (произвольную) программу π_0, вычисляющую ту же самую функцию, что и обфускируемая программа π. Таким образом, мы приходим к нескольким определениям обфускатора алгоритмов, которые были предложены в работах [13,14,16].

Определение 4 (обфускация алгоритмов) [14]

Обфускатором алгоритмов для семейства ТМ M называется такая PPT O, которая удовлетворяет условиям функциональной эквивалентности и полиномиальных издержек, а также следующему требованию стойкости:

Свойство сокрытия алгоритма. Для любой PPT A (противника) существует такая PPT S (симулятор), для которой соотношение

$$|\Pr[A(O(\pi),\pi_0)=1]-\Pr[S(1^{|\pi|},\pi_0)=1]|\le neg(|\pi|)$$

выполняется для любой пары эквивалентных ТМ π, π_0 из класса M, удовлетворяющей условию $|\pi_0|=poly(|\pi|)$, причем первая вероятность вычисляется для случайных величин, используемых обфускатором O и противником A, а вторая вероятность вычисляется для случайных величин, используемых симулятором S.

Помимо указанного выше определения обфускатора алгоритмов в статьях [13,16] для сокрытия алгоритмов были предложены два сходных альтернативных определения обфускации, основанные на концепции вычислительно неотличимых распределений случайных величин – неотличимая обфускация и наилучшая обфускация программ.

Определение 5 (неотличимая обфускация) [13]

Неотличимым обфускатором для семейства ТМ M называется такая PPT O, которая удовлетворяет условиям функциональной эквивалентности и полиномиальных издержек, а также следующему условию:

Свойство эффективной неотличимости программ. Для любой пары эквивалентных ТМ π_1, π_2 из класса M, имеющих одинаковый размер, распределения вероятностей случайных величин $O(\pi_1)$ и $O(\pi_2)$ вычислительно неотличимы, т.е. для любой PPT D справедливо соотношение $|\Pr[D(O(\pi_1))=1]-\Pr[D(O(\pi_2))=1]|\leq neg(|\pi|)$

Нетрудно видеть, что свойство неотличимости программ следует из свойства виртуального «черного ящика», однако тот прием, который применялся в теореме 1 для доказательства невозможности построения обфускатора в модели виртуального «черного ящика», уже непригоден в случае неотличимой обфускации. Недостаток определения 5 состоит в том, что оно не дает интуитивно понятных гарантий того, что обфускированная программа действительно маскирует реализуемый алгоритм. Поэтому в статье [16] было предложено альтернативное определение, в основу которого также положена идея о неотличимости обфускированных программ.

Определение 6 (наилучшая обфускация) [16]

Наилучшим обфускатором для семейства ТМ M называется такая PPT O, которая удовлетворяет условиям функциональной эквивалентности и полиномиальных издержек, а также следующему условию:

Свойство ограниченной эффективной выведываемости. Для любой PPT L (выведыватель) существует такая PPT S (симулятор), что для любой пары эквивалентных ТМ π_1, π_2 из класса M, имеющих одинаковый размер, распределения вероятностей случайных величин $L(O(\pi_1))$ и $S(\pi_2)$ вычислительно неотличимы.

Свойство ограниченной выведываемости подразумевает, что противник может извлечь из текста обфускированной программы лишь такую полезную информацию, которую можно было бы эффективно вычислить, имея текст любой эквивалентной программы.

В определениях 5 и 6 условия вычислительной неотличимости распределений вероятностей можно усилить, потребовав, чтобы рассматриваемые распределения вероятностей совпадали (абсолютная неотличимость) или чтобы статистическое расстояние между ними не превосходило некоторой заранее заданной константы (статистическая неотличимость). И хотя взаимосвязь определений обфускации 1, 5 и 6 неочевидна, справедлива следующая

Теорема 5. [16,95]. Следующие три вида обфускации программ равносильны:

1. неотличимая обфускация,
2. наилучшая обфускация,

3. обфускация в модели виртуального «черного ящика» с симулятором, имеющим неограниченные вычислительные возможности.

Из этой теоремы видно, что обфускация всякого класса программ в модели виртуального «черного ящика» влечет за собой и наилучшую обфускацию того же класса программ. Поэтому требование эффективности симулятора S в определении 6 является избыточным. Кроме того, обфускация алгоритмов для любого класса программ влечет наилучшую обфускацию того же класса программ. Однако вопрос о том, верно ли обратное включение, остается открытым. Открыт и вопрос о существовании семейств программ, для которых не существует наилучшей обфускации. Однако в статье [16] была доказана следующая теорема

Теорема 6. [16]. Если семейство программ, представленных в виде 3-КНФ, имеет наилучшую (в смысле статистической неотличимости) обфускацию, то коллапс полиномиальной иерархии классов сложности происходит на втором уровне.

В настоящее время вопрос об устройстве полиномиальной иерархии остается открытым, хотя большинство специалистов в области теории вычислений придерживаются мнения о том, что она имеет бесконечно много уровней.

Важным частным случаем обфускации алгоритмов является обфускация констант. Предположим, что имеется параметризованное семейство ТМ $M(C)=\{\pi(c):c\in C\}$, которые отличаются друг от друга только значением одного секретного параметра, используемого в качестве константы. Таким параметром может быть, например, секретный ключ в процедуре шифрования или электронно-цифровой подписи. Алгоритм, реализуемый программами семейства $M(C)$, не составляет секрета и не нуждается в маскировке.

В публикациях, посвященных проблеме обфускации программ, предложено несколько определений параметризованных программ. Одно из них является простой адаптацией определения обфускатора в модели виртуального «черного ящика».

Определение 7 (обфускация констант в модели виртуального «черного ящика») [14,21]

Обфускатором констант для семейства ТМ $\mathrm{M}(\mathrm{C})$ называется такая PPT O, которая удовлетворяет условиям функциональной эквивалентности и полиномиальных издержек, а также следующему требованию стойкости:

Свойство сокрытия констант. Для любой PPT A (противника) существует такая PPT S (симулятор), для которой соотношение

$$|\Pr[A(O(\pi(c_0)),\pi(c))=1]-\Pr[S^{\pi(c_0)}(1^{|\pi(c_0)|},\pi(c))=1]|\le neg(|\pi(c_0)|)$$

выполняется для любой пары ТМ $\pi(c),\pi(c_0)$ из класса $\mathrm{M}(\mathrm{C})$,

удовлетворяющей условию $|\pi(c_0)| = poly(|\pi(c)|)$, причем первая вероятность вычисляется для случайных величин, используемых обфускатором O и противником A, а вторая вероятность вычисляется для случайных величин, используемых симулятором S.

Пример программ, для которых невозможно построить обфускацию в модели виртуального «черного ящика», свидетельствует также и том, что для некоторых семейств параметризованных программ невозможно провести обфускацию констант.

Другое определение является переложением определения неотличимого обфускатор. Пусть на множестве констант C задано некоторое распределение вероятностей.

Определение 8 (неотличимая обфускация констант) [21]

Обфускатором констант для семейства ТМ $M(C)$ называется такая пара РРТ (G, O), которая удовлетворяет следующим трем условиям:

1. **Функциональная эквивалентность.** Для любой константы $c, c \in C,$ выходом $G(c)$ является константа $d, d \in C,$ а выходом $O(G(c), c)$ является ТМ π', такие, что ТМ $\pi(c)$ эквивалентна ТМ $\pi'(d)$.
2. **Полиномиальные издержки.** Для любой константы $c, c \in C,$ размер любой константы $G(c)$ отличается от размера константы c не более чем полиномиально, и размер и время выполнения любой ТМ $O(\pi)$ отличается от размера ТМ π не более чем полиномиально.
3. **Свойство неотличимости констант.** Для любой РРТ D (распознавателя) существует такая РРТ S (симулятор), для которой соотношение

$$|\Pr[D^{\pi(c)}(1^{|c|}, G(c)) = 1] - \Pr[D^{\pi(c)}(1^{|c|}, S^{\pi(c)}(1^{|c|})) = 1]| \le neg(|c|)$$

где константа c выбирается согласно указанному выше распределению вероятностей, и обе вероятности, фигурирующие в левой части соотношения вычисляется для случайных величин, используемых обфускатором O, распознавателем D и симулятором S.

Здесь обфускации подвергается сама константа, а программа, использующая ее, лишь модифицируется, чтобы удовлетворять условию функциональной эквивалентности. Это определение обфускации констант специально введено для использования его в решении задач преобразования криптосистем с секретным ключом в криптосистемы с открытым ключом. Однако для определения 8 существуют программы, которые нельзя обфускировать.

Теорема 7. [21]. Семейство программ $M(C)$, реализующих псевдослучайную функцию невозможно обфускировать в соответствие с определением 8.

3.4. Обфускация предикатов

Спектр формальных определений обфускации программ замыкает обфускация предикатов. Она предназначена для сокрытия какого-либо выделенного свойства программ. Предположим, что на множестве ТМ M задан предикат P.

Определение 9 (обфускация предикатов) [14]

Обфускатором предиката P для семейства ТМ M называется такая PPT O, которая удовлетворяет условиям функциональной эквивалентности и полиномиальных издержек, а также следующему требованию стойкости:

Свойство сокрытия предиката. Для любой PPT A (противника) существует такая PPT S (симулятор), для которой соотношение

$$|\Pr[A(O(\pi)) = P(\pi)] - \Pr[S^{\pi}(1^{|\pi(c_0)|}) = P(\pi)]| \leq neg(|\pi|)$$

выполняется для любой ТМ π из класса M, причем первая вероятность вычисляется для случайных величин, используемых обфускатором O и противником A, а вторая вероятность вычисляется для случайных величин, используемых симулятором S.

Сопоставив определения 1 и 8, можно заметить, что обфускация программ в модели виртуального «черного ящика» должна скрывать все возможные предикаты на заданном семействе программ, в то время как определение 8 допускает построение различных обфускаторов для разных предикатов. Существование необфускируемых предикатов для некоторых семейств программ следует из теоремы 1.

4. Заключение

Как видно из приведенной в этом разделе онтологии определений обфускации программ, обфускаторы могут быть условно разделены на два класса – «сильные» обфускаторы (определения 1-3), призванные маскировать любые невыведываемые свойства программ, и «слабые» обфускаторы (определения 4-8), предназначенные для сокрытия лишь отдельных свойств программ. Для каждого определения обфускации доказано существование таких представительных классов функций (включающих функции, имеющие криптографические приложения), которые не могут быть реализованы обфускируемыми программами. Математическая техника, развитая в работах [13,14,16,20-22,96] позволяет, по-видимому, построить примеры такого рода для любого «разумного» определения обфускации. Гораздо более трудно

доказать, что некоторые практически значимые программы и функции могут быть обфускированы. Обзор наиболее интересных результатов в этом направлении исследований приведен в разделе 5. Но прежде чем обратиться к эти результатам, в какой мере обфускация программ может быть полезна для решения тех или иных задач криптографии и компьютерной безопасности.

Список литературы

[1]. Diffie W., Hellman M. New directions in cryptography // *IEEE Transactions on Information Theory*, IT-22(6), 1976, p.644-654.

[2]. Collberg C., Thomborson C., Low D. A Taxonomy of Obfuscating Transformations // Technical Report, N 148, Univ. of Auckland, 1997.

[3]. Cohen F. Operating system protection through program evolution // Computers and Security, v. 12, N 6, 1993, p. 565-584.

[4]. Chess D., White S. An undetectable computer virus // *Proceedings of the 2000 Virus Bulletin Conference*, 2000.

[5]. Szor P., Ferrie P. Hunting for metamorphic // *Proceedings of the 2001 Virus Bulletin Conference*, 2001, p.123-144.

[6]. Collberg C., J. Nagra. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Program Protection. Addison-Wesley Professional, 2009.

[7]. Aucsmith D. Tamper resistant software: an implementation // *Information Hiding Conference*, Lecture Notes in Computer Science, v. 1174, 1996, p. 317-333.

[8]. Scud T.T. ObjObf - x86/Linux ELF relocateable object obfuscator, 2003. http://packetstormsecurity.org/files/31524/objobf-0.5.0.tar. bz2.

[9]. Solutions P. DashO - the premier Java obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dasho/.

[10]. Solutions P. Dotfuscator - the premier .NET obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dotfuscator/.

[11]. Z. KlassMaster. The second generation Java obfuscator. http://www.zelix.com/.

[12]. Ge J., Chaudhuri S., Tyagi A. Control Flow Based Obfuscation // *Proceedings of the Digital Rights Management Workshop*. Alexandria, VA, USA, 2005, p. 83-92

[13]. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Ke Yang. On the (im)possibility of obfuscating programs // *Advances in Cryptology - CRYPTO'01*, Lecture Notes in Computer Science, v. 2139, 2001, p. 1-18 (см. также Journal of the ACM 2012).

[14]. Varnovsky N.P. A note on the concept of obfuscation // Proceedings of Institute for System Programming, Moscow, N 6, 2004, p. 127-137.

[15]. Kuzurin N.N., Shokurov A.V., Varnovsky N.P., Zakharov V.A. On the concept of software obfuscation in computer security // *Information Security Conference*, Lecture Notes in Computer Science, v. 4779, 2007, p. 281-298.

[16]. Goldwasser S., Rothblum G.N. On best possible obfuscation // *Theory of Cryptography Conference*, Lecture Notes in Computer Science, v. 4392, 2007, p. 194-213.

[17]. Canetti R. Towards realizing random oracles: hash functions that hide all partial information // *Advances in Cryptology --- CRYPTO'97*, Lecture Notes in Computer Science, v. 1294, 1997, p. 455-469.

[18]. Varnovsky N.P., Zakharov V.A. On the possibility of provably secure obfuscating programs // *Conference ``Perspectives of System Informatics''*, Lecture Notes in Computer Science, v. 2890, 2004, p. 91-102.

[19]. Lynn B., Prabhakaran M., Sahai A. Positive results and techniques for obfuscation // *Advances in Cryptology - EUROCRYPT 2004*, Lecture Notes in Computer Science, v. 3027, 2004, p. 20-39.
[20]. Wee H. On obfuscating point functions // *Proceedings of the 37-th Symposium on Theory of Computing*, 2005, p. 523-532.
[21]. Hofheinz D., Malone-Lee J., Stam M. Obfuscation for cryptographic purpose // *Theory of Cryptography Conference*, Lecture Notes in Computer Science, v. 4392, p. 214-232.
[22]. Canetti R., Dakdouk R. R. Obfuscating point functions with multibit output // *Advances in Cryptology – EUROCRYPT 2008*, Lecture Notes in Computer Science, 2008, v. 4965, p. 489–508.
[23]. Hohenberger S., Rothblum G.N., Shelat A., Vaikuntanathan V. Securely obfuscating re-encryption // *Proceedings of the 4-th Conference on Theory of Cryptography*, 2007, p. 233-252
[24]. Canetti R., Rothblum G.N., Varia M. Obfuscation of hyperplane membership // *Proceedings of the 7-th Conference on Theory of Cryptography*, 2010, p. 72-89.
[25]. Collberg C., Thomborson C., Low D. Manufacturing cheap, resilient and stealthy opaque constructs // *Proceedings of the Symposium on Principles of Programming Languages*, 1998, p. 184-196.
[26]. de Oor A., van der Oord L. Stealthy obfuscation techniques: misleading pirates // Technical Report of Department of Computer Science University of Twente Enschede, The Netherlands, 2003.
[27]. Naumovich G, Memon N. Preventing piracy, reverse engineering,and tampering // *IEEE Computer*, 2003, v. 36, N 7, p. 64-71.
[28]. Collberg C, Thomborson C., Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection // IEEE Transactions on Software Engineering, v. 28, N 6, 2002.
[29]. Arboit G. A method for watermarking Java programs via opaque predicates//Proceedings of the International Conference on Electronic Commerce Research (ICECR-5). Montreal, Canada, 2002: 1-8.
[30]. Zhu W., Thomborson C., Wang F.-Y. A survey of software watermarking // Lecture Notes in Computer Science, v.3495, 2005, p. 454-458.
[31]. Myles G, Collberg C. Software watermarking via opaque predicates: Implementation, analysis, and attacks. Electronic Commerce Research, 2006, 6(2): 155-171.
[32]. Sander T., Tchudin C.F. Protecting mobile agents against malicious hosts // *Mobile Agents and Security*, Lecture Notes in Computer Science, 1997, p. 44-60.
[33]. Hohl F. Time limited blackbox security: protecting mobile agents from malicious hosts // *Mobile Agents and Security*, Lecture Notes in Computer Science, v. 1419, 1998, p. 92-113.
[34]. D'Anna L., Matt B., Reisse A., Van Vleck T. , Schwab S., LeBlanc P. Self-Protecting Mobile Agents Obfuscation Report // Technical Report N 03-015, Network Associates Laboratories, June 2003.
[35]. Wu J., Zhang Y., Wang X. et al. A scheme for protecting mobile agents based on combining obfuscated control flow and time checking technology // *Proceedings of the Conference on Computational Intelligence and Security*. Harbin, Heilongjiang, China, 2007, p. 912-916
[36]. Roeder T., Schneider F.B. Proactive Obfuscation // *ACM Transactions on Computer Systems*, v. 28, N 2, 2010.
[37]. Ostrovsky R., Skeith W.E. Private searching on streaming data //*Advances in Cryptology - CRYPTO-2005*, Lecture Notes in Computer Science, v. 3621, 2005, p. 223-240.

[38]. Narayanan A., Shmatikov V. Obfuscated databases and group privacy // *Proceedings of the 12th ACM Conference on Computer and Communications Security*, 2005, p. 102-111.
[39]. Иванников В.П., Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В., Кононов А.Н., Калинин А.В. Методы информационной защиты проектных решений при изготовлении микроэлектронных схем // Известия Таганрогского радиотехнического университета, 2005, т. 4, с. 112-119.
[40]. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Чернов А.В., Шокуров А.В. Задачи и методы обеспечения информационной безопасности при производстве микроэлектронных схем // Труды Института системного программирования РАН, 2006, т. 11. с. 29-61.
[41]. Borello J.M., Me L. Code obfuscation technique for metamorphic viruses // *Journal of Computer Virology*, 2008, v. 4, p, 211-220.
[42]. Bhatkar S., Du Varney D.C., Sekar R. Efficient techniques for comprehensive protection from memory error exploits // *USENIX Security*, 2005.
[43]. Wroblewski G. General method of program code obfuscation // Draft, 2002, 84 p.
[44]. Linn C., Debray S. Obfuscation of executable code to improve resistance to static disassembly // *Proceedings of the 10-th ACM Conference on Computer and Communication Security*, 2003, p. 290-299.
[45]. Sosonkin M, Naumovich G, Memon N. Obfuscation of design intent in object-oriented applications // *Proceedings of the Digital Rights Management Workshop*. Washington, DC, USA, 2003, p. 142-153.
[46]. Collberg C., Myles G., Huntwort A. Sandmark – a tool for software protection research // IEEE Security and Privacy, 2003, v. 1, N 4, p. 40-49.
[47]. Heffner K., Collberg C. The obfuscation executive // *Information Security Conference*, Lecture Notes in Computer Science, 2004, v. 3225.
[48]. Chan J. T., Yang W. Advanced obfuscation techniques for Java bytecode. Journal of Systems and Software, 2004, v. 71, N 1-2, p. 1-10.
[49]. Cimato S., De S. A., Petrillo U. F. Overcoming the obfuscation of Java programs by identifier renaming. Journal of Systems and Software, 2005, v. 78, N 1, p. 60-72.
[50]. Madou M., Anckaert B., de Sutter B., de Bosschere K. Hybrid static-dynamic attacks against software protection mechanisms // *Proceedings of the 5th ACM workshop on Digital rights management*, 2005, p. 75-82.
[51]. Udupa S. K., Debray S. K., Madou M. Deobfuscation: Reverse engineering obfuscated code // *Proceedings of the 12-th Working Conference on Reverse Engineering*. Pittsburgh, PA, USA, 2005, p. 45-54.
[52]. Ge J., Chaudhuri S., Tyagi A. Control Flow Based Obfuscation // *Proceedings of the Digital Rights Management Workshop*. Alexandria, VA, USA, 2005, p. 83-92.
[53]. Chen K., Chen J. B. On instrumenting obfuscated java bytecode with aspects // *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*. Shanghai, China, 2006, p. 19-26.
[54]. Madou M., Anckaert B., de Sutter B., de Bosschere K., Cappaert J., Preenel B. On the effectiveness of source code transformations for binary obfuscation // *Proceedings of the International Conference on Software Engineering Research and Practice*, 2006, p.527-533.
[55]. Madou M., Anckaert B., Moseley P., Debray S., de Sutter B., de Bosschere K. Software protection through dynamic code mutation // *Proceedings of the 6-th international conference on Information Security Applications*, 2006, p. 194-206.

[56]. Drape S, Majumdar A, Thomborson C. Slicing aided design of obfuscating transforms // *Proceedings of the International Computing and Information Systems Conference (ICIS 2007)*. Melbourne, Australia, 2007, p. 1019-1024.
[57]. Majumdar A., Drape S., Thomborson C. Slicing obfuscations: Design, correctness, and evaluation // *Proceedings of the 2007 ACM Workshop on Digital Rights*. Alexandria, VA, USA, 2007, p. 70-81.
[58]. Batchelder M., Hendren L. Obfuscating Java: The most pain for the least gain // *Proceedings of the Compiler Construction*. Braga, Portugal, 2007, p. 96-110.
[59]. Ceccato M., Di. P. M., Nagra J. et al. Towards experimental evaluation of code obfuscation techniques // *Proceedings of the 4th ACM Workshop on Quality of Protection*. Alexandria, VA, USA, 2008, p. 39-46.
[60]. Darwish S.M., Guirguis S.K., Zalat M.S. Stealthy code obfuscation technique for software security // *Proceedings of the International Conference on Computer Engineering and Systems*, 2010, p. 93-99.
[61]. А.В. Чернов. Об одном методе маскировки программ // *Труды Института системного программирования РАН*, 2003, т. 4. с. 85-119.
[62]. Majumdar A., Drape S., Thomborson C. et al. Metrics-based evaluation of slicing obfuscations // *Proceedings of the 3rd International Symposium on Information Assurance and Security*. Manchester, United Kingdom, 2007, p. 472-477.
[63]. Naeem N. A., Batchelder M., Hendren L. Metrics for Measuring the Effectiveness of Decompilers and Obfuscators // *Proceedings of the 15th IEEE International Conference on Program*. Banff, Alberta, Canada, 2007, p. 253-258.
[64]. Anckaert B., Madou M., De S. B. et al. Program obfuscation: A quantitative approach // *Proceedings of the 2007 ACM Workshop on Quality of Protection*. Alexandria, VA, USA, 2007, p. 15-20.
[65]. Tsai H. Y., Huang Y. L., Wagner D. A graph approach to quantitative analysis of control-flow obfuscating // *IEEE Transactions on Information Forensics and Security*, 2009, v. 4, N 2, p. 257-267.
[66]. Cousot P., Cousot R. An abstract interpretation-based framework for software watermarking // Proceedings of 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2004, p. 173-185.
[67]. Zakharov V.A. Ivanov K.S. Program obfuscation as obstruction of program static analysis // *Труды Института системного программирования РАН*, 2004, т. 6. с. 141-161.
[68]. Захаров В.А., Иванов К.С. О противодействии некоторым алгоритмам статического анализа программ // *Труды конференции 'Математика и безопасность информационных технологий' (МаБИТ-03)*, 2003, с. 282-286.
[69]. Dalla Preda M., Giacobazzi R. Semantic-based code obfuscation by abstract interpretation // *International Colloquium on Automata, Language and Programming*, Lecture Notes in Computer Science, v. 3580, 2005, p.1325-1336.
[70]. Захаров В.А., Иванов К.С. О моделях программ в связи с задачей противодействия алгоритмам статического анализа программ // *Труды Института системного программирования РАН*, 2006, т. 11.
[71]. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Подловченко Р.И., Шокуров А.В., Щербина В.Л. О применении методов деобфускации программ для обнаружения сложных компьютерных вирусов // *Известия Таганрогского радиотехнического университета*, 2006, т. 6, с. 18-27.

[72]. Kuzurin N.N., Podlovchenko R.I., Scherbina V.L., Zakharov V.A. Using algebraic models of programs for detecting metamorphic malwares // *Труды Института системного программирования РАН*, 2007, т. 12, с. 77-94.
[73]. Della Preda M., Giacobazzi G. Semantic-based code obfuscation by abstract interpretation // Journal of Computer Security, 2009, v. 17, N 6, p. 855-908.
[74]. Christodorescu M., Jha S. Static analysis of executables to detect malicious patterns // *Proceedings of the 12-th Security Symposium*, 2003, p. 169-186.
[75]. Della Preda M., Christodorescu M., Jha S., Debray S. A semantic-based approach to malware detection // Proceedings of the *34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2007, p. 377-388.
[76]. Della Preda M., Giacobazzi G., Debray S., Coogan K., Townsend G. Modelling Metamorphism by Abstract Interpretation // *Proceedings of the 17th International Static Analysis Symposium (SAS'10)*. Lecture Notes in Computer Science, 2010, v. 6337, p. 218-235.
[77]. Majumdar A, Thomborson C. On the use of opaque predicates in mobile agent code obfuscation // *Proceedings of the ISI 2005*. Altanta, GA, USA, 2005, p. 648-649.
[78]. Majumdar A., Thomborson C. Manufacturing opaque predicates in distributed systems for code obfuscation // *Proceedings of the 4th International Conference on Information Security*. Hobart, Tasmania, Australia, 2006, p. 187-196.
[79]. Della Preda M., Giacobazzi G., Madou M., de Bosschere K. Opaque predicate detection by abstract interpretation // *11th International Conference on Algebriac Methodology and Software Technology.* Lecture Notes in Computer Science, v 4019, 2006, p. 81-95.
[80]. Wang C., Davidson J., Hill J., Knight J. Protection of software-based survivability mechanisms // Proceedings of the International Conference of Dependable Systems and Networks, 2001.
[81]. Chow S., Gu Y., Johnson H., Zakharov V. An approach to obfuscation of control-flow of sequential programs // *Information Security Conference*, Lecture Notes in Computer Science, v. 2000, 2001, p. 144-155
[82]. Ogiso T., Sakabe Y., Soshi M. Miyaji A. Software obfuscation on a theoretical basis and its implementation // IEEE Transactions Fundamentals, E86-A(1), 2003.
[83]. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. О перспективах решения задачи обфускации компьютерных программ // Труды конференции 'Математика и безопасность информационных технологий' (МаБИТ-03), 2003, с. 344-351.
[84]. Ostrovsky R. Efficient computation on oblivious RAMs // *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990, p. 514-523.
[85]. Zhuang X., Zhang T., Lee H.-H. S., Pande S. Hardware assisted control flow obfuscation for embedded processes // Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2004, p. 292-302.
[86]. Bhatkar S., du Varney D.C., Sekar R. Address obfuscation: an efficient approach to combat a broad range of memory error exploits // Proceedings of the 12th conference on USENIX Security Symposium, 2003, v. 8.
[87]. Garg S., Gentry C., Halevi S., Raykova M., Sahai A., Waters B. Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits // IACR Cryptology ePrint Archive 2013, 451 (2013).
[88]. Hada S. Secure obfuscation for encrypted signatures // *Advances in Cryptology - EUROCRYPT 2010*, Lecture Notes in Computer Science, v. 6110, 2010, p. 92-112.
[89]. Adida B. , Wikström D. How to shuffle in piblic // *Proceedings of the 4th Conference on Theory of Cryptography*, Lecture Notes in Computer Science, 2007, v. 4392, p. 555-574.

[90]. Canetti R., Dwork C., Naor M., Ostrovsky R. Deniable encryption // *Advances in Cryptology- CRYPTO 97*, Lecture Notes in Computer Science, v. 1294, 1997, p. 90-104.
[91]. Sahai A., Waters B. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More // CRYPTO ePrint 2013.
[92]. Hada S. Zero-knowledge and code obfuscation // *Advances in Cryptology- ASIACRYPT 2000*, Lecture Notes in Computer Science, v. 1976, 2000, p. 443-457.
[93]. Savage J. Models of Computation: Exploring the Power of Computing. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997, 672 p.
[94]. Valiant L. A theory of learnable // Communications of the ACM, 1984, v. 27, N 11, p. 1134-1142.
[95]. Bitansky N., Canetti R. On obfuscation with strong simulators // *Advances in Cryptology- CRYPTO 2010*, Lecture Notes in Computer Science, v. 6223, 2010, p. 520-537.
[96]. Goldwasser S., Kalai T.Y. On the impossibility of obfuscation with auxiliary input // *Proceedings of the 46-th IEEE Symposium on Foundations of Computer Science*, 2005, p. 553-562.

The current state of art in program obfuscations:definitions of obfuscation security

[1] *N.P. Varnovsky*
[2] *V.A.Zakharov <zakh@cs.msu.su>*
[3] *N.N. Kuzurin <nnkuz@ispras.ru>*
[3] *V.A. Shokurov <shok@ispras.ru>*
[1] *Information Security Institute, Office 10, 1, Michurinskiy prospect, Moscow, 119192, Russian Federation*
[2] *Lomonosov Moscow State University, Faculty CMC, 2nd Education Building, GSP-1, Leninskie Gory, Moscow*
[3] *ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation*

Annotation. Program obfuscation is a semantic-preserving transformation aimed at bringing a program into such a form, which impedes the understanding of its algorithm and data structures or prevents extracting of some valuable information from the text of a program. Since obfuscation could find wide use in computer security, information hiding and cryptography, security requirements to program obfuscators became a major focus of interests for pioneers of theory of software obfuscation. In this paper we give a survey of various definitions of obfuscation security and main results that establish possibility or impossibility of secure program obfuscation under certain cryptographic assumptions. We begin with a short retrospective survey on the origin and development of program obfuscation concept in software engineering and mathematical cryptography. In the introduction we also point out on the main difficulties in the development of practical and secure obfuscation techniques. In the next section we discuss the main line of research in the application of program obfuscation to the solution of various problems in system programming and software security. Finally, in section 3 we present and discuss a compendium of formal definitions of the program obfuscation concept developed so far in mathematical cryptography - black-box obfuscation, gray-box obfuscation, the best possible obfuscation, obfuscation with auxiliary inputs, etc.. We also make a comparative analysis of these definitions and present the main results on the (im)possibility of secure program obfuscation w.r.t. every such definition.

Keywords: program, obfuscation, security, complexity, black box model, Turing machine

References

[1]. Diffie W., Hellman M. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), 1976, p.644-654.

[2]. Collberg C., Thomborson C., Low D. A Taxonomy of Obfuscating Transformations. Technical Report, N 148, Univ. of Auckland, 1997.
[3]. Cohen F. Operating system protection through program evolution. *Computers and Security*, v. 12, N 6, 1993, p. 565-584.
[4]. Chess D., White S. An undetectable computer virus. *Proceedings of the 2000 Virus Bulletin Conference*, 2000.
[5]. Szor P., Ferrie P. Hunting for metamorphic. *Proceedings of the 2001 Virus Bulletin Conference*, 2001, p.123-144.
[6]. Collberg C., J. Nagra. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Program Protection. Addison-Wesley Professional, 2009.
[7]. Aucsmith D. Tamper resistant software: an implementation. *Information Hiding Conference*, Lecture Notes in Computer Science, v. 1174, 1996, p. 317-333.
[8]. Scud T.T. ObjObf - x86/Linux ELF relocateable object obfuscator, 2003. http://packetstormsecurity.org/files/31524/objobf-0.5.0.tar. bz2.
[9]. Solutions P. DashO - the premier Java obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dasho/.
[10]. Solutions P. Dotfuscator - the premier .NET obfuscator and efficiency enhancing tool. http://www.preemptive.com/products/dotfuscator/.
[11]. Z. KlassMaster. The second generation Java obfuscator. http://www.zelix.com/.
[12]. Ge J., Chaudhuri S., Tyagi A. Control Flow Based Obfuscation. *Proceedings of the Digital Rights Management Workshop*. Alexandria, VA, USA, 2005, p. 83-92
[13]. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Ke Yang. On the (im)possibility of obfuscating programs. *Advances in Cryptology - CRYPTO'01*, Lecture Notes in Computer Science, v. 2139, 2001, p. 1-18 (see also *Journal of the ACM* 2012).
[14]. Varnovsky N.P. A note on the concept of obfuscation. *Trudy ISP RAN* [The Proceedings of ISP RAS], vol. 6, 2004, p. 127-137.
[15]. Kuzurin N.N., Shokurov A.V., Varnovsky N.P., Zakharov V.A. On the concept of software obfuscation in computer security. *Information Security Conference*, Lecture Notes in Computer Science, v. 4779, 2007, p. 281-298.
[16]. Goldwasser S., Rothblum G.N. On best possible obfuscation. *Theory of Cryptography Conference*, Lecture Notes in Computer Science, v. 4392, 2007, p. 194-213.
[17]. Canetti R. Towards realizing random oracles: hash functions that hide all partial information. *Advances in Cryptology - CRYPTO'97*, Lecture Notes in Computer Science, v. 1294, 1997, p. 455-469.
[18]. Varnovsky N.P., Zakharov V.A. On the possibility of provably secure obfuscating programs. *Conference ``Perspectives of System Informatics''*, Lecture Notes in Computer Science, v. 2890, 2004, p. 91-102.
[19]. Lynn B., Prabhakaran M., Sahai A. Positive results and techniques for obfuscation. *Advances in Cryptology - EUROCRYPT 2004*, Lecture Notes in Computer Science, v. 3027, 2004, p. 20-39.
[20]. Wee H. On obfuscating point functions. *Proceedings of the 37-th Symposium on Theory of Computing*, 2005, p. 523-532.
[21]. Hofheinz D., Malone-Lee J., Stam M. Obfuscation for cryptographic purpose. *Theory of Cryptography Conference*, Lecture Notes in Computer Science, v. 4392, p. 214-232.
[22]. Canetti R., Dakdouk R. R. Obfuscating point functions with multibit output. *Advances in Cryptology – EUROCRYPT 2008*, Lecture Notes in Computer Science, 2008, v. 4965, p. 489–508.

[23]. Hohenberger S., Rothblum G.N., Shelat A., Vaikuntanathan V. Securely obfuscating re-encryption. *Proceedings of the 4-th Conference on Theory of Cryptography*, 2007, p. 233-252

[24]. Canetti R., Rothblum G.N., Varia M. Obfuscation of hyperplane membership. *Proceedings of the 7-th Conference on Theory of Cryptography*, 2010, p. 72-89.

[25]. Collberg C., Thomborson C., Low D. Manufacturing cheap, resilient and stealthy opaque constructs. *Proceedings of the Symposium on Principles of Programming Languages*, 1998, p. 184-196.

[26]. de Oor A., van der Oord L. Stealthy obfuscation techniques: misleading pirates. Technical Report of Department of Computer Science University of Twente Enschede, Netherlands, 2003.

[27]. Naumovich G, Memon N. Preventing piracy, reverse engineering, and tampering. *IEEE Computer*, 2003, v. 36, N 7, p. 64-71.

[28]. Collberg C, Thomborson C., Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection. *IEEE Transactions on Software Engineering*, v. 28, N 6, 2002.

[29]. Arboit G. A method for watermarking Java programs via opaque predicates. *Proceedings of the International Conference on Electronic Commerce Research*. Montreal, Canada, 2002: 1-8.

[30]. Zhu W., Thomborson C., Wang F.-Y. A survey of software watermarking. Lecture Notes in Computer Science, v.3495, 2005, p. 454-458.

[31]. Myles G, Collberg C. Software watermarking via opaque predicates: Implementation, analysis, and attacks. *Electronic Commerce Research*, 2006, v. 6, N 2, p. 155-171.

[32]. Sander T., Tchudin C.F. Protecting mobile agents against malicious hosts. *Mobile Agents and Security*, Lecture Notes in Computer Science, 1997, p. 44-60.

[33]. Hohl F. Time limited blackbox security: protecting mobile agents from malicious hosts. *Mobile Agents and Security*, Lecture Notes in Computer Science, v. 1419, 1998, p. 92-113.

[34]. D'Anna L., Matt B., Reisse A., Van Vleck T. , Schwab S., LeBlanc P. Self-Protecting Mobile Agents Obfuscation Report. Tech. Rep. N 03-015, Network Associates Laboratories, June 2003.

[35]. Wu J., Zhang Y., Wang X. et al. A scheme for protecting mobile agents based on combining obfuscated control flow and time checking technology. *Proceedings of the Conference on Computational Intelligence and Security*. Harbin, Heilongjiang, China, 2007, p. 912-916

[36]. Roeder T., Schneider F.B. Proactive Obfuscation. *ACM Transactions on Computer Systems*, v. 28, N 2, 2010.

[37]. Ostrovsky R., Skeith W.E. Private searching on streaming data. *Advances in Cryptology - CRYPTO-2005*, Lecture Notes in Computer Science, v. 3621, 2005, p. 223-240.

[38]. Narayanan A., Shmatikov V. Obfuscated databases and group privacy. *Proceedings of the 12th ACM Conference on Computer and Communications Security*, 2005, p. 102-111.

[39]. Ivannikiov V.P., Varnovsky N.P., Zakharov V.A., Kuzurin N.P., Shokurov A.V., Kononov A.N., Kalinin A.V. Metody informazionnoy zaschity proektnyh resheniy pri izgotovlenii microelectronnyh shem [Information security techniques in the development of microelectronic circuits] *Izvestiya Taganrogskogo radiotehnicheskogo universiteta* [Bulletin of Taganrog Radiotechnical University], 2005, v. 4, p. 112-119.

[40]. Varnovsky N.P., Zakharov V.A., Kuzurin N.P., Chernov A.V., Shokurov A.V. Zadaschi I metody obespecheniya informazionnoy bezopasnosti pri proizvodstve microelectronnyh shem [Information security problems and techniques in the

development of microelectronic circuits], *Trudy ISP RAN* [The Proceedings of ISP RAS], 2006, т. 1, с. 29-61.
[41]. Borello J.M., Me L. Code obfuscation technique for metamorphic viruses. *Journal of Computer Virology*, 2008, v. 4, p, 211-220.
[42]. Bhatkar S., Du Varney D.C., Sekar R. Efficient techniques for comprehensive protection from memory error exploits. *USENIX Security*, 2005.
[43]. Wroblewski G. General method of program code obfuscation. Draft, 2002, 84 p.
[44]. Linn C., Debray S. Obfuscation of executable code to improve resistance to static disassembly. *Proceedings of the 10-th ACM Conference on Computer and Communication Security*, 2003, p. 290-299.
[45]. Sosonkin M, Naumovich G, Memon N. Obfuscation of design intent in object-oriented applications. *Proceedings of the Digital Rights Management Workshop*. Washington, DC, USA, 2003, p. 142-153.
[46]. Collberg C., Myles G., Huntwort A. Sandmark – a tool for software protection research. *IEEE Security and Privacy*, 2003, v. 1, N 4, p. 40-49.
[47]. Heffner K., Collberg C. The obfuscation executive. *Information Security Conference*, Lecture Notes in Computer Science, 2004, v. 3225.
[48]. Chan J. T., Yang W. Advanced obfuscation techniques for Java bytecode. *Journal of Systems and Software*, 2004, v. 71, N 1-2, p. 1-10.
[49]. Cimato S., De S. A., Petrillo U. F. Overcoming the obfuscation of Java programs by identifier renaming. *Journal of Systems and Software*, 2005, v. 78, N 1, p. 60-72.
[50]. Madou M., Anckaert B., de Sutter B., de Bosschere K. Hybrid static-dynamic attacks against software protection mechanisms. *Proceedings of the 5th ACM workshop on Digital rights management*, 2005, p. 75-82.
[51]. Udupa S. K., Debray S. K., Madou M. Deobfuscation: Reverse engineering obfuscated code. *Proceedings of the 12-th Working Conference on Reverse Engineering*. Pittsburgh, PA, USA, 2005, p. 45-54.
[52]. Ge J., Chaudhuri S., Tyagi A. Control Flow Based Obfuscation. *Proceedings of the Digital Rights Management Workshop*. Alexandria, VA, USA, 2005, p. 83-92.
[53]. Chen K., Chen J. B. On instrumenting obfuscated java bytecode with aspects. *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*. Shanghai, China, 2006, p. 19-26.
[54]. Madou M., Anckaert B., de Sutter B., de Bosschere K., Cappaert J., Preenel B. On the effectiveness of source code transformations for binary obfuscation. *Proceedings of the International Conference on Software Engineering Research and Practice*, 2006, p.527-533.
[55]. Madou M., Anckaert B., Moseley P., Debray S., de Sutter B., de Bosschere K. Software protection through dynamic code mutation. *Proceedings of the 6-th international conference on Information Security Applications*, 2006, p. 194-206.
[56]. Drape S, Majumdar A, Thomborson C. Slicing aided design of obfuscating transforms. *Proceedings of the International Computing and Information Systems Conference (ICIS 2007)*. Melbourne, Australia, 2007, p. 1019-1024.
[57]. Majumdar A., Drape S., Thomborson C. Slicing obfuscations: Design, correctness, and evaluation. *Proceedings of the 2007 ACM Workshop on Digital Rights*. Alexandria, VA, USA, 2007, p. 70-81.
[58]. Batchelder M., Hendren L. Obfuscating Java: The most pain for the least gain. *Proceedings of the Compiler Construction*. Braga, Portugal, 2007, p. 96-110.

[59]. Ceccato M., Di. P. M., Nagra J. et al. Towards experimental evaluation of code obfuscation techniques. *Proceedings of the 4th ACM Workshop on Quality of Protection*., 2008, p. 39-46.
[60]. Darwish S.M., Guirguis S.K., Zalat M.S. Stealthy code obfuscation technique for software security. *Proceedings of the International Conference on Computer Engineering and Systems*, 2010, p. 93-99.
[61]. Chernov A.V. Ob odnom metode maskirovki program [On one program obfuscation techniques]. *Trudy ISP RAN* [The Proceedings of ISP RAS], 2003, v. 4, p. 85-119.
[62]. Majumdar A., Drape S., Thomborson C. et al. Metrics-based evaluation of slicing obfuscations. *Proceedings of the 3rd International Symposium on Information Assurance and Security*. Manchester, United Kingdom, 2007, p. 472-477.
[63]. Naeem N. A., Batchelder M., Hendren L. Metrics for Measuring the Effectiveness of Decompilers and Obfuscators. *Proceedings of the 15th IEEE International Conference on Program*. Banff, Alberta, Canada, 2007, p. 253-258.
[64]. Anckaert B., Madou M., De S. B. et al. Program obfuscation: A quantitative approach. *Proceedings of the 2007 ACM Workshop on Quality of Protection*. 2007, p. 15-20.
[65]. Tsai H. Y., Huang Y. L., Wagner D. A graph approach to quantitative analysis of control-flow obfuscating. *IEEE Trans. on Information Forensics and Security*, 2009, v. 4, N 2, p. 257-267.
[66]. Cousot P., Cousot R. An abstract interpretation-based framework for software watermarking. *Proceedings of 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2004, p. 173-185.
[67]. Zakharov V.A. Ivanov K.S. Program obfuscation as obstruction of program static analysis. *Trudy ISP RAN* [The Proceedings of ISP RAS], 2004, v. 6. p. 141-161.
[68]. Zakharov V.A. Ivanov K.S. O protivodeystvii nekotorym algorytmam staticheskogo analiza program [On the hindering some program static analysis algorithms]. *Trudy konferencii "Matematika i bezopasnost' informazionnyh tehnologiy" (MaBIT-03)* [Proceedings of the Conference "Mathematics and security of information technologies"], 2003, с. 282-286.
[69]. Dalla Preda M., Giacobazzi R. Semantic-based code obfuscation by abstract interpretation. *International Colloquium on Automata, Language and Programming*, Lecture Notes in Computer Science, v. 3580, 2005, p.1325-1336.
[70]. Zakharov V.A. Ivanov K.S. O modelyah program v svyazi s zadachey protivideystviya algoritmam ststicheskogo analiza [On the program models related with the proble of hindering program static analysis algorithms]. *Trudy ISP RAN* [The Proceedings of ISP RAS], 2006, т. 11.
[71]. Varnovsky N.P., Zakharov V.A., Kuzurin N.P., Podlovchenko R.I., Shokurov A.V., Shcherbina V.L. O primenenii metodov deobfuscazii program dlya obnaruzheniya slojnyh komputernyh virusov [On the application of program deobfuscation techniques for detecting non-trivial computer viruse]. *Izvestiya Taganrogskogo radiotehnicheskogo universiteta* [Bulletin of Taganrog Radiotechnical University], 2006, т. 6, с. 18-27.
[72]. Kuzurin N.N., Podlovchenko R.I., Scherbina V.L., Zakharov V.A. Using algebraic models of programs for detecting metamorphic malwares. *Trudy ISP RAN* [The Proceedings of ISP RAS], 2007, v. 12, p. 77-94.
[73]. Della Preda M., Giacobazzi G. Semantic-based code obfuscation by abstract interpretation. *Journal of Computer Security*, 2009, v. 17, N 6, p. 855-908.
[74]. Christodorescu M., Jha S. Static analysis of executables to detect malicious patterns. *Proceedings of the 12-th Security Symposium*, 2003, p. 169-186.

[75]. Della Preda M., Christodorescu M., Jha S., Debray S. A semantic-based approach to malware detection. *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2007, p. 377-388.

[76]. Della Preda M., Giacobazzi G., Debray S., Coogan K., Townsend G. Modelling Metamorphism by Abstract Interpretation. *Proceedings of the 17th International Static Analysis Symposium (SAS'10).* Lecture Notes in Computer Science, 2010, v. 6337, p. 218-235.

[77]. Majumdar A, Thomborson C. On the use of opaque predicates in mobile agent code obfuscation. *Proceedings of the ISI 2005*. Altanta, GA, USA, 2005, p. 648-649.

[78]. Majumdar A., Thomborson C. Manufacturing opaque predicates in distributed systems for code obfuscation. *Proceedings of the 4th International Conference on Information Security*. Hobart, Tasmania, Australia, 2006, p. 187-196.

[79]. Della Preda M., Giacobazzi G., Madou M., de Bosschere K. Opaque predicate detection by abstract interpretation. *11th International Conference on Algebriac Methodology and Software Technology.* Lecture Notes in Computer Science, v 4019, 2006, p. 81-95.

[80]. Wang C., Davidson J., Hill J., Knight J. Protection of software-based survivability mechanisms. *Proceedings of the International Conference of Dependable Systems and Networks*, 2001.

[81]. Chow S., Gu Y., Johnson H., Zakharov V. An approach to obfuscation of control-flow of sequential programs. *Information Security Conference*, Lecture Notes in Computer Science, v. 2000, 2001, p. 144-155

[82]. Ogiso T., Sakabe Y., Soshi M. Miyaji A. Software obfuscation on a theoretical basis and its implementation. I*EEE Transactions Fundamentals*, E86-A(1), 2003.

[83]. Varnovsky N.P., Zakharov V.A., Kuzurin N.P., Podlovchenko R.I., Shokurov A.V. O perspektivah resheniya zadach obfuscacii komputernyh program [On the prospects of the solution of the obfuscation problems for computer programs] *Trudy konferencii "Matematika i bezopasnost' informazionnyh tehnologiy" (MaBIT-03)* [Proceedings of the Conference "Mathematics and security of information technologies"], 2003, c. 344-351.

[84]. Ostrovsky R. Efficient computation on oblivious RAMs. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990, p. 514-523.

[85]. Zhuang X., Zhang T., Lee H.-H. S., Pande S. Hardware assisted control flow obfuscation for embedded processes. *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2004, p. 292-302.

[86]. Bhatkar S., du Varney D.C., Sekar R. Address obfuscation: an efficient approach to combat a broad range of memory error exploits. *Proceedings of the 12th conference on USENIX Security Symposium*, 2003, v. 8.

[87]. Garg S., Gentry C., Halevi S., Raykova M., Sahai A., Waters B. Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits. *IACR Cryptology ePrint Archive 2013*, 451 (2013).

[88]. Hada S. Secure obfuscation for encrypted signatures. *Advances in Cryptology - EUROCRYPT 2010*, Lecture Notes in Computer Science, v. 6110, 2010, p. 92-112.

[89]. Adida B. , Wikström D. How to shuffle in piblic. *Proceedings of the 4th Conference on Theory of Cryptography*, Lecture Notes in Computer Science, 2007, v. 4392, p. 555-574.

[90]. Canetti R., Dwork C., Naor M., Ostrovsky R. Deniable encryption. *Advances in Cryptology- CRYPTO 97*, Lecture Notes in Computer Science, v. 1294, 1997, p. 90-104.

[91]. Sahai A., Waters B. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. *CRYPTO ePrint* 2013.

[92]. Hada S. Zero-knowledge and code obfuscation. *Advances in Cryptology- ASIACRYPT 2000*, Lecture Notes in Computer Science, v. 1976, 2000, p. 443-457.
[93]. Savage J. Models of Computation: Exploring the Power of Computing. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997, 672 p.
[94]. Valiant L. A theory of learnable. *Communications of the ACM*, 1984, v. 27, N 11, p. 1134-1142.
[95]. Bitansky N., Canetti R. On obfuscation with strong simulators. *Advances in Cryptology-CRYPTO 2010*, Lecture Notes in Computer Science, v. 6223, 2010, p. 520-537.
[96]. Goldwasser S., Kalai T.Y. On the impossibility of obfuscation with auxiliary input. *Proceedings of the 46-th IEEE Symposium on Foundations of Computer Science*, 2005, p. 553-562.