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 27, выпуск 4

Volume 27, issue 4

Москва 2015

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

Труды Института системного программирования РАН

Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областям системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

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

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.

The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

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

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge.

Proceedings of ISP RAS is abstracted and/or indexed in:

УДК004.45

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

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

Содержание

Table of Contents

Модель представления данных при проведении глубокого анализа сетевого трафика*

[1]*А.И. Гетьман <thorin@ispras.ru>*
[1, 2, 3, 4]*В.П. Иванников <ivan@ispras.ru>*
[1]*Ю.В. Маркин <ustas@ispras.ru>*
[1, 2]*В.А. Падарян <vartan@ispras.ru>*
[1]*А.Ю. Тихонов <fireboo@ispras.ru>*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2]*119991 ГСП-1 Москва, Ленинские горы, Московский государственный университет имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК*
[3]*Московский физико-технический институт (государственный университет), 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[4]*НИУ Высшая школа экономики,*
Россия, Москва, 101000, ул. Мясницкая, д. 20

Аннотация. В статье предложена объектная модель представления данных при проведении глубокого анализа сетевого трафика. В отличие от модели, используемой большинством существующих сетевых анализаторов, в ней поддерживается восстановление потоков данных, а также проведение их дальнейшего разбора. Тем самым обеспечивается повышение уровня представления (согласно модели OSI) данных, необходимое при анализе сетевого трафика: для понимания механизмов взаимодействия сетевых приложений нужно восстанавливать данные в том виде, в котором этими данными оперируют приложения. На базе предложенной модели реализована инфраструктура для проведения глубокого анализа трафика. Модель предлагает универсальный механизм связывания разборщиков заголовков сетевых протоколов – появляется возможность для независимой разработки функций разбора. Модель также предоставляет функционал для работы с модифицированными (в частности, зашифрованными) данными.

Ключевые слова: анализ сетевого трафика; восстановление потоков данных; модель представления данных; распознавание данных.

DOI: 10.15514/ISPRAS-2015-27(4)-1

* Работа поддержана грантом РФФИ 15-07-07652 А

Для цитирования: Гетьман А. И., Иванников В.П., Маркин Ю.В., Падарян В.А., Тихонов А.Ю. Модель представления данных при проведении глубокого анализа сетевого трафика. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 5-22. DOI: 10.15514/ISPRAS-2015-27(4)-1.

1. Введение

В настоящее время задача анализа сетевого трафика приобретает все большую актуальность: этому способствует как развитие и внедрение новых сетевых технологий, так и появление большого количества новых сетевых протоколов прикладного уровня. Перечислим некоторые практические задачи анализа:

- выявление проблем в работе сети
- тестирование (отладка) сетевых протоколов [1, 2]
- сбор статистики, мониторинг сетевых каналов
- предотвращение сетевых атак [3]

Существует большое количество как коммерческих, так и свободно распространяемых сетевых анализаторов [4]. Как правило, каждый инструмент направлен на решение только одной практической задачи, опираясь на "базовый" функционал – разбор заголовков сетевых протоколов и восстановление потоков передаваемых данных.

Большинство существующих сетевых анализаторов поддерживают два режима работы:

- анализ трафика, поступающего на сетевой интерфейс в режиме реального времени (далее *онлайн*-анализ)
- анализ предварительно сохраненных сетевых трасс (далее *оффлайн*-анализ)

В режиме онлайн-анализа инструмент должен работать непрерывно с производительностью, достаточной для разбора трафика, поступающего на сетевой интерфейс. При этом должна обеспечиваться возможность обработки потенциально бесконечного входного потока данных.

В случае оффлайн-анализа инструмент получает входные данные (конечного размера) из файла. Поэтому может проводиться более детальный анализ по сравнению с онлайн-анализом на аналогичном трафике.

На практике в большинстве существующих инструментов оффлайн режим полностью повторяет работу в онлайне за одним исключением: вместо сетевого интерфейса пакеты считываются из файла с сетевой трассой. Отсутствие требований к скорости обработки данных в оффлайн режиме открывает дополнительные возможности:

- визуализировать структуру всех разобранных данных
- анализировать восстановленные потоки данных прикладного уровня
- применять другие разборщики (по сравнению с разборщиком, выбранным инструментом) к данным при просмотре результатов
- проводить отладку модулей разбора заголовков протоколов
- расследовать инциденты нарушения сетевой безопасности

Ограничения на проведение детального оффлайн-анализа главным образом обусловлены архитектурными особенностями инструментов: большая часть анализаторов изначально ориентирована на работу в режиме онлайн. При этом отсутствует возможность дальнейшего разбора восстановленных потоков передаваемых данных. Предлагается реализовать две *независимые* системы: одну для онлайн-, другую – для оффлайн-анализа. Обе системы должны использовать единую инфраструктуру, включающую модули разбора заголовков протоколов (полный список требований будет приведен ниже). Построенная по такому принципу система позволит в полной мере использовать преимущества оффлайн-анализа, а также осуществлять разбор заголовков протоколов и восстановление потоков передаваемых данных в режиме онлайн. Основной результат, представляемый в статье – объектная *модель* представления сетевых данных, которая будет использоваться двумя системами.

Требования к системам глубокого анализа пакетов, работающим в онлайн режиме, достаточно проработаны: в 2012 году был опубликован стандарт МСЭ-Т [5]. В свою очередь для аналогичных оффлайн систем требования не уточнялись. В разделе 2 составляется список уточненных функциональных требований для каждой из систем, из которых естественным образом формулируются требования к модели представления данных. В разделе 3 описана модель данных, используемая существующими анализаторами сетевого трафика, такими как Snort [6], Wireshark [7], The Bro Network Security Monitor [8]. Особое внимание уделяется ее недостаткам и ограничениям. Способ преодоления этих недостатков в виде новой модели представления сетевых данных приводится в Разделе 4. В разделе 5 делаются итоговые заключения.

2. Требования к системе разбора

Здесь и далее будем считать, что данные по сети передаются посредством пакетов. Каждый сетевой пакет состоит из управляющей информации и полезной нагрузки. В процессе разбора в пакете выделяются заголовки протоколов, анализируются значения полей этих заголовков. Если структура заголовка определяется спецификацией, то полезная нагрузка может

содержать произвольным образом организованные данные, хотя обычно представляет собой пакет протокола более высокого уровня – анализатор должен самостоятельно определить, какой это протокол. В случае успеха разбор будет продолжен. Рис. 1 иллюстрирует этот процесс.

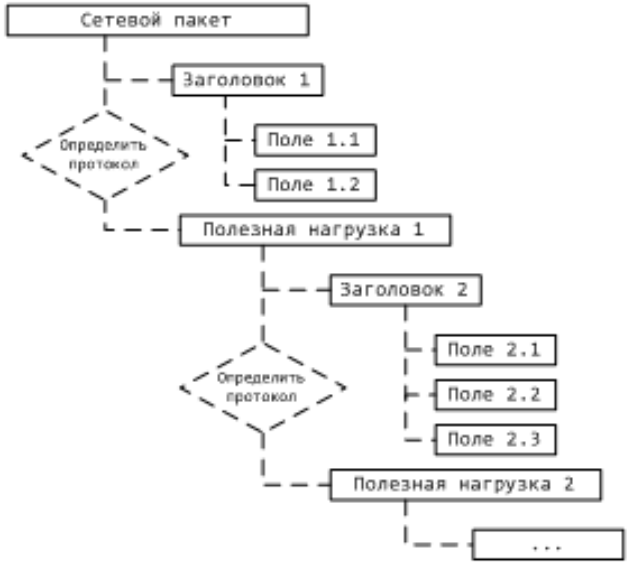

Рис. 1 – Выделение и разбор заголовков протоколов.

В соответствии с моделью OSI содержимое пакета может быть проинтерпретировано посредством стека заголовков сетевых протоколов. Обычно порядок следования этих заголовков естественный – от протоколов более низкого уровня к протоколам более высокого уровня. Однако при организации туннельного соединения порядок следования заголовков может быть нарушен. Система разбора должна корректно обрабатывать подобные ситуации.

Некоторые сетевые протоколы (например, IPv4 [9]) характеризуются максимально допустимым размером полезной нагрузки, передаваемой в одном пакете (Maximum Transmission Unit, или MTU). Соответственно, если величина MTU превышена, полезная нагрузка разбивается на части допустимого размера и передается посредством нескольких сетевых пакетов. В таких случаях система разбора должна проводить дефрагментацию (рис. 2).

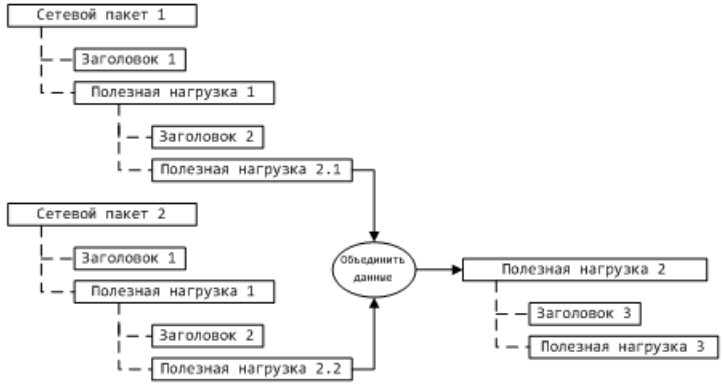

Рис. 2 – Дефрагментация данных (объединение).

Для обеспечения большей безопасности соединений многие протоколы (в частности, TLS [10, 11, 12], SSH [13]) осуществляют передачу данных в зашифрованном виде. Для проведения разбора зашифрованных данных необходимо их предварительно расшифровать, использовав предоставленный пользователем ключ (рис. 3). Обобщая, можно сказать, что система разбора должна предоставлять пользователю интерфейс для добавления недостающей информации, необходимой при проведении разбора. Заметим, что это требование не является обязательным согласно рекомендации МСЭ-Т.

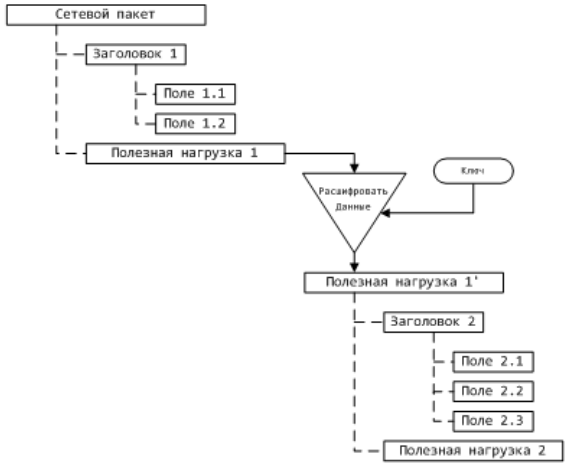

Рис. 3 – Дешифрование данных (преобразование).

При анализе трафика неизбежно возникают *ошибки разбора*. Под ошибкой разбора понимается несоответствие между спецификацией протокола (кодом

разборщика) и сетевым пакетом, разбор которого осуществляется согласно этой спецификации. Причины возникновения ошибок разбора различны:

- ошибки в коде разборщиков,
- недокументированные возможности протоколов,
- искажения данных при передаче по сети.

В системе должна присутствовать возможность локализации данных, разбор которых приводит к возникновению ошибки. При этом, если обнаруженная ошибка не является критической, анализ должен продолжаться.

Как правило, анализаторы сетевого трафика имеют модульную структуру (Wireshark, The Bro Network Security Monitor). Это обусловлено тем, что со временем появляются новые сетевые протоколы, и их необходимо поддерживать. Расширять систему, в которой все разборщики сосредоточены в одном функциональном модуле, затруднительно. В случае модульной архитектуры для каждого протокола создается отдельный модуль, в котором локализована функциональность по работе с этим протоколом. Возникает дополнительный вопрос о зависимостях: при добавлении нового модуля, необходимо "сообщить" о его существовании остальным модулям. Вносить изменения в код существующих модулей крайне неудобно и неэффективно – это потенциальный источник ошибок. К тому же, по окончании правки потребуется повторная сборка модуля. Поэтому необходимо реализовать механизм, позволяющий добавлять новые разборщики (модули разбора) без внесения изменений в уже существующие функции разбора.

Глубокий анализ сетевого трафика предполагает проведение *полного* разбора. Это разбор заголовков протоколов всех уровней в соответствии с моделью OSI, а также сохранение восстановленных потоков передаваемых данных. Возможность проведения полного разбора является ключевым требованием к системе.

С учетом сформулированных требований предлагается следующий (основной) сценарий эксплуатации. Посредством системы онлайн-анализа проводится разбор пакетов (некоторого сетевого интерфейса) в режиме 24/7. *Результаты разбора* сохраняются на жестком диске. Под результатами здесь понимается множество всех восстановленных потоков (см. ниже) всех протоколов. Аналитик с некоторой периодичностью просматривает сообщения об ошибках, выдаваемые системой разбора. Если количество ошибок, возникающих при работе разборщика, превышает предварительно заданное пороговое значение, принимается решение о необходимости доработки кода этого разборщика. Для последующей отладки кода разборщика сохраняется сетевая трасса. Решение о внесении изменений в код принимается по результатам оффлайн-анализа сохраненной трассы.

Фактически работа аналитика с системой оффлайн-анализа заключается в получении и накоплении достаточных знаний о структуре данных для усовершенствования на их основе системы онлайн-анализа. Поэтому крайне важна совместимость модулей разбора заголовков протоколов для двух систем. Обеспечение такой совместимости может быть достигнуто только при достаточном уровне проработки архитектуры этих систем.

Главное, но вполне естественное ограничение системы онлайн-анализа – это ресурсы вычислительной машины. Поскольку поток входных данных потенциально бесконечен, время от времени или по наступлению некоторого события необходимо сохранять результаты разбора на жесткий диск (или передавать их другому анализатору). Также заметим, что система онлайн-анализа не предполагает никакой визуализации структуры разобранных данных.

Система оффлайн-анализа напротив имеет дело с сетевой трассой конечного размера (фактически, отсутствует ограничение, связанное с вычислительными ресурсами). Основное предназначение данной системы – отладка разборщиков. Наглядное представление (визуализация) структуры разобранных пакетов существенно упрощает процесс отладки. Также может оказаться полезным удобный механизм навигации между пакетами и восстановленными потоками.

3. Существующие анализаторы сетевого трафика

Большинство существующих анализаторов поддерживает проведение как онлайн-, так и оффлайн-анализа. В первом случае в качестве источника пакетов выступает сетевое оборудование, во втором – файл сетевой трассы. Разбор пакета заключается в выделении полей всех присутствующих в нем заголовков сетевых протоколов. Под выделением поля подразумевается сопоставление ему некоторого непрерывного *блока* данных известного размера. Как следствие, выделенный блок может приобрести некоторую семантику, например, блок, задающий длину пакета.

Большинство существующих систем разбора оперирует такими понятиями, как пакет, протокол, блок данных. Соответствующая такому подходу модель (далее *базовая модель*) представления данных показана на рис. 4.

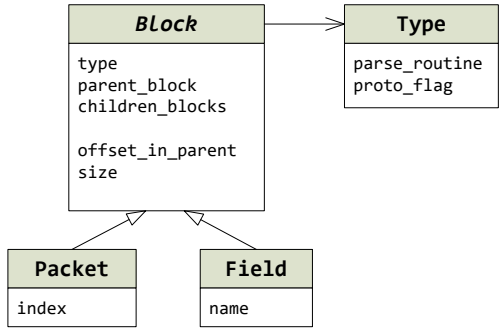

Рис. 4 – Базовая модель представления данных.

Каждый блок хранит указатель на *родительский* блок (parent_block), а также список указателей на *дочерние* блоки (children_blocks). Родительский блок – это блок, внутри которого содержится данный блок. Дочерний блок – это блок, для которого данный блок является родительским. Таким образом поддерживается отношение вложенности блоков, благодаря чему можно ассоциировать сетевой пакет (буфер данных) с *деревом* его разбора. Заметим, что размер блока равен сумме размеров всех его дочерних блоков (если таковые имеются).

Блок обладает *размером* (size) и *смещением* (offset_in_parent) по отношению к родительскому блоку. Блок обладает *типом* (type). Тип определяет разборщик (parse_routine), который будет использоваться при анализе данных соответствующего блока. Дополнительно тип может обладать признаком (proto), указывающим на то, что блок с таким типом представляет собой заголовок сетевого протокола. Этот флаг позволяет группировать блоки и выделять заголовки протоколов при отображении структуры разобранных пакетов.

Каждый блок обладает дополнительным атрибутом: либо *именем* (name – строка), либо порядковым номером (index – целочисленная переменная). Нумерованные блоки описывают сетевые пакеты и элементы массивов, именованные – поля заголовков протоколов.

Обработка данных в соответствии с базовой моделью происходит следующим образом. На вход инструменту подается буфер данных (сетевая трасса в случае оффлайн-анализа). В соответствии с типом данных буфера выбирается нужный разборщик. Разборщик выделяет блоки и при необходимости вызывает другие функции разбора. Результатом работы такой системы является некоторое древовидное представление буфера, в котором каждому выделенному блоку соответствует вершина дерева. Система разбора позволяет аналитику просматривать данные, соответствующие каждому блоку.

Перечислим основные недостатки базовой модели представления данных:

1. Каждый разборщик должен самостоятельно определять, какие функции разбора следует вызывать. Это лишает систему гибкости: при добавлении новых типов потребуется модифицировать функции разбора существующих типов так, чтобы они могли использовать добавленные типы. На практике это означает, что для предоставления возможности добавлять новые функции разбора, разработчик должен открыть исходный код всех реализованных разборщиков. Ни один коммерческий анализатор таких возможностей не предоставляет.
2. Древовидное описание структуры не позволяет описывать пакеты, содержащие зашифрованные или сжатые данные. В то же время объем данных, передаваемых по сети в модифицированном виде, постоянно растет.
3. Описание структуры единственного входного буфера, содержащего все пакеты, не позволяет описывать *сборку сеансов*. Под сеансом в данном случае понимается восстановленная (частично или полностью) единица передачи данных потокового протокола, отправленная (или полученная) посредством более чем одного сетевого пакета. В некоторых инструментах восстановление сеансов реализовано дополнительными модулями (например, Snort, The Bro Network Security Monitor), однако эти сеансы не подвергаются дальнейшему разбору. В то же время, анализ собранных сеансов позволяет восстанавливать высокоуровневые данные прикладных протоколов, тем самым проясняя логику взаимодействия между узлами сети.

Заметим также, что большинство анализаторов предназначено либо для длительной работы в режиме сбора статистики без проведения полного разбора и сохранения структуры данных, либо для кратковременной работы в режиме накопления данных, их разбора и последующего отображения структуры.

4. Формальное описание модели представления данных

Здесь и далее будем называть декларируемую модель *расширенной* по сравнению с базовой моделью.

В расширенной модели процесс разбора заключается в выделении логически непрерывных *блоков* данных. Блок по сути является обобщением понятия поля базовой модели (полем в расширенной модели называется блок, обладающий определенными характеристиками). В процессе анализа блока может потребоваться отдельно обработать какую-либо его часть. Для этого будет создан новый блок, являющийся дочерним по отношению к исходному блоку. Понятия *родительского* и *дочернего* блоков аналогичны понятиям родительского и дочернего полей соответственно. Каждый блок характеризуется положительным числом – размером. В процессе разбора каждому блоку в соответствие ставится *тип*. Тип (разбора) определяет разборщик, посредством которого обрабатываются данные блока. Тип обладает уникальными именем.

Для описания *сборки сеансов* вводится понятие *буфера данных*. Данные блоков могут быть добавлены в конец буфера посредством копирования. Буфер характеризуется размером: в каждый момент времени размер буфера определяется как сумма размеров добавленных в него блоков данных. Содержимое буфера анализируется (посредством разборщиков) так же, как и данные блоков. Здесь возникает необходимость различать блоки, владеющие буфером – *потоки*, и блоки, не владеющие буфером – *фрагменты*. Для случаев, когда требуется сборка сеанса (в частности, при разборе TCP-пакетов [14]), необходимо создать поток, после чего добавить в его буфер нужные данные. Схематично этот процесс показан на рис. 5. Заметим, что поток может быть создан или дополнен любой функцией разбора.

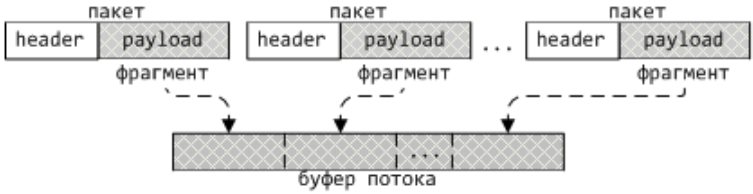

Рис. 5 – Добавление данных нескольких фрагментов в буфер потока.

Также отметим, что данные фрагмента локализуются посредством *смещения* относительно начала данных блока-родителя. Для случаев, когда в качестве родителя выступает поток, смещение задается относительно начала буфера этого потока.

Функциональность по сборке сеансов обеспечивает последовательное повышение уровня (согласно модели OSI) представления данных: по окончанию сборки проводится разбор сеанса, в процессе которого, возможно, будут восстановлены сеансы для протоколов более высокого уровня (рис. 6).

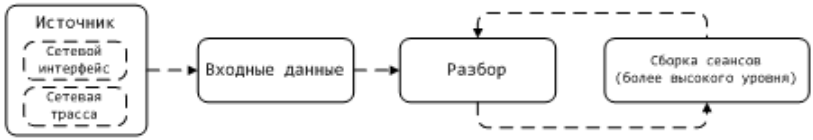

Рис. 6 – Схема работа с данными.

Для того, чтобы система разбора могла работать с модифицированными (например, зашифрованными) данными, вводится понятие *замещающего* фрагмента (фрагменты, не являющиеся замещающими, далее будем называть *простыми*). Проводить разбор зашифрованных данных невозможно – предварительно необходимо провести дешифровку (аналитику могут быть известны алгоритмы дешифрования, а также значения используемых ключей). Дешифрованные данные помещаются в предварительно созданный буфер. Этот буфер ассоциируется с тем же фрагментом, который описывает зашифрованные данные. Таким образом, фрагмент может обладать дополнительным буфером памяти, причем дальнейшему разбору подлежат данные именно этого (*замещающего*) буфера. Предложенный подход естественным образом обобщается для любой модификации данных, допускающей обратное преобразование. Дочерние блоки замещающего фрагмента характеризуются смещением относительно начала данных соответствующего замещающего буфера. Заметим, что размер замещающего буфера в общем случае никак не связан с размером *оригинальных* данных фрагмента. Также отметим, что для описания замещающих фрагментов в рамках древовидной структуры необходимо добавить отличительный признак для вершин, владеющих буфером – будем называть такие вершины *источниками данных*. Тогда простой фрагмент определяется смещением относительно "ближайшего" (при переходе к родительскому блоку) источника данных.

Для реализации в системе универсальных (не требующих внесения изменений в код при добавлении новых типов) разборщиков вводится понятие *распознавателя*. Распознаватель – это функция, которая по данным блока и, возможно, некоторой дополнительной информации определяет тип этого блока. Перед началом разбора блока его тип либо известен, либо нет. В последнем случае необходимо определить тип с помощью распознавателей. Система разбора предоставляет аналитику функции для регистрации распознавателей, а также отвечает за корректное использование уже зарегистрированных распознавателей.

Чаще всего функциональность по распознаванию требуется для полей в заголовках протоколов, допускающих данные различных типов. Например, поле "Payload" в заголовке протокола IPv4 может содержать данные протоколов TCP, UDP [15] и т. д. В данном случае тип определяется значением поля "Protocol" того же заголовка. Более формально, для

определения типа фрагмента используются данные блока-родителя. Такие распознаватели будем называть *распознавателями полей*. При регистрации распознавателя поля в системе необходимо указать имя соответствующего поля, а также тип блока-родителя. Распознаватель поля применяется только к полю с заданным именем при условии, что соответствующий родительский блок обладает заданным типом.

Для определения типа собранных сеансов используются *распознаватели потоков*. Распознаватель потока на вход получает только данные соответствующего буфера. Дополнительно может использоваться *тип сборки* потока (тип сборки потока выставляется в соответствии с типом блоков, данные которых попали в буфер потока). Тип сборки позволяет ограничить (при необходимости) множество применяемых к потоку распознавателей. Такие распознаватели будем называть *распознавателями потоков с известным типом сборки*. Если распознаватель потока не использует тип сборки, то его также можно применять для определения типа фрагмента. Такие распознаватели будем называть *распознавателями потоков с неизвестным типом сборки*.

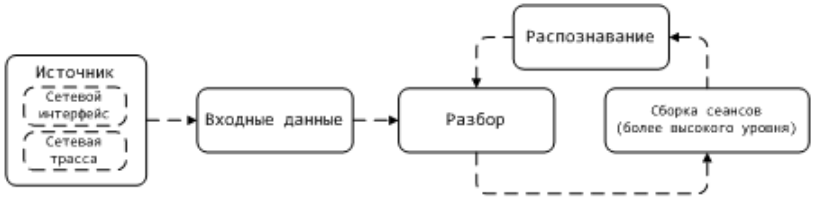

Рис. 7 – Дополненная схема работы с данными.

Распознаватели естественным образом дополняют схему работы с данными в рамках системы (рис. 7). Они служат основным средством обеспечения независимости между функциями (и, как следствие, модулями) разбора: при добавлении в систему новых типов достаточно зарегистрировать соответствующий распознаватель. В противном случае (когда функциональность по распознаванию сосредоточена в функциях разбора) требуется вносить изменения в код разборщиков. Отметим также, что распознаватели могут быть размещены в любом модуле разбора.

Предложенные модификации ликвидируют перечисленные ранее недостатки базовой модели. Однако возникает новая проблема, связанная со сборкой сеансов: при добавлении данных соответствующий сеанс необходимо однозначно идентифицировать. В качестве примера рассмотрим протокол IPv4. Этот протокол используется для передачи данных между узлами сети, которые идентифицируются посредством уникальных IP-адресов. Однако эта уникальность имеет место только внутри определенной IP-подсети. В сетевом трафике, как правило, представлено множество таких подсетей. Кроме того, пакеты одной IP-сети могут быть вложены в пакеты другой IP-сети. Таким

образом, для идентификации IP-сеанса в рамках глобальной сети недостаточно пары IP-адресов. Посредством протокола TCP происходит передача данных между парой портов – отправителя и получателя. То есть для идентификации TCP-сеанса необходимо задать значения портов. Обобщая, можно сказать, что:

- для каждого протокола понятие сеанса различно, то есть имеют место различные *типы сеансов*
- сетевой протокол определяет набор идентификационных характеристик (ключ) для сеансов этого протокола
- уникальность идентификационных характеристик сеанса, определяемых протоколом, имеет место в рамках некоторого *контекста*

Для IP-сеанса таким контекстом является некоторая IP-подсеть, а для TCP-сеанса – IP-сеанс в рамках IP-подсети. Заметим, что чем выше уровень протокола в модели OSI, тем большим набором идентификационных характеристик должен обладать сеанс этого протокола в рамках глобальной сети. Для TCP-сеанса, в частности, этот набор включает характеристики IP-сеанса, а также сам IP-сеанс. Фактически имеет место дерево контекстов, где каждая вершина определяет набор характеристик для соответствующих сеансов, причем этот набор содержит все характеристики родительской вершины и хотя бы одну дополнительную (рис. 8). Для корня дерева набор таких характеристик пуст.

С каждой вершиной дерева контекстов будем ассоциировать набор *экземпляров контекста*. Экземпляр контекста – это контекст, для которого известны значения всех идентификационных характеристик.

Понятие потока, введенное ранее, в полной мере описывает сеанс, причем тип сборки потока (по определению) является типом сеанса. Сборка (и последующее хранение) потока осуществляется в рамках экземпляра контекста.

Для описания контекстов расширим введенное ранее понятие типа, добавив в него признак необходимости создания контекста. Если тип обладает данным признаком, то перед разбором блока соответствующего типа ядро системы осуществит *переключение* контекста (в частности, перед разбором заголовка протокола IPv4). В каждый момент проведения анализа ровно один контекст является *активным*. Переключение заключается в смене активного контекста – если нужного контекста не существует к моменту переключения, то он будет создан. Будем называть *типом контекста* тип блока, разбор которого потребовал создания этого контекста.

Менее формально, контексты и экземпляры контекстов предназначены для группировки блоков одного уровня вложенности. Фактически речь идет о классификации трафика (подробнее о методах классификации в статье [16]). Критерий группировки определяется типом (контекста) и задается функцией, которая на основе анализа содержимого блока относит его к конкретной

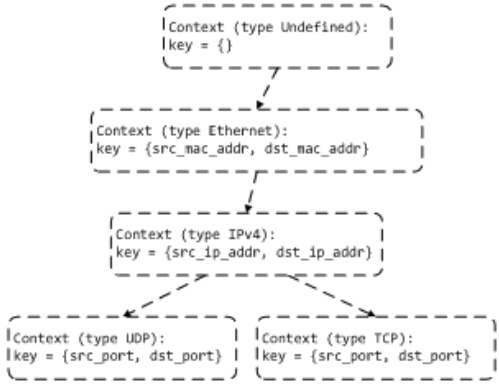

Рис. 8 – Пример дерева контекстов.

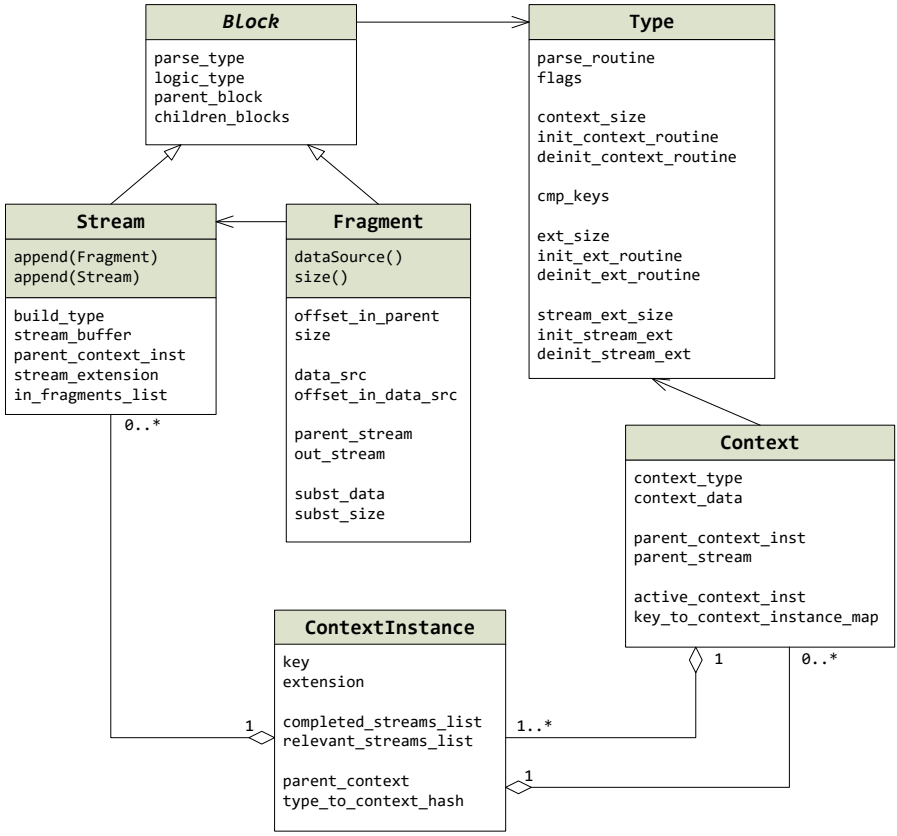

Рис. 9 – Расширенная модель представления данных.

группе (каждый экземпляр в рамках некоторого контекста описывает одну такую группу).

Сборка потоков проводится в рамках одного экземпляра контекста. Это значит, что блоки, принадлежащие разным экземплярам контекста, не могут стать частями одного и того же потока.

Объектная модель, учитывающая все перечисленные особенности, представлена на рис. 9. Подчеркнем, что на уровне блоков поддерживается семантика (атрибут logic_type): в соответствии с типом (атрибут flags) блок является либо *структурой* (по аналогии с типом "struct" языка Си), либо *последовательностью*. Соответственно, дочерние блоки в первом случае называются *полями*, а во втором – *элементами последовательности*. Каждое поле характеризуется именем, каждый элемент последовательности – порядковым номером. Расширенная модель, таким образом, полностью покрывает функционал базовой модели.

Также добавим, что в рамках экземпляра контекста потоки идентифицируются посредством ключа (атрибут stream_extension), размер (атрибут stream_ext_size) и структура которого определяются типом соответствующего контекста.

5. Заключение

В статье предложена объектная модель представления данных, применяемая в рамках инструментов оффлайн- и онлайн-анализа. В отличие от базовой модели, лежащей в основе большинства существующих инструментов анализа сетевого трафика, расширенная модель предоставляет функциональность по сборке сеансов, поддерживает работу с модифицированными данными, позволяет вести разработку модулей разбора независимо. Схожую модель предлагает инструмент Wireshark, однако она обладает существенным недостатком: сборка сеансов целиком ложится на плечи разработчика модулей разбора. На уровне ядра Wireshark отсутствуют аналоги таких понятий как поток, контекст и экземпляр контекста: для каждого модуля (при необходимости проведения сборки) их нужно создавать отдельно. В результате, код разборщиков оказывается перегруженным логикой сборки. С другой стороны, предложенная в статье модель вводит абстракции для сборки сеансов на уровне ядра, что позволяет существенно упростить (и сократить) код разборщиков и избежать дополнительных ошибок.

Список литературы

[1]. P. Tsankov, M. T. Dashti, D. Basin. SECFUZZ: Fuzz-testing Security Protocols // Proceedings of the 7th International Workshop on Automation of Software Test (AST 2012), pp. 1-7, 2012

[2]. Н. В. Пакулин, В. З. Шнитман, А. В. Никешин. Автоматизация тестирования соответствия для телекоммуникационных протоколов // Труды Института

системного программирования РАН, том 26, выпуск 1, 2014, стр. 109-148. DOI: 10.15514/ISPRAS-2014-26(1)-4

[3]. Karen Scarfone, Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS) // National Institute of Standards and Technology Special Publication 800-94, 127 pages, February 2007

[4]. Ю. В. Маркин, А. С. Санаров. Обзор современных инструментов анализа сетевого трафика // Препринты ИСП РАН, №27, 2014

[5]. Рекомендация МСЭ-Т Y.2770, Требования к углубленной проверке пакетов в сетях последующих поколений, издание 1.0, 20.11.2012

[6]. Snort. http://www.snort.org/, дата обращения 07.10.2015

[7]. Wireshark. http://www.wireshark.org/, дата обращения 07.10.2015

[8]. The Bro Network Security Monitor. http://www.bro.org/, дата обращения 07.10.2015

[9]. IETF RFC 791. Information Sciences Institute, Internet Protocol, September 1981

[10]. IETF RFC 5246. T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, August 2008

[11]. Н. В. Пакулин, В. З. Шнитман, А. В. Никешин. Разработка тестового набора для верификации реализаций протокола безопасности TLS // Труды Института системного программирования РАН, том 23, 2012, стр. 387-404. DOI: 10.15514/ISPRAS-2012-23-22

[12]. А. В. Никешин, Н. В. Пакулин, В. З. Шнитман. Тестирование реализаций клиента протокола TLS // Труды Института системного программирования РАН, том 27, выпуск 2, 2015, стр. 145-160. DOI: 10.15514/ISPRAS-2015-27(2)-9

[13]. IETF RFC 4251. T. Ylonen, C. Lonvick, The Secure Shell (SSH) Protocol Architecture, January 2006

[14]. IETF RFC 791. Information Sciences Institute, Transmission Control Protocol, September 1981

[15]. IETF RFC 768. J. Postel, User Datagram Protocol, August 1980

[16]. F. Risso, A. Baldini, M. Baldi, P. Monclus, O. Morandi, Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation // IEEE International Conference on Communications (ICC 2008), Beijing (China), pp. 5869-5875, May 2008

Model of Data Handling for In-Depth Analysis of Network Traffic★

[1]*A. I. Get'man <thorin@ispras.ru>*
[1, 2, 3, 4]*V.P. Ivannikov <ivan@ispras.ru>*
[1]*Yu. V. Markin <ustas@ispras.ru>*
[1, 2]*V. A. Padaryan <vartan@ispras.ru>*
[1]*A. Yu. Tikhonov <fireboo@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia*
[2]*Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation*
[3]*Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[4]*Higher School of Economics, National Research University, 20 Myasnitskaya Ulitsa, Moscow 101000, Russia*

Abstract. The article suggests a new object model of data for in-depth analysis of network traffic. In contrast to the model used by most existing network analyzers, such as Wireshark or Snort, the core of our model supports data streams reassembling and next processing of them. Analysis continues even in case of loss of individual packets. The model supports both stateless and statefull network protocols. State of protocol machine may be stored in a special memory location related to each connection of relevant type. The article stated the requirements for network traffic analysis tools. A high speed data processing in resource-limited environment is the main requirement for online systems. Offline analyzer operates with a network trace of the fixed size, so the processing speed is not so important. It becomes possible to visualize the data structure disassembled. Offline analyzer also traces how network streams formed from packets. The model provides an interface for parsers implemented in the form of dynamic link libraries. It also provides a convenient universal mechanism for binding parsers so one can develop parsers independently. This is achieved through the use of special functions (recognizers) allowing for the data itself to determine which parser should be used. It is crucial for parsers to be compatible with both online and offline analyzers. Our model also provides processing of modified, e.g. compressed or encrypted, data. Unlike Snort the model supports nested tunneling protocols. Actually it forms the basis of the infrastructure for in-depth analysis of network traffic.

Keywords: network traffic analysis; data stream assembling; model of data; recognition of data.

DOI: 10.15514/ISPRAS-2015-27(4)-1

★ This work is supported by RFBR grant 15-07-07652 A

For citation: Get'man A.I., Ivannikov V.P., Markin Yu.V., Padaryan V.A., Tikhonov A.Yu. Model of Data Handling for In-Depth Analysis of Network Traffic. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 5-22 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-1.

References

[1]. P. Tsankov, M. T. Dashti, D. Basin. SECFUZZ: Fuzz-testing Security Protocols // Proceedings of the 7th International Workshop on Automation of Software Test (AST 2012), pp. 1-7, 2012

[2]. A. V. Nikeshin, N. V. Pakulin, V. Z. Shnitman. Avtomatizatsiya testirovaniya sootvetstviya dlya telekommunikatsionnykh protokolov [Automation of conformance testing for communication protocols] // Trudy ISP RAN [The Proceedings of ISP RAS], 2014, vol. 26, no. 1, pp. 109-148 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-4

[3]. Karen Scarfone, Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS) // National Institute of Standards and Technology Special Publication 800-94, 127 pages, February 2007

[4]. Yu. V. Markin, A. S. Sanarov. Obzor sovremennykh instrumentov analiza setevogo trafika [The modern network traffic analyzers overview] // Preprinty ISP RAN [Preprints of ISP RAS], №27, 2014 (in Russian)

[5]. Recommendation ITU-T Y.2770, Requirements for deep packet inspection in next generation networks, edition 1.0, 2012.11.20

[6]. Snort. http://www.snort.org/, access date: 2015.10.07

[7]. Wireshark. http://www.wireshark.org/, access date: 2015.10.07

[8]. The Bro Network Security Monitor. http://www.bro.org/, access date: 2015.10.07

[9]. IETF RFC 791. Information Sciences Institute, Internet Protocol, September 1981

[10]. IETF RFC 5246. T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, August 2008

[11]. A. V. Nikeshin, N. V. Pakulin, V. Z. Shnitman. Razrabotka testovogo nabora dlya verifikatsii realizatsii protokola bezopasnosti TLS [Creation of a test suite for verification of the TLS security protocol] // Trudy ISP RAN [The Proceedings of ISP RAS], 2012, vol. 23, pp. 387-404 (in Russian). DOI: 10.15514/ISPRAS-2012-23-22

[12]. A. V. Nikeshin, N. V. Pakulin, V. Z. Shnitman. Testirovanie realizatsii klienta protokola TLS [TLS clients testing] // Trudy ISP RAN [The Proceedings of ISP RAS], 2015, vol. 27, no. 2, pp. 145-160 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-9

[13]. IETF RFC 4251. T. Ylonen, C. Lonvick, The Secure Shell (SSH) Protocol Architecture, January 2006

[14]. IETF RFC 791. Information Sciences Institute, Transmission Control Protocol, September 1981

[15]. IETF RFC 768. J. Postel, User Datagram Protocol, August 1980

[16]. F. Risso, A. Baldini, M. Baldi, P. Monclus, O. Morandi, Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation // IEEE International Conference on Communications (ICC 2008), Beijing (China), pp. 5869-5875, May 2008

Метод поиска уязвимости форматной строки

[1]И.А. Вахрушев <ilia.vahrushev@ispras.ru>
[1]В.В. Каушан <korpse@ispras.ru>
[1, 2]В.А. Падарян <vartan@ispras.ru>
[1]А.Н. Федотов <fedotoff@ispras.ru>
[1] Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2]Московский государственный университет имени М.В. Ломоносова,
119991 ГСП-1 Москва, Ленинские горы, 2-й учебный корпус, факультет ВМК

Аннотация. В статье рассматривается метод поиска уязвимостей форматной строки в исполняемом бинарном коде. Предлагаемый метод использует символьное выполнение и применяется к бинарным файлам программ, не требуя какой-либо отладочной информации. Метод был реализован в виде программного инструмента. Возможности инструмента были продемонстрированы на известных уязвимостях приложений, работающих под управлением ОС Linux.

Ключевые слова: уязвимость форматной строки; бинарный код; эксплуатация уязвимостей; динамический анализ; символьное выполнение.

DOI: 10.15514/ISPRAS-2015-27(4)-2

Для цитирования: Вахрушев И.А., Каушан В.В., Падарян В.А., Федотов А.Н. Метод поиска уязвимости форматной строки. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 23-38. DOI: 10.15514/ISPRAS-2015-27(4)-2.

1. Введение

Обеспечение безопасности компьютерных программ становится все более злободневным вопросом. Наследственное ПО, разработанное без учета современных требований безопасности, продолжает эксплуатироваться, довольствуясь фрагментарными исправлениями. Да и современные технологии разработки безопасного ПО не решают проблемы безопасности в полной мере. В частности, несмотря на развитие графических интерфейсов, программы продолжают пользоваться функциями форматного вывода.

Форматный вывод используется для ведения системных журналов, в сообщениях об ошибках и т.д. Еще в работах 90-х годов [1] было показано, что неправильная работа с форматной строкой может привести к серьезным

уязвимостям в безопасности ПО, таким как выполнение произвольного кода, повышение привилегий, а также утечка чувствительных данных. Уязвимость форматной строки позволяет злоумышленнику целенаправленно перезаписывать определенные участки оперативной памяти. Одним из таких участков может быть ячейка памяти на стеке, которая содержит адрес возврата из функции. Перезапись адреса возврата приводит к перехвату потока управления и выполнению произвольного кода, размещённого в памяти. Размещение в стеке перед адресом возврата специального значения «канарейки», призванного выявлять ошибки переполнение буфера на стеке, не защищает от уязвимости форматной строки, так как перезаписывается не диапазон адресов памяти, а только адрес возврата.

Поиск ошибок и уязвимостей можно проводить как на уровне исходных текстов [2], так и в бинарном коде [3]. Поиск ошибок на уровне бинарного кода имеет преимущества не только, когда исходные тексты исследуемого ПО недоступны. Наиболее важная причина – исполняемый код позволяет гораздо точней оценить критичность ошибки, может ли она эксплуатироваться или нет [4], т.к. на уровне исходного кода еще не известно точное размещение переменных в памяти.

Существуют различные методы поиска ошибок и уязвимостей в бинарном коде. Активно изучаемым подходом является символьное выполнение [5], получившее «второе рождение» в последние годы [6].

Во время символьного выполнения конкретные значения переменных заменены символьными значениями, операции над переменными порождают формулы над символьными значениями и константами. На практике символьные значения задаются на основе входных данных программы, получаемых из различных источников: сеть, файлы, аргументы командной строки, переменные окружения и т.д. Каждое условное ветвление, на которое влияют входные данные, порождает дополнительное уравнение, характеризующее прохождение потока управления по определённой ветке. Совокупность таких уравнений, описывающих прохождение некоторого пути выполнения, называется предикатом пути. Этот набор уравнений подаётся на вход SMT-решателю, где символьные переменные выступают в качестве неизвестных величин. Результат решения – набор входных данных, при обработке которого поток управления программы пойдёт по заданному пути.

Для выявления уязвимости к предикату пути добавляются дополнительные уравнения, описывающие срабатывание уязвимости. Если полученная система уравнений совместна, то результатом её решения является эксплойт – набор входных данных, эксплуатирующий уязвимость. Из-за погрешностей в моделировании уязвимости необходимо дополнительно проверить работоспособность полученного эксплойта, запустив с ним программу.

Существует ряд инструментов, например, AEG [7], MAYHEM [8], разработанных в ведущих университетах США, которые решают задачу поиска уязвимости форматной строки. К сожалению, эти инструменты

недоступны. Существующие системы символьного выполнения, находящиеся в открытом доступе, например, S2E [9], предоставляют только инфраструктуру перебора состояний, инструменты поиска уязвимостей в этом случае приходится реализовывать самостоятельно.

Работы, проводимые в ИСП РАН, используют другой подход: вместо перебора состояний детально исследуется ограниченное число трасс, как правило, это одна трасса, обеспечивающая приемлемое покрытие кода. Был предложен метод символьной интерпретации трассы, нацеленный на поиск и оценку критичности сработавших ошибок работы с памятью. В своем развитии метод получил возможность анализировать подсемейство трасс, абстрактно интерпретируя длины буферов памяти [10]. Основываясь на результатах предыдущих работ, был разработан метод поиска и эксплуатации важного типа уязвимостей, а именно – уязвимости форматной строки.

Статья организована следующим образом. Во втором разделе описан механизм эксплуатации уязвимости форматной строки. В третьем разделе описан предлагаемый метод и схема работы инструмента, реализующего его. Результаты практического применения приведены в четвёртом разделе. В последнем, пятом разделе обсуждаются полученные результаты и рассматриваются перспективные направления исследований.

2. Эксплуатация уязвимости форматной строки

Функции форматного вывода, такие как `printf`, `fprintf`, `vfprintf`, `syslog`, используются для вывода значений разных типов, отформатированных согласно заданному шаблону (форматной строке). Шаблон представляет собой строку, содержащую управляющие последовательности (спецификаторы). Уязвимость форматной строки возникает в случаях, когда разработчик программы в качестве форматной строки передаёт буфер, зависящий от входных данных.

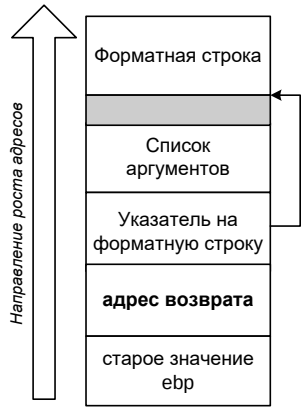

Рис. 1. Состояние стека в момент вызова функции форматного вывода.

На рис. 1 показан кадр стека функции форматного вывода `printf`. Перед вызовом функции в стеке размещается (в обратном порядке) список дополнительных аргументов и указатель на форматную строку. Во время вызова на стеке размещается адрес возврата в вызывающую функцию, после чего вызываемая функция может сохранять некоторые регистры и выделять память для локальных переменных.

Множество используемых дополнительных аргументов функции определяется спецификаторами форматной строки. При обработке форматной строки аргументы считываются по очереди, начиная с аргумента, следующего непосредственно за указателем на форматную строку. Если в форматной строке количество спецификаторов превосходит количество фактически переданных дополнительных аргументов, то вышележащие ячейки стека будут интерпретированы как потенциальные аргументы функции `printf`. Некоторые функции форматного вывода (например, `vfprintf`) в качестве списка аргументов принимают структуру `va_list`. В этом случае адреса потенциальных аргументов вычисляются с помощью содержимого этой структуры.

Помимо спецификаторов, позволяющих вывести значения в поток вывода, в функциях форматного вывода поддерживается спецификатор `%n`, который позволяет записывать в память по адресу, переданному в качестве аргумента, значение, равное выведенному в поток количеству символов. Поскольку адрес перезаписываемой ячейки памяти передаётся в качестве аргумента функции форматного вывода, этот спецификатор позволяет записывать данные в память по контролируемому адресу. Используя спецификатор несколько раз можно перезаписать произвольное количество ячеек памяти. Ячейки памяти, содержащие контролируемые адреса, должны находиться на стеке среди потенциальных аргументов. Как правило, эти аргументы размещаются внутри форматной строки. В этом случае, форматная строка должна располагаться на стеке.

С помощью спецификатора `%n` злоумышленник может перезаписать адрес возврата из функции указателем на произвольный код, что приведёт к выполнению этого кода и нарушению безопасности. Этот код также может быть размещён внутри форматной строки. Таким образом, можно сформировать такую форматную строку, обработка которой приведёт к выполнению кода, заданного злоумышленником.

При использовании спецификатора `%n` в память по заданному адресу записывается четырёхбайтное значение, равное количеству символов, выведенных в поток вывода. Это значение можно контролировать с помощью форматных спецификаторов, таких как: `%x`, `%d`. Спецификаторы форматной строки можно использовать с параметром ширины. Значение параметра ширины непосредственно влияет на значение, записываемое в память при обработке спецификатора `%n`. Таким образом, можно указать такое значение ширины, чтобы в память по заданному адресу записывалось требуемое

значение. В некоторых функциях форматного вывода может применяться ограничение на величину параметра ширины. В этом случае возможно использование нескольких спецификаторов с уменьшенным до допустимого значения параметром ширины. Помимо того, можно уменьшить величину записываемых значений, если перезаписывать заданную ячейку памяти по частям. Например, спецификатор `%hn` позволяет перезаписывать только два байта вместо четырёх. Уменьшение записываемых значений позволяет сократить объем выводимых текстовых данных, и как следствие – сократить размер форматной строки.

Адреса перезаписываемых ячеек памяти расположены выше по стеку относительно кадра стека функции форматного вывода и расположены среди аргументов этой функции. Аргументы функции считываются последовательно по мере обработки спецификаторов форматной строки, поэтому для доступа к нужному аргументу перед обработкой спецификатора `%n` необходимо предварительно обработать некоторое количество других спецификаторов вывода. Так как на стеке могут располагаться произвольные значения аргументов, следует использовать спецификаторы вывода с параметром ширины, позволяющим выводить фиксированное количество символов вне зависимости от значений аргументов. В качестве такого спецификатора можно использовать спецификатор `%8x`. В операционных системах Linux в спецификаторах форматной строки поддерживается возможность доступа к произвольному аргументу по его порядковому номеру (используется символ '`$`') [11]. Это значительно упрощает задачу доступа к нужному аргументу и ощутимо сокращает размеры форматной строки.

Уязвимость форматной строки рассматривается как обособленный, крайне важный тип уязвимостей. Практическое применение этой уязвимости – выборочная перезапись одного машинного слова, содержащего адрес. Как правило, это адрес возврата из функции, его изменение позволяет перехватить поток управления. Относительно недавно было показано [12], что уязвимость форматной строки удобна для реализации другого автоматизируемого типа атаки – перехвата потока данных. В результате успешной атаки злоумышленник получает доступ к чувствительной информации, как пользовательской, так и служебной (GOT, адреса стека, "канарейки", указатели и др.). Доступ к служебным данным позволяет получить значения рандомизированных адресов, что открывает дальнейшие возможности по конструированию работоспособных эксплойтов для данного экземпляра программы, содержащей уязвимость.

Важно отметить, что существующие механизмы защиты («канарейка», неисполняемый стек, ASLR, «безопасные» функции форматного вывода и др.) не способны противодействовать перехвату потока данных.

3. Схема работы

Представленные в данной статье исследования опираются на возможности среды анализа бинарного кода [13]. Основным предметом анализа являются трассы машинных инструкций, полученные в результате работы полносистемного эмулятора [14, 15]. Трассы содержат состояния регистров, информацию о прерываниях и взаимодействии с периферийными устройствами, что позволяет восстанавливать статико-динамическое представление [16] для всех работавших в системе программ и эффективно исследовать его свойства. Основным назначением среды анализа является автоматизация метода выделения алгоритмов из бинарного кода и повышение уровня представления этих выделенных алгоритмов.

Поиск уязвимостей форматной строки проводится по трассе выполнения анализируемой программы для каждого вызова функций форматного вывода. Поиск вызовов этих функций в трассе выполняется автоматически, рассматриваются только те вызовы, на форматную строку которых могут воздействовать входные данные. Для каждого анализируемого вызова функции из трассы отбираются инструкции, обрабатывающие соответствующие входные данные. Для отбора инструкций используется алгоритм динамического слайсинга трассы [17], дополненный анализом помеченных данных. В качестве входных данных выступают буферы, используемые для чтения файлов, получения данных из сети, а также содержащие параметры командной строки.

Для анализа машинных инструкций используется промежуточное представление Pivot [18], позволяющее унифицировано описывать операционную семантику инструкций различных процессорных архитектур. Промежуточное представление отвечает требованием SSA-формы, что позволяет значительно упростить анализ.

Поиск уязвимости форматной строки и генерация эксплойта разделена на пять последовательных этапов (рис.2).

Рис. 2. Декомпозиция метода на пять этапов.

Так как анализируются трассы выполнения программ, состоящих из последовательного набора машинных команд, то для начала работы алгоритма необходимо задать шаг трассы, начиная с которого в программе уже содержаться входные данные, полученные из внешнего источника. Также необходимо задать буфер памяти, в котором содержаться эти входные данные. Для поиска буфера с входными данными и нужного шага трассы, аналитик

может воспользоваться возможностями среды анализа бинарного кода, а именно – навигацией по вызовам функций. Как правило, программы используют для получения входных данных известные библиотечные функции. Заранее зная, какой источник входных данных (сеть, файлы, аргументы командной строки и т.п.) был использован, аналитик выбирает вызовы соответствующих библиотечных функций.

В результате работы алгоритма слайсинга формируется подтрасса, содержащая инструкции, которые участвуют в обработке входных данных. Точное выделение подтрассы позволяет рассматривать ограниченное число машинных команд, которые впоследствии будут оттранслированы в SMT-формулы, что в свою очередь позволяет ускорить решение результирующей системы уравнений. Начальному шагу работы алгоритма соответствует точка получения входных данных, а конечному – шаг трассы, на котором происходит вызов функции работы с форматной строкой.

Предикат пути строится по подтрассе и представляет собой набор уравнений на языке SMT [19]. Предикат пути используются для описания ограничений на входные данные, при прохождении потока управления от точки получения до вызова функции работы с форматной строкой. Каждая машинная инструкция подтрассы последовательно транслируется в промежуточное представление Pivot [18]. Затем по промежуточному представлению строятся SMT-формулы, описывающие работу машинной инструкции в целом. В результате трансляции всех инструкций подтрассы будет получен предикат пути. Более подробно применяющийся способ трансляции инструкций в SMT-формулы описан в работе [4].

Для поиска потенциальной уязвимости проверяется наличие помеченных данных в буфере, передаваемом в качестве форматной строки. Если этот буфер содержит помеченные данные, конструируется содержимое форматной строки, обработка которой приведет к выполнению заданного шелл-кода. На рис. 3 представлена структура генерируемой форматной строки. В начале создаваемой форматной строки размещается шелл-код, затем размещаются спецификаторы форматного вывода, а в самом конце строки расположен адрес перезаписываемой ячейки памяти.

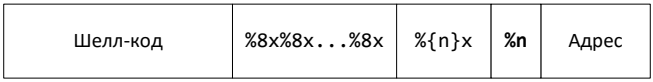

Рис. 3. Структура форматной строки, используемой в эксплойте.

Форматная строка представляет собой нуль-терминированную строку, следовательно, размещение нулевых байтов внутри строки недопустимо. Если адрес перезаписываемой ячейки памяти в старшем байте содержит ноль, то его можно совместить с терминатором строки при условии, что используется порядок байтов «от младшего к старшему». Наличие других нулевых байтов в адресе не допускается.

Последовательность спецификаторов `%8x` готовит доступ к адресу во время обработки спецификатора `%n`. Спецификатор `%{n}x` с параметром ширины `n` используется для достижения требуемого значения суммарной ширины. Спецификатор `%n` осуществляет запись суммарной ширины по адресу, записанному в конце строки. В процессе обработки форматной строки функция поддерживает указатель на текущий аргумент в списке потенциальных аргументов, в также указатель на текущий обрабатываемый символ форматной строки. При обработке спецификатора `%n` указатель на текущий аргумент должен указывать на ячейку, содержащую адрес перезаписываемой ячейки памяти. Для изменения этого указателя необходимо обработать некоторое количество спецификаторов `%8x`. Каждая обработка спецификатора `%8x` сдвигает указатель на текущий аргумент на 4 байта вверх по стеку, а указатель на текущий обрабатываемый символ форматной строки – на 3 байта (количество символов, занимаемых спецификатором форматной строки). Требуемое количество спецификаторов `%8x` подбирается, исходя из следующих соображений. Пусть перед обработкой последовательности спецификаторов `%8x` значение указателя на текущий аргумент равно *ArgStart*, а значение указателя на текущий обрабатываемый символ форматной строки равно *FmtStart*. Соответственно, в момент обработки спецификатора `%n` значение указателя на текущий аргумент равно *ArgEnd*, а на текущий обрабатываемый символ – *FmtEnd*. Структура списка аргументов и размещение указателей изображены на рис. 4.

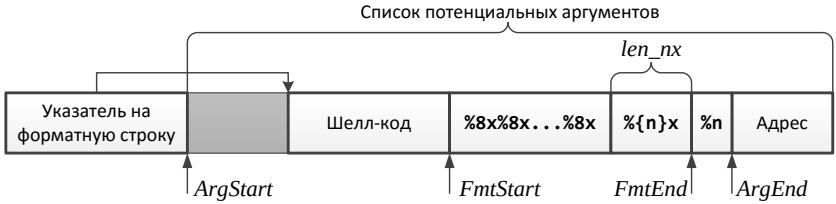

Рис. 4. Структура списка аргументов.

Пусть форматная строка содержит k спецификаторов %8x. В результате обработки форматных спецификаторов указатели должны соотноситься следующим образом:

$$ArgEnd = ArgStart + (k+1)*4,$$
$$FmtEnd = FmtStart + k*3 + len\_nx,$$
$$ArgEnd = FmtEnd + 2.$$

Первое уравнение описывает сдвиг указателя на текущий аргумент при обработке спецификаторов. Так как после последовательности спецификаторов `%8x` располагается спецификатор `%{n}x`, количество обработанных спецификаторов равно *(k+1)*. Второе уравнение отражает сдвиг

указателя на текущий обрабатываемый символ форматной строки с учётом обработки спецификатора `%{n}x`, длина записи которого обозначена как *len_nx*. Третье уравнение описывает требуемое состояние указателей на момент выполнения спецификатора `%n`. В этот момент указатели должны различаться на 2 байта – длину спецификатора `%n`. С учётом данных соотношений, требуемое количество спецификаторов `%8x` вычисляется по формуле:

$$k = FmtStart - ArgStart - 2 + len\_nx.$$

При размещении спецификатора `%{n}x` значение ширины n подбирается таким образом, чтобы в совокупности с шириной остальных спецификаторов, а также длиной шелл-кода получить требуемое суммарное значение ширины, которое в дальнейшем будет записано в память при обработке спецификатора `%n`. Завершается генерация форматной строки размещением спецификатора `%n`, вслед за которым размещается адрес перезаписываемой ячейки памяти. Если расстояние между ячейкой памяти, в которой будет лежать адрес, и ячейкой памяти первого элемента из списка потенциальных аргументов, не кратно 4, то в форматную строку после шелл-кода добавляется необходимое количество произвольных байтов.

При генерации эксплойта, перезаписывающего адрес возврата, в операционной системе Windows адрес перезаписываемой ячейки часто содержит нулевой байт в старшем разряде, поэтому его можно размещать только в конце форматной строки. В операционной системе Linux адреса ячеек стека не содержат нулевой байт в старшем разряде, поэтому адрес перезаписываемой ячейки, если он не содержит нулевых байтов, можно размещать в любом месте форматной строки. Также использование таких адресов позволяет размещать более одного адреса в форматной строке и перезаписывать память по частям. Перезапись ячеек памяти по частям позволяет использовать меньшие значения суммарной ширины, что, в свою очередь, позволяет использовать меньшие значения ширины в спецификаторах `%{n}x`. Так как в функциях библиотеки libc ОС Linux поддерживается возможность обращения к произвольному аргументу в спецификаторе форматной строки, целесообразно использовать эту возможность вместо генерации последовательности спецификаторов `%8x`.

После генерации форматной строки производится проверка того, что размер сгенерированной форматной строки не превосходит размер буфера, контролируемого пользователем. Размер этого буфера определяется с помощью вычисления размера непрерывной области помеченных данных, передаваемых в функцию форматного вывода качестве форматной строки.

Если проверка выполнена успешно, выполняется построение уравнений, описывающих передачу сгенерированной форматной строки в функцию форматного вывода. Эти уравнения объединяются с предикатом пути и описывают состояние программы, в котором активируется уязвимость форматной строки. Полученная система уравнений подаётся на вход SMT-

решателю. Если система уравнений совместна, её решением является набор входных данных, обработка которого приводит к срабатыванию уязвимости. Выполнение программы на полученном наборе входных данных непосредственно подтверждает наличие уязвимости.

4. Реализация метода и результаты практического применения

Предложенный метод был реализован и интегрирован в уже существующий модуль-расширение среды анализа бинарного кода [4]. Для решения систем уравнений используется решатель Z3 [20], так как он обеспечивает высокую скорость работы по сравнению с другими решателями. Для приложений под управлением ОС Linux применяется возможность перезаписи ячейки памяти по частям с помощью спецификатора `%hn`.

Разработанный инструмент был опробован на нескольких примерах известных уязвимостей. Для запуска приложений использовалась операционная система Arch Linux. В табл. 1 приведены основные результаты работы реализованного алгоритма: размер слайса обработки входных данных, размер блока входных данных, общее время работы алгоритма. Следует отметить, что время работы SMT-решателя не превышало одной секунды на всех примерах. Кроме того, в таблице приведены названия и версии анализируемых приложений, используемая функция форматного вывода, а также тип источника входных данных (аргументы командной строки или сетевой сокет).

Табл.1. Результаты работы алгоритма генерации эксплойта для уязвимости форматной строки.

Приложение	Функция	Размер слайса	Размер данных, байт	Общее время работы, с	Тип входных данных
socat-1.4	syslog	2847	254	25	Арг.
orzhttpd-r140	fprintf	7006	254	49	Сеть
sharutils-4.2.1	vfprintf	1970	255	52	Арг.
gpsd-2.7	syslog	320	255	1	Сеть

Для успешной эксплуатации уязвимости были отключены механизмы защиты, которые препятствуют выполнению внедренного кода. Исследуемые приложения компилировались с флагами компилятора `-z execstack -D_FORTIFY_SOURCE=0`. Флаг `–z execstack` даёт возможность выполнять код, располагающийся на стеке. Директива `-D_FORTIFY_SOURCE=0` отключает использование безопасных функций. Также в операционной системе Arch Linux был отключен механизм рандомизации адресного пространства (ASLR). Стоит отметить, что отключение «канарейки» не потребовалась, так как

перезапись адреса возврата из функции происходит, не затрагивая «канарейку».

Предложенный метод обладает следующими ограничениями. Генерируемая форматная строка не должна быть большей длины, чем форматная строка, обработка которой зафиксирована в трассе. Кроме того, размещаемый внутри форматной строки шелл-код не должен содержать символов "%". Также, при записи больших значений в память могут использоваться большие значения модификатора ширины. При обработке таких спецификаторов на поток вывода будет выведено большое количество данных. В некоторых случаях это может привести к срабатыванию защитных проверок, ошибкам выделения памяти, ошибкам файловой системы и другим аварийным ситуациям, что, в свою очередь, приводит к преждевременному завершению обработки форматной строки.

Иногда на момент вызова функции форматного вывода предикат пути накладывает существенные ограничения на содержимое форматной строки. В приложениях под управлением операционной системы Windows адрес перезаписываемой ячейки памяти содержит нулевой байт. Это приводит к необходимости размещения адреса в конце форматной строки, что в совокупности с ограничениями на последние байты форматной строки приводит к несовместности системы ограничений. В связи с этим данный алгоритм не позволил провести эксплуатацию нескольких приложений под управлением ОС Windows.

5. Заключение

В статье представлен метод поиска уязвимости форматной строки. Метод основан на символьном выполнении бинарных трасс полученных с полносистемного эмулятора. Метод был реализован в рамках среды анализа бинарного кода и позволяет генерировать эксплойты для уязвимости форматной строки. При помощи данного метода успешно удалось сгенерировать рабочие эксплойты для четырёх приложений, в которых содержались известные уязвимости форматной строки.

Наиболее близкими являются работы коллектива из университета Карнеги-Меллон. В работе AEG [7] с помощью анализа исходного кода программы производится поиск двух видов уязвимостей: уязвимость переполнения буфера и уязвимость форматной строки. Затем уже по бинарному коду программы генерируется эксплойт для найденной уязвимости. Работа MAYHEM [8] является продолжением работы AEG и описывает метод генерации эксплойтов на основе анализа только бинарного кода. Предлагаемый инструмент имеет возможность перебирать различные траектории выполнения во время динамического символьного выполнения. К сожалению, разработанные этим коллективом инструменты недоступны. Стоит отметить, что в этих работах для успешной эксплуатации уязвимостей также требовалось отключение различных механизмов защиты.

Инструмент CRAX [21] использует инфраструктуру символьного выполнения S2E [8] и позволяет обнаруживать и эксплуатировать уязвимости форматной строки. В работе [22] также предложен метод поиска уязвимости форматной строки, но эксплуатация найденной уязвимости не производится.

Среди дальнейших направлений исследований можно выделить эксплуатацию уязвимостей с использованием перехвата потока данных [12], а не потока управления. В настоящее время повсеместно применяются механизмы защиты, препятствующие перехвату потока управления. Среди таких механизмов можно отметить DEP, ASLR, а также использование безопасных функций. В современных операционных системах, как правило, DEP и ASLR включены по умолчанию, а современные компиляторы используют безопасные функции при сборке программ. Таким образом, большая часть современных программ достаточно защищена от перехвата потока управления. Однако в некоторых случаях защита намеренно отключается разработчиком, а главное, указанные механизмы не защищают программу от перехвата потока данных. Посредством перехвата потока данных можно реализовать такие атаки, как утечка чувствительных данных, повышение привилегий и т.д. На данный момент защиты, которые препятствуют изменению потока данных, вносят значительное замедление в работу программ, поэтому их применение не всегда целесообразно.

Ещё одним перспективным направлением является адаптация существующих алгоритмов к архитектурам ARM и MIPS. Это позволит расширить круг устройств, для которых можно проводить анализ. Помимо настольных компьютеров, анализ сможет применяться к мобильным устройствам, планшетам и сетевым маршрутизаторам. Адаптация разработанных ранее алгоритмов потребует определенных усилий из-за особенностей упомянутых выше архитектур. Например, в архитектуре ARM некоторые функции могут сохранять адрес возврата на регистре, а не на стеке, что существенно осложняет перехват потока управления.

Список литературы

[1]. B.P. Miller; L. Fredriksen; B So. An Empirical Study of the Reliability of UNIX Utilities. // Commun. ACM. – 1990. – No 33.

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

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

[4]. Падарян В.А., Каушан В.В., Федотов А.Н. Автоматизированный метод построения эксплойтов для уязвимости переполнения буфера на стеке. // Труды Института

системного программирования РАН, том 26, выпуск 3, 2014, стр. 127-144. DOI: 10.15514/ISPRAS-2014-26(3)-7.

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

[6]. C. Cadar, K. Sen. Symbolic Execution for Software Testing: Three Decades Later. // Commun. ACM. – 2013. – No 56.

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

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

[9]. 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.

[10]. Каушан В.В., Мамонтов А.Ю., Падарян В.А., Федотов А.Н. Метод выявления некоторых типов ошибок работы с памятью в бинарном коде программ. // Труды Института системного программирования РАН, том 27, выпуск 2, 2015, стр. 105-126. DOI: 10.15514/ISPRAS-2015-27(2)-7

[11]. Ericson J. Hacking: The Art of Exploitation. 2nd Edition. // No Starch Press, 2008, pp.167-183.

[12]. Hong Hu, Zheng Leong Chua, Sendroiu Adrian, Prateek Saxena, Zhenkai Liang. Automatic Generation of Data-Oriented Exploits. // 24th USENIX Security Symposium 2015

[13]. А.Ю. Тихонов, А.И. Аветисян. Комбинированный (статический и динамический) анализ бинарного кода. // Труды Института системного программирования РАН, том 22, 2012, стр. 131-152. DOI: 10.15514/ISPRAS-2012-22-9

[14]. 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.

[15]. 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, выпуск 1, 2014, стр. 277-296. DOI: 10.15514/ISPRAS-2014-26(1)-9.

[16]. 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, выпуск 1, 2014, стр. 251-276. DOI: 10.15514/ISPRAS-2014-26(1)-8.

[17]. 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

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

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

[20]. Nikolaj Bjørner, Leonardo de Moura. Z3: Applications, Enablers, Challenges and Directions // Sixth International Workshop on Constraints in Formal Verification Grenoble, 2009.

[21]. Huang S. K. et al. Software Crash Analysis for Automatic Exploit Generation on Binary Programs // Reliability, IEEE Transactions on. – 2014. – Т. 63. – №. 1. – С. 270-289.
[22]. Wu B. et al. Directed symbolic execution for binary vulnerability mining // Electronics, Computer and Applications, 2014 IEEE Workshop on. – IEEE, 2014. – С. 614-617.

Search Method for Format String Vulnerabilities

[1]I.A. Vakhrushev <ilia.vahrushev@ispras.ru>
[1]V.V. Kaushan <korpse@ispras.ru>
[1, 2]V.A. Padaryan <vartan@ispras.ru>
[1]A.N. Fedotov <fedotoff@ispras.ru>
[1] Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia
[2]Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation

Abstract. In this paper search method for format string vulnerabilities is presented. Format string vulnerabilities can cause serious security problems providing ability to write an arbitrary value to an arbitrary location. Using this opportunity an attacker may hijack the control flow of a program and execute a malicious code. Besides, it is possible to bypass some protection mechanisms such as the stack canary due to exact overwriting of function return address. The method is based on dynamic analysis and symbolic execution. It is applied to program binaries without requiring debug information. We use dynamic analysis to find possible unsafe usage of format string function. If user controls data in a format string parameter, we consider that it is an unsafe usage of format string. Then a path predicate is build. The starting point of path predicate is a place where input data was received, the ending point is an unsafe format string function call. After building the path predicate we need to generate a special format string that provides a control flow hijack and a payload execution. By asserting such constraints on the format string parameter we are able to determine whether unsafe function usage is vulnerable or not. If the generated constraints are solvable, the method produces an exploit. We present a tool implementing this method. We used this tool to detect known vulnerabilities in Linux programs.

Keywords: format string vulnerability; binary code; vulnerability exploitation; dynamic analysis; symbolic execution.

DOI: 10.15514/ISPRAS-2015-27(4)-2
For citation: Vakhrushev I.A., Kaushan V.V., Padaryan V.A., Fedotov A.N. Search Method for Format String Vulnerabilities. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp.23-38 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-2.

References

[1]. B.P. Miller; L. Fredriksen; B So. An Empirical Study of the Reliability of UNIX Utilities. // Commun. ACM. – 1990. – No 33.

[2]. V.P. Ivannikov, A.A. Belevantsev, A.E. Borodin, V.N. Ignatiev, D.M. Zhurikhin, A.I. Avetisyan, M.I. Leonov. Staticheskij analizator Svace dlja poiska defektov v ishodnom kode programm. [Static analyzer Svace for finding of defects in program source code] // Trudy ISP RAN [The Proceedings of ISP RAS], vol. 26, issue 1, 2014, pp. 231-250 (in Russian). DOI: DOI: 10.15514/ISPRAS-2014-26(1)-7

[3]. 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).

[4]. V.A. Padaryan, V.V. Kaushan, A.N. Fedotov. Avtomatizirovannyj metod postroenija jeksplojtov dlja ujazvimosti perepolnenija bufera na steke. [Automated exploit generation method for stack buffer overflow vulnerabilities] // Trudy ISP RAN [The Proceedings of ISP RAS], vol. 26, issue 3, 2014, pp. 127-144 (in Russian). DOI: 10.15514/ISPRAS-2014-26(3)-7).

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

[6]. C. Cadar, K. Sen. Symbolic Execution for Software Testing: Three Decades Later. // Commun. ACM. – 2013. – No 56.

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

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

[9]. 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.

[10]. Kaushan V.V., Mamontov A.Yu., Padaryan V.A., Fedotov A.N. // Metod vyyavleniya nekotorykh tipov oshibok raboty s pamyat'yu v binarnom kode programm. [Memory violation detection method in binary code]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 27, issue 2, 2015, pp. 105-126 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-7

[11]. Ericson J. Hacking: The Art of Exploitation. 2nd Edition. // No Starch Press, 2008, pp.167-183.

[12]. Hong Hu, Zheng Leong Chua, Sendroiu Adrian, Prateek Saxena, Zhenkai Liang. Automatic Generation of Data-Oriented Exploits. // 24th USENIX Security Symposium 2015

[13]. 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). DOI: 10.15514/ISPRAS-2012-22-9

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

[15]. Dovgalyuk P.M., Makarov V.A., Romaneev M.S., Fursova N.I. Primenenie programmyih emulyatorov v zadachah analiza binarnogo koda.[Applying program emulators for binary code analysis] // Trudy ISP RAN [The Proceedings of ISP RAS], vol. 26, issue 1, 2014, pp. 277-296 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-9.

[16]. 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.

[17]. 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).

[18]. 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.

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

[20]. Nikolaj Bjørner, Leonardo de Moura. Z3: Applications, Enablers, Challenges and Directions // Sixth International Workshop on Constraints in Formal Verification Grenoble, 2009.

[21]. Huang S. K. et al. Software Crash Analysis for Automatic Exploit Generation on Binary Programs // Reliability, IEEE Transactions on. – 2014. – Т. 63. – №. 1. – С. 270-289.

[22]. Wu B. et al. Directed symbolic execution for binary vulnerability mining // Electronics, Computer and Applications, 2014 IEEE Workshop on. – IEEE, 2014. – pp. 614-617.

Обнаружение и оценка количества промахов когерентности на основе вероятностной модели

Е.А. Велесевич <evel@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. Ложное разделение кэша возникает, когда нити, параллельно выполняющиеся на разных ядрах, поочередно обновляют разные переменные, попадающие в одну строку кэша, что приводит к устареванию информации о текущем состоянии памяти в кэше ядра, используемом первой нитью приложения, и необходимости нити ждать, пока информация в кэше обновится. В статье для оценки количества промахов предлагается использовать инструментацию кода и постобработку ее результатов: по наблюденным промахам кэша в трассе обращений к памяти с временными метками для каждой нити вычисляется вероятность того, что во время обращения, выбранного с некоторой заранее определенной вероятностью, и следующего обращения к этой строке в этой же нити была запись в ту же строку в другой нити. Трассировщик программы реализован как проход в открытом компиляторе GCC, добавляющий специальные инструкции перед каждым обращением к памяти и выполняющийся после всех оптимизирующих проходов, благодаря чему становится возможной оценка эффективности использования кеша в оптимизированных приложениях. Анализатор реализован в виде отдельного приложения, которому подается на вход сгенерированные на тестовом наборе данных трассы обращений к памяти, имевших место в нитях анализируемого приложения. Замедление работы программы при трассировке для проверенных экспериментально тестовых приложений составляет примерно 10 раз, при этом оно зависит от вероятности выборки, которая в свою очередь выбирается в зависимости от характера и времени работы приложения, но практически не зависит от длины кэш-линии.

Ключевые слова: ложное разделение кэша; GCC; инструментация кода.

DOI: 10.15514/ISPRAS-2015-27(4)-3

Для цитирования: Велесевич Е.А. Обнаружение и оценка количества промахов когерентности на основе вероятностной модели. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 39-48. DOI: 10.15514/ISPRAS-2015-27(4)-3.

1. Введение

Кэширование часто используемых данных из оперативной памяти является одним из ключевых методов обеспечения высокой производительности современных процессоров. Кэши не является общими ресурсами: каждое процессорное ядро, как правило, имеет свой кэш первого уровня; в многопроцессорных системах каждый физический процессор имеет свою иерархию кэшей. Кэш хранит множество выровненных участков оперативной памяти, называемых строками. Один и тот же участок оперативной памяти может одновременно находиться в кэшах разных процессоров; когда один из процессоров модифицирует эти данные, необходимо обновить или удалить их из кэша другого процессора. Эта задача решается с использованием протокола согласования кэшей (cache coherency protocol).

Поскольку модификация даже одного байта приводит к обновлению целой строки кэша, могут возникать ситуации, когда нити, параллельно выполняющиеся на разных ядрах, поочередно обновляют разные переменные, попадающие в одну строку кэша. В таких случаях обновление на одном процессоре будет приводить к вытеснению соответствующей строки из кэша другого процессора и сериализовывать выполнение нитей, как если бы они обновляли одну и ту же переменную. Такое поведение известно как ложное разделение (false sharing) и ухудшает производительность параллельных программ. Соответственно, желательно иметь инструменты для анализа эффективности использования кэша, в том числе поиска ситуаций ложного разделения.

Далее в статье в разделе 2 описывается предлагаемый метод поиска ситуаций борьбы за кэш. В разделе 3 предлагается его реализация в инструменте на основе открытого компилятора GCC. В разделе 4 содержатся некоторые экспериментальные результаты. Раздел 5 завершает статью.

2. Метод поиска ошибок борьбы за кэш

Для оценки количества промахов доступа к кэшу можно использовать инструментацию кода. Основная идея заключается в том, чтобы собрать прореженную трассу доступов к памяти, включая расстояния повторного использования, и затем оценить количество промахов исходя из собранной трассы и некоторой теоретической модели кэша. Хагерстен [1] предлагает вероятностную модель для оценки эффективности использования кэша со случайным вытеснением на основе расстояния повторного использования строк кэша. Недостатком его подхода для поиска ситуаций ложного разделения в многопоточных программах является необходимость дополнительной синхронизации при сборе трассы. Нашей целью будет обойти эту необходимость, так как она может существенно замедлить выполнение программы.

Пусть L — количество строк в кэше; тогда вероятность того, что строка, находящаяся в кэше, уже не будет в нем после n промахов, равна:

$$f(n) = 1 - \left(1 - \frac{1}{L}\right)^n$$

Пусть $r(i)$ – вероятность промаха при i -ом обращении, а $A(i)$ – значение расстояния повторного использования для i -го обращения. Тогда количество промахов кэша между i -м и (i-$A(i)$-1)-м обращениями можно оценить как:

$$\sum_{j=i-A(i)}^{i-1} r(j)$$

Тогда вероятность того, что на момент обращения i искомая строка не будет находиться в кэше $f\left(\sum_{j=i-A(i)}^{i-1} r(j)\right)$. Но в тоже время, вероятность промаха исходно равна $r(i)$, т.е.:

$$r(i) = f\left(\sum_{j=i-A(i)}^{i-1} r(j)\right)$$

Предположив, что вероятность промаха постоянна, т.е. $r(i) = R$ для всех i (на некотором участке трассы программы), Хагерстен получает основное уравнение своей модели:

$$R * N = N_{cold} + \sum_{i=1}^{N_{nocold}} f(A(i) * R) \qquad (1)$$

В многопоточном случае модели Хагерстена запись в трассе помимо строки кэша и расстояния повторного использования содержит информацию о том, была ли запись в эту строку в другой нити приложения между первым и вторым обращением в первой нити. Такой подход к сбору информации о записях в другой нити требует синхронизации. Но возможен и другой подход — вычисление вероятности того, что во время обращения, записанного в трассу, и следующего обращения к этой строке в этой же нити, была запись в ту же строку в другой нити (на основе трасс обращений к памяти с временными метками для каждой нити приложения). Эта вероятность и будет искомым процентом промахов когерентности.

Для решения поставленной задачи требуется вычислить априорную вероятность того, что если промах когерентности произошел на обращении, записанном в трассе, то запись, благодаря которой произошел промах, была записана в трассу другой нити (таких записей может быть несколько). Пусть N_A – количество записей в кэш-строку A в исследуемом участке трассы второй нити, N – количество всех записей в участке, P – вероятность выборки

обращения для записи в трассу, T – продолжительность исследуемого участка, T_A – время между первым и вторым доступом к A в первой нити.

Тогда количество записей в участке есть $N_w = N/P$, среднее время между записями есть $T_w = T/N_W$, количество записей во второй нити между первым и вторым обращением в первой нити есть $N_{A_g} = T_A/T_w$.

А вероятность того, что между первым и вторым доступом будет k записей в ту же строку в другой нити, есть:

$$P_{A_k} = C_k^{N_{A_g}} P_A^K (1 - P_A)^{N_{A_g} - k}, \text{ где } P_A = N_A/N. \qquad (2)$$

Тогда вероятность, что в трассе будет обращение, благодаря которому произошел промах (хотя бы одно из них):

$$P_{A_d} = 1 - \sum_{k=1}^{N_{A_g}} P_{A_k} (1 - P)^k$$

А вероятность, что промах произошел, независимо от того, попал он в трассу или нет:

$$P_{A_m} = 1 - (1 - P_A)^{N_{A_g}}$$

Однако, нам известна статистическая вероятность попадания промаха в трассу:

$$\bar{P}_{A_d} = \frac{\bar{N}_{A_d}}{N},$$

где $\bar{N}_{A_d}$– количество замеченных промахов.

Тогда условная вероятность попадания промаха в трассу при наличии такого промаха:

$$\frac{\bar{P}_{A_d}}{P} = \frac{P_{A_d}}{P_{A_m}}$$

Из этого уравнения можно получить P – искомую вероятность промаха когерентности, или же процент обращений к памяти, повлекших промах когерентности.

Для достаточно точного определения $\bar{P}_{A_d}$ требуется намного более частая выборка обращений для записи в трассу, так как вероятность того, что между выбранными с вероятностью P обращениями в другой нити была выбрана запись, влекущая промах когерентности, тоже равна P, т.е. $\bar{P}_{A_d}$ порядка P^2. Поэтому P следует выбирать как квадратный корень из вероятности в модели Хагерстена.

3. Реализация инструмента поиска ситуаций борьбы за кэш

На основе вышеизложенной модели была реализована программа, оценивающая использования кэшей приложениями с несколькими исполняемыми потоками. Программа состоит из двух частей: трассировщик снимает трассы приложения для последующей обработки, анализатор вычисляет количество промахов кэша (промахи объема и промахи когерентности), а также информирует о месте в программе в которой произошел промах.

Трассировщик реализован в виде отдельного прохода в компиляторе GCC [2], вставляющего в необходимых местах вызовы функции (реализованной в отдельной библиотеке), осуществляющей выборку доступов в память для записи в трассу. Для сборки с трассировкой модернизированному компилятору нужно передать необходимые опции и при необходимости указать путь к библиотеке трассировки. Запуск трассируемого приложения происходит в обычном режиме. По окончании работы приложения в папке с исполняемым файлом появится несколько файлов трасс (по одной для каждой нити приложения), который необходимо будет передать анализатору.

Трассировку можно настраивать, задавая вероятность выборки доступа в память и длинны кэш-линии путем установки переменных окружения. Отметим необходимость задания длины кэш-линии уже в трассировщике, так как ему необходимо определять, в какую кэш-линию происходит доступ для решения о внесении его в трассу. В целом замедление работы программы составляет ~10 раз, но оно зависит от вероятности выборки (естественно, чем меньше доступов мы будем записывать, тем меньше замедление) и практически не зависит от длины кэш-линии.

Анализатор реализован в виде отдельного приложения, которому подаются на вход трасса (состоящая из нескольких подтрасс) анализируемого приложения. Анализатор работает в несколько стадий:

1. Вычисление распределения доступов к памяти по расстояния повторного использования и пометка промахов когерентности, попавших в трассу (и сообщение о них пользователю).
2. Численное решение модели Хагерстена для оценки промахов объема. Уравнение (1) решается методом простых итераций для поиска корня на интервале (0,1).
3. Вычисление априорной вероятности попадания промахов когерентности в трассу.
4. Вычисления на основе сравнения априорной и статистической вероятности попадания промахов в трассу.

В качестве параметра анализатору можно передать объем кэша.

4. Экспериментальные результаты

Для первичной проверки модели расчета промахов когерентности была использована тестовая трасса, состоящая их трех подтрасс (т.е. в приложении было три нити) с обращениями в память упорядоченными таким образом.

t1	t2	t3	t1	t2	t3	t1	t2	t3	t1

т.е. за обращением в первой нити, следовало обращений во второй нити, потом в третьей и снова в первой. Обращения происходили в массив размером с кэш, т.е. промахи объема почти отсутствовали. Естественно, трасса доступов к памяти была прорежена случайным образом, так что в итоге могло получаться примерно такое:

t1	t1	t1	t1	t2	t3	t2	t3	t1	t1

Можно вычислить априорную вероятность промахов когерентности следующим образом. Пусть b – вероятность обращения по определенному адресу (обратно пропорциональна размеру массива (кэша)), тогда вероятность, что при обращении по адресу A в нити 1 следующее обращение будет не в нити 1:

$$\big(1-(1-b)(1-b)\big)+(1-b)(1-b)(1-b)(\big(1-(1-b)(1-b)\big)+\cdots=$$

$$=(1-(1-b)^2)\frac{1}{1-(1-b)^3}=(2-b)/(b^2-3b+3)$$

$$\lim_{b\to 0}\frac{2-b}{b^2-3b+3}=\frac{2}{3}$$

Разработанный анализатор выдает правильные результаты для тестового примера (>66.(6)%, так как массив все же не бесконечный):

16мб кэш	нить 0	нить 1	нить 2
Ошибки когерентности	67.1%	66.8%	66.8%

Далее, было проведено тестирование инструмента на нескольких простых приложениях из пакета coreutils. Промахи кэша в программе sort (64 байт в

кэш-линии) представлены в таблице 1 (промахи объема) и таблице 2 (промахи когерентности).

Таблица 1. Промахи объема в программе sort.

Размер кэша	нить 0	нить 1	нить 2	нить 3	нить 4	нить 5	нить 6	нить 7
16мб	6.5%	2.7%	2.3%	2.4%	3.2%	3.0%	2.5%	3.0%
8мб	7.3%	3.1%	2.5%	2.6%	3.7%	3.3%	2.8%	3.4%
2мб	9.5%	4.7%	4.3%	4.5%	4.4%	4.4%	4.5%	4.8%

Таблица 2. Промахи когерентности в программе sort.

Размер кэша	нить 0	нить 1	нить 2	нить 3	нить 4	нить 5	нить 6	нить 7
16мб	0.5%	0.7%	0.7%	1.5%	0.2%	0.6%	1.0%	0.5%
8мб	0.5%	0.7%	0.7%	1.4%	0.3%	0.7%	1.0%	0.5%
2мб	0.5%	0.7%	0.7%	1.4%	0.3%	0.7%	1.0%	0.5%

Как можно видеть, промахи когерентности не слишком зависят от объема кэша. Чем меньше объем кэша, тем меньше вероятность, что другая нить испортит его содержимое. Но, как мы видим, объем кэша не очень сильно влияет на ошибки объема в приложении sort, поэтому мы и не видим уменьшения промахов когерентности в таблице.

Было получено несколько сообщений о ложном разделении, например:

```
    False sharing is detected:

    Previous access at ../../coreutils/coreutils-
8.16/src/sort.c:3080
```

```
    Conflicting access at ../../coreutils/coreutils-
8.16/src/sort.c:3375

    Recent access(FS) at ../../coreutils/coreutils-
8.16/src/sort.c:2673
```

В данном случае ложное разделение происходит при слиянии отсортированных участков входных данных, поэтому оно не представляет собой проблемы, несмотря на корректное сообщение инструмента, и избавляться от него не нужно.

5. Заключение

В статье описан метод поиска ситуаций борьбы за кэш и разработанный на его основе инструмент, состоящий из использующего компилятор GCC трассировщика и анализатора собранных трасс. Предложенная модель работы программы с памятью считает постоянной вероятность обращения к памяти в пределах наблюдаемого участка трассы для данной нити, что является достаточно сильным предположением, и в связи с ним часть ситуаций ложного разделения может быть потеряна инструментом. С другой стороны, наиболее вероятные ситуации разделения (а, следовательно, и наиболее нуждающиеся в исправлении) имеют больше шансов попасть в пределы наблюдаемого участка и быть обнаруженными. Поэтому можно утверждать, что ситуации ложного разделения, недостаточно проявляющиеся для того, чтобы быть обнаруженными инструментом, и не нуждаются в исправлении. Требуются дополнительные исследования на больших приложениях для ответа на вопрос о том, нужно ли расширение предложенной модели до учета переменной вероятности обращения к памяти.

Список литературы

[1]. Erik Berg, Håkan Zeffer and Erik Hagersten, A Statistical Multiprocessor Cache Model, In Proceedings of the 2006 IEEE International Symposium on Performance Analysis of System and Software, Austin, Texas, USA, March 2006

[2]. E. Berg and E. Hagersten. StatCache: A probabilistic approach to efficient and accurate data locality analysis, Technical report 2003-058, Department of Information Technology,Uppsala University, November 2003.

[3]. E. Berg and E. Hagersten. StatCache: A probabilistic approach to efficient and accurate data locality analysis. In Proceedings of International Symposium on Performance Analysis of Systems And Software, March 2004

[4]. S. R. Goldschmidt and J. L. Hennessy. The accuracy of trace-driven simulations of multiprocessors. In SIGMETRICS ’93, pages 146–157. ACM Press, 1993.

[5]. J. Mellor-Crummey, R. Fowler, and D. Whalley. Tools for Application-Oriented Performance Tuning. In Proceedings of 15th ACM International Conference on Supercomputing, Italy, June 2001.

[6]. R.A. Uhlig and T.N. Mudge, Trace-driven memory simulation: A survey, ACM Computing Surveys, 29 (2), 1997, 128–170.

[7]. Stunkel, C. and Fuchs, W. TRAPEDS: producing traces for multicomputers via execution-driven simulation. In Proceedings of the 1989 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Berkeley, CA, ACM, 70-78, 1989.
[8]. F. Rawson. Mempower: A simple memory power analysis tool set. Technical report, IBM Austin Research Laboratory, 2004
[9]. X. Gao, B. Simon, and A. Snavely, ALITER: An Asynchronous Lightweight Instrumentation Tool for Event Recording, Workshop on Binary Instrumentation and Applications, St. Louis, Mo. Sept. 2005.
[10]. Erik Berg, Methods for Run Time Analysis of Data Locality, Licentiate Thesis 2003-015, Department of Information Technology, Uppsala University, December 2003.
[11]. GCC, the GNU Compiler Collection. https://gcc.gnu.org.

Evaluating a Number of Cache Coherency Misses Based on a Statistical Model

Evgeny Velesevich <evel@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., 109004, Moscow, Russia

Annotation. False cache sharing happens when different threads executing in parallel on distinct processor cores simultaneously update the variables that reside in the same cache line. This results in the invalidation of the current memory state data that is saved in the cache utilized by the first thread's core, and also in the necessity to stall for the second thread on the memory state data update.
We suggest in this paper to evaluate the number of cache misses using code instrumentation and post-mortem trace analysis: the probability of the false sharing cache miss (defined as a memory write issued by one thread between two consecutive memory accesses issued by another thread) is calculated based on the gathered event trace, where each observed event is a memory access with a timestamp. The tracer tool is implemented as a GCC compiler pass inserting necessary tracing instructions before each memory access. The pass is scheduled after all other optimization passes that allow to use the tracer for optimized code. The post-mortem analyzer is a separate application that gets the trace collection gathered on a sample application input data as its own input. Program slowdown in our approach is ~10 times, and it is dependent on a sampling probability but it does not depend on a cache line size.

Keywords: false cache sharing; GCC compiler; code instrumentation

DOI: 10.15514/ISPRAS-2015-27(4)-3

For citation: Velesevich Evgeny. Evaluating a Number of Cache Coherency Misses Based on a Statistical Model. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 39-48 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-3.

References.

[1]. Erik Berg, Håkan Zeffer and Erik Hagersten, A Statistical Multiprocessor Cache Model, In Proceedings of the 2006 IEEE International Symposium on Performance Analysis of System and Software, Austin, Texas, USA, March 2006

[2]. E. Berg and E. Hagersten. StatCache: A probabilistic approach to efficient and accurate data locality analysis, Technical report 2003-058, Department of Information Technology,Uppsala University, November 2003.

[3]. E. Berg and E. Hagersten. StatCache: A probabilistic approach to efficient and accurate data locality analysis. In Proceedings of International Symposium on Performance Analysis of Systems And Software, March 2004

[4]. S. R. Goldschmidt and J. L. Hennessy. The accuracy of trace-driven simulations of multiprocessors. In SIGMETRICS '93, pages 146–157. ACM Press, 1993.

[5]. J. Mellor-Crummey, R. Fowler, and D. Whalley. Tools for Application-Oriented Performance Tuning. In Proceedings of 15th ACM International Conference on Supercomputing, Italy, June 2001.

[6]. R.A. Uhlig and T.N. Mudge, Trace-driven memory simulation: A survey, ACM Computing Surveys, 29 (2), 1997, 128–170.

[7]. Stunkel, C. and Fuchs, W. TRAPEDS: producing traces for multicomputers via execution-driven simulation. In Proceedings of the 1989 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Berkeley, CA, ACM, 70-78, 1989.

[8]. F. Rawson. Mempower: A simple memory power analysis tool set. Technical report, IBM Austin Research Laboratory, 2004

[9]. X. Gao, B. Simon, and A. Snavely, ALITER: An Asynchronous Lightweight Instrumentation Tool for Event Recording, Workshop on Binary Instrumentation and Applications, St. Louis, Mo. Sept. 2005.

[10]. Erik Berg, Methods for Run Time Analysis of Data Locality, Licentiate Thesis 2003-015, Department of Information Technology, Uppsala University, December 2003.

[11]. GCC, the GNU Compiler Collection. https://gcc.gnu.org.

О дедуктивной верификации Си программ, работающих с разделяемыми данными[1]

[1]М.У. Мандрыкин <mandrykin@ispras.ru>
[1, 2, 3, 4]А.В. Хорошилов <khoroshilov@ispras.ru>
[1] Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2] Московский государственный университет имени М.В. Ломоносова 119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК
[3]Московский физико-технический институт (государственный университет), 141700, Московская область, г. Долгопрудный, Институтский пер., 9
[4]НИУ Высшая школа экономики, Россия, Москва, 101000, ул. Мясницкая, д. 20

Аннотация. В статье рассматривается задача дедуктивной верификации кода ядра ОС Linux, написанного на языке Си и выполняющегося в окружении с высокой степенью параллелизма. Существенной особенностью этого кода является наличие работы с разделяемыми данными, что не позволяет применять классические методы дедуктивной верификации. Для преодоления этих сложностей в работе представлены предложения по формированию подхода к спецификации и верификации кода, работающего с разделяемыми данными, основанные на доказательстве соответствия этого кода заданной спецификации некоторой дисциплины синхронизации. Подход иллюстрируется примерами упрощенной модели спецификации спин-блокировок и внешнего интерфейса механизма синхронизации RCU (Read-copy-update), широко используемого в ядре ОС Linux.

Ключевые слова: Верификация, параллелизм, владение, инварианты, семантика языка С.

DOI: 10.15514/ISPRAS-2015-27(4)-4

Для цитирования: Мандрыкин М.У., Хорошилов А.В. О дедуктивной верификации Си программ, работающих с разделяемыми данными. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 49-68. DOI: 10.15514/ISPRAS-2015-27(4)-4.

[1] Исследование проводилось при финансовой поддержке Министерства образования и науки Российской Федерации (уникальный идентификатор проекта – RFMEFI60414X0051).

1. Введение

Дедуктивная верификация – это область статической верификации, в которой изучаются различные подходы к представлению условий корректности программ в соответствии с некоторыми заданными спецификациями в виде множества математических утверждений, называемых *условиями верификации* (УВ), с целью последующего анализа полученных УВ с помощью частично или полностью автоматизированного логического вывода. Одной из основных характеристик большинства методов дедуктивной верификации является их *корректность*, которая означает возможность гарантировать соответствие программы заданной спецификации в некоторых известных предположениях, полагаясь на корректность реализации применяемых инструментов. Инструменты логического вывода, наиболее широко используемые в области дедуктивной верификации, включают интерактивные и автоматические доказатели теорем, а также решатели формул в теориях. Помимо различных решателей и доказателей теорем реализации методов дедуктивной верификации обычно снабжаются соответствующим автоматическим инструментарием для перевода исходной программы в конечный результирующий набор УВ.

В контексте задачи наиболее полной и корректной верификации фрагментов кода ядра Linux применение инструментов дедуктивной верификации имеет как свои преимущества, так и недостатки. С одной стороны, инструменты дедуктивной верификации потенциально могут позволить осуществлять корректную модульную верификацию широкого диапазона свойств непосредственно на существующем низкоуровневом коде ядра, практически не прибегая к его модификации, что особенно важно для кода ядра ОС Linux, который изначально не разрабатывался с целью последующей формальной верификации. Это становится возможным благодаря большой универсальности используемого общего подхода, который в свою очередь полагается на значительную выразительность используемых логических систем. С другой стороны, соответствующие используемым логическим системам инструменты логического вывода всегда имеют ограниченные возможности в силу алгоритмической неразрешимости получаемых задач выполнимости в общем случае и большой алгоритмической сложности используемых алгоритмов логического вывода (для алгоритмически разрешимых фрагментов). На практике инструменты дедуктивной верификации обычно идут по пути частичного решения этих проблем с помощью взаимодействия с пользователем, а также с помощью применения различных сложных техник редукции, декомпозиции и упрощения получаемых УВ. Корме этого, часто инструменты верификации часто предоставляют различные вспомогательные инструменты для облегчения управления множеством используемых инструментов логического вывода и результатов взаимодействия с пользователем (например, полученных полуавтоматических доказательств теорем), получаемых в ходе процесса верификации. Возможность применения редукции обычно является следствием модульности применяемого метода дедуктивной верификации; декомпозиция

достигается за счет разбиения, применяемого к результирующим УВ на нескольких уровнях, таких как, например, разбиение по путям выполнения в исходном коде, разделение памяти на непересекающимся области, раздельная проверка различных аспектов поведения фрагмента программы (безопасности (условий корректности для всех используемых операций), явно специфицированного поведения (функциональной корректности), контекстных условий для ограничения эффектов выполнения императивного кода и др.), разбиение получаемых логических формул в соответствии с их пропозициональной структурой; упрощение формул также обычно возможно благодаря некоторым дополнительным легко проверяемым предположениям, наиболее важные из которых – это системы типов и модели памяти, которые позволяют предварительно использовать высокоэффективные алгоритмы статического анализа для разрешения многих условий корректности и быстрого извлечения дополнительных полезных знаний о решаемой задаче, которые затем могут быть использованы инструментами логического вывода для ускорения решения задачи выполнимости.

В применении дедуктивной верификации к коду ядра Linux основные сложности, таким образом, состоят в достижении эффективности получаемого представления семантики исходного кода в виде логических формул, несмотря на нестрогость типизации, разнообразную прагматику и сложную модель памяти, характерные для языка C, с учетом используемых в коде ядра Linux языковых расширений и низкоуровневых операций при работе с памятью; а также, в еще большей степени, в сохранении модульности применяемого метода дедуктивной верификации в контексте высокой степени параллелизма, характерной для ядра Linux.

Целью проекта *Astraver* [1] является развитие набора инструментов для высокоавтоматизированной дедуктивной верификации модулей ядра Linux. Набор инструментов разрабатывается на основе двух платформ анализа исходного кода и дедуктивной верификации. Первая, внешняя платформа – *Frama-C* [2] – это пакет инструментов для анализа исходного кода ПО, написанного на языке C. Frama-c – расширяемая и модульная платформа с архитектурой на основе динамически подключаемых модулей, которая включает свой собственный подключаемый модуль дедуктивной верификации *WP* [3], а также позволяет создавать и использовать сторонние подключаемые модули. В частности, модуль *Jessie* [4], который изначально был частью платформы дедуктивной верификации *Why*, был отщеплен и в настоящее время разрабатывается в рамках проекта Astraver. Являясь полноценным практическим средством анализа исходного кода на языке C, платформа Frama-C обеспечивает хорошую совместимость с диалектом GNU C, включая его различные специфичные возможности и расширения, используемые в коде ядра Linux. Frama-C также предоставляет свою собственную адаптированную модификацию инфраструктуры CIL [5] для анализа и трансформации исходного кода, которая значительно облегчает реализацию серии преобразований и нормализации исходного кода,

применяемых модулем Jessie с целью предварительного упрощения сложной модели памяти и семантики исходной С-программы. Модуль *Jessie* реализует дедуктивную верификацию путем перевода исходной программы на языке С вместе с соответствующими функциональными спецификациями, написанными на специальном языке спецификаций *ACSL* [6], в набор модулей на языке программирования и спецификации *Why3ML* [7], являющемся входным языком платформы дедуктивной верификации *Why3* [8]. Таким образом, платформа *Why3* служит второй, внутренней платформой дедуктивной верификации, предоставляющей выразительный входной язык и реализующей управление УВ: их генерацию, преобразование, разрешение с использованием внешних инструментов проверки выполнимости и управление ручными доказательствами. *Why3* реализует расширяемый механизм поддержки внешних инструментов проверки выполнимости (решателей и доказателей теорем), с использованием которого в инструменте реализована встроенная поддержка нескольких решателей формул в теориях ([9], [10]), доказателей теорем на основе расширения резолютивного вывода ([11], [12]) и интерактивных доказателей теорем ([13], [14]). Эффективность кодирования УВ достигается модулями WP и Jessie по-разному. В то время как модуль WP обеспечивает гибкость и эффективность, реализуя генерацию УВ напрямую, реализуя высокоуровневые преобразования УВ (с помощью модуля *Qed* [3]) и несколько различных моделей памяти (“модель Хоара”, типизированную и побитовую модели), которые могут быть заданны отдельно для каждого УВ, модуль Jessie реализует одну оптимизированную гибридную модель памяти на основе регионов [15] и эффектов [16] с частичной поддержкой явной низкоуровневой переинтерпретации типов указателей [17], оставляя непосредственно генерацию и управление (в том числе различные преобразования) УВ платформе Why3. В ходе начальной стадии исследований в рамках проекта Astraver, модель памяти Jessie (со значительными модификациями [17]) показала достаточную выразительность для представления последовательной части семантики фрагментов кода ядра Linux (включая различные виды приведений типа указателей и побитовые операции). В то же время в к настоящему времени ни в платформе Frama-C, на в платформе Why3 не была реализована поддержка параллельной части семантики для какого-либо класса параллельных программ на языке С.

Как и большая часть современного системного программного обеспечения, модули ядра Linux работают параллельно. Более того, в коде ядра Linux широко используются неблокирующие механизмы синхронизации, такие как атомарные операции, барьеры и особенно RCU [18].

Наиболее существенной сложностью при верификации параллельных программ одидаемо является отсутствие непрерывного потока управления. В то время как общие методы дедуктивной верификации последовательных программ, такие как исчисление слабейших предусловий полагаются на заданные пользователем контракты, представленные в виде предусловий, постусловий и инвариантов, соответствующих точкам в потоке управления программы, па-

раллельная среда на первый взгляд делает применение аналогичных методов верификации невозможным, разрешая недетерминированные наложения эффектов параллельного выполнения различных участков других потоков управления на большинстве путей между точками рассматриваемого потока управления. Такая возможность прерывания потока управления нарушает свойства локальности и композируемости методов дедуктивной верификации и приводит к комбинаторному взрыву числа возможных путей выполнения. Даже введение в рассмотрение примитивов синхронизации и атомарных операций само по себе практически не помогает сократить возникающее огромное пространство поиска среди всевозможных чередований. К счастью, существуют техники, позволяющие избежать явного полного перебора в возникающем пространстве чередований с помощью переноса инвариантов, ранее сопоставленных точкам в потоке управления последовательной программы, на типы данных, с которыми работает параллельная программа. С учетом такого изменения методологии (при определенных дополнительных ограничениях на инварианты используемых типов данных) введение дисциплины синхронизации позволяет свести полный перебор всевозможных чередований к так называемому *крупнозернистому параллелизму*, в рамках которого вмешательство параллельных потоков выполнения может происходить только в относительно небольшом числе явно обозначенных точек в потоке управления программы, а заданные инварианты типов данных требуется проверять лишь *локально*, то есть на заранее известном конечном множестве отрезков путей выполнения. Это позволяет полностью восстановить локальность метода верификации ценой потери гарантии завершимости параллельной программы. Методология *владения с локально проверяемыми двухметочными инвариантами* (англ. *locally-checked two-state invariants*, сокр. *LCI*) [22] является одним из таких локальных методов верификации параллельных программ. Основные понятия, водимые в методологии владения, – *двухметочный инвариант*, *взаимная допустимость* инвариантов и *динамическое утверждение*. Вместе они составляют основу локального метода дедуктивной верификации параллельных программ с использованием инвариантов типов данных. Общая методология владения может быть приспособлена для рассуждений в классической логике первого порядка, поддерживаемой большинством современных автоматических средств логического вывода, и ранее была успешно применена в инструменте дедуктивной верификации VCC [20].

В то время как многие локальные методы верификации многопоточных программ допускают использование только блокирующих примитивов синхронизации [19], инструмент VCC реализует модель параллелизма, позволяющую выражать многие варианты использования как блокирующих, так и неблокирующих примитивов. Модель параллелизма VCC является расширением общей методологии *LCI* дополнительными операциями атомарных обновлений асинхронно-изменяемых (англ. volatile) данных. По многим характеристикам инструмент VCC является концептуально схожим с набором Frama-c – Jessie – Why3, в частности, он использует в качестве входного языка подмножество

языка C, расширенное собственным языком спецификаций, предполагает использование некоторой модели типизированных объектов поверх языка C, реализует типизированную модель памяти с явной переинтерпретацией [21], а также использует решатель формул Z3 [9] (поддерживаемый Why3) в качестве средства автоматического логического вывода.

В данной статье предлагается предварительное описание расширения языка спецификаций ACSL, поддерживаемого платформой анализа C-программ Frama-C, набором примитивов методологии LCI, аналогичных тем, которые включены в язык спецификаций для инструмента VCC, а также приводится несколько примеров использования предлагаемых расширений для формализации примитивов синхронизации, включая модель для проверки корректности использования базовых примитивов (внешнего интерфейса) механизма синхронизации RCU, используемого в коде модулей ядра Linux.

2. Методология владения и LCI

Методология владения – это объектно-ориентированная дисциплина синхронизации, предполагающая, что каждый поток управления может осуществлять последовательные (неатомарные) операции записи только в объекты, которыми этот поток *владеет* и может осуществлять любые операции чтения только из объектов, которыми он либо владеет, либо имеет возможность доказать, что во время чтения эти объекты не будут изменены (иначе говоря, останутся *закрытыми*). Объекты как вершины, связанные направленными дугами, соответствующими отношению владения, организуются в лес, в котором каждый объект всегда имеет строго одного владельца, отличного от самого объекта, кроме потоков управления, которые также рассматриваются как объекты, всегда владеющие самими собой. Объектам типизированы и каждому типу объектов приписывается некоторый набор *двухметочных инвариантов*, которые ограничивают возможные операции обновления объектов соответствующего типа и могут быть сформулированы с использованием значений из состояния объекта как до, так и после его обновления. Несколько идущих подряд обновлений одного объекта могут быть объединены в серию так, что каждая такая серия обновлений (операция последовательной записи) может быть представлена для системы в целом как одно атомарное обновление состояния объекта. Это соответствует синхронизации с использованием блокирующих примитивов и при определенных ограничениях на инварианты типов объектов, параллельная программа, использующая только такую блокирующую синхронизацию, может быть без потери общности представлена с использованием *крупнозернистого параллелизма*, который гарантированно корректно аппроксимирует реальный, мелкозернистый параллелизм в предположении корректного использования методологии владения [23]. Для поддержки серий обновлений, а также неатомарной инициализации вновь выделяемых в памяти объектов в условиях методологии с двухметочными инвариантами, все объекты расширяются специальным булевым теневым состоянием, которое отражает, нахо-

дится ли соответствующий объект (с точки зрения текущего потока) в состоянии обновления. В фиксированной точке потока управления программы обновляемые объекты называются *открытыми*, в то время как гарантированно неизменяемые объекты называются *закрытыми*. Корректность использования методологии LCI, таким образом, прямо включает требование того, что все потоки управления должны работать только либо с закрытыми объектами, либо с объектами, которыми они владеют (и, как следствие, могут работать с ними эксклюзивно, в последовательном режиме). Из методологии LCI также прямо следует, что открывать объект должен только поток, который им владеет. В то же время если возможно доказать, что объект остается закрытым, он не может изменен и чтение из него корректно. Двухметочные инварианты предполагаются выполненными только для переходов между состояниями, в которых соответствующие объекты являются закрытыми. Наиболее важным свойством, которое обеспечивается методологией владения, является свойство локальности, означающее, что проверки двухметочных инвариантов для каждой сери операций обновления каждого объекта в отдельности при определенных ограничениях оказывается достаточно для гарантии сохранения инвариантов во всей параллельной программе целиком. Ограничение, налагаемое на двухметочные инварианты для достижения этого свойства называется ограничением *взаимной допустимости*. Двухметочный инвариант (взаимно) *допустим*, если он *рефлексивен*, то есть для него выполнено следствие $I(s_1, s_2) \rightarrow I(s_2, s_2)$ для любой пары состояний s_1 и s_2соответствующего объекта, и *стабилен*, то есть сохраняется при любом переходе, сохраняющем инварианты всех измененных объектов. Взаимная допустимость двухметочных инвариантов – это нелокальное свойство, которое в общем случае может зависеть от любого инварианта какого-либо типа объектов, но это свойство монотонно в том смысле, что будучи выполненным для какого-либо набора инвариантов, оно не может быть нарушено только лишь путем добавления в программу новых типов объектов или инвариантов. Проверка взаимной допустимости всех заданных инвариантов, таким образом, позволяет локализовать проверки сохранения этих инвариантов, каждый раз принимая в рассмотрение только инварианты объектов, непосредственно затрагиваемых при данном обновлении состояния.

В инструменте VCC используется расширение методологии владения, в котором вводится понятие *асинхронно-изменяемых* полей, для которых операции чтения и записи выполняются *атомарно* и разрешены вне зависимости от владения объектом, но только при условии, что они не выполняются одновременно с операциями последовательного обновления состояния объекта, то есть объект при обновлении остается закрытым. Любое атомарное обновление асинхронно-изменяемых полей объекта должно сохранять все его инварианты. Расширение методологии владения асинхронно-изменяемыми полями и атомарными операциями не нарушает свойства локальности всей методологии, но позволяет выразить (или специфицировать) с её помощью более широкий класс механизмов синхронизации.

3. Методология владения и ACSL

Язык спецификации *ACSL* не ориентирован на поддержку методологии владения, поэтому, ожидаемо, существует ряд трудностей, неизбежных как при непосредственной интеграции поддержки методологии владения в этот язык, так и при применении методологии владения поверх этого языка. Наиболее явные и значительные проблемы при этом включают следующие:

- В отличие от VCC, ACSL не обеспечивает и не предполагает использования какой-либо объектно-ориентированной парадигмы поверх языка C и даже не имеет достаточно высокоуровневой семантики. Однако, модуль Jessie в текущей реализации фактически уже осуществляет трансляцию исходных специфицированных C/ACSL-программ в объектно-ориентированный промежуточный язык (также называемый *Jessie* [24]). Таким образом, одним из первых шагов в направлении поддержки методологии владения в языке ACSL является предоставление с помощью него доступа к некоторым возможностям, доступным в промежуточном языке Jessie (или аналогичном объектно-ориентированном языке), в частности, к дополнительным предопределенным теневым полям каждого объекта, требуемым для поддержки методологии владения (например, *\closed* и *\owns*).
- В VCC (и в методологии владения в целом) отсутствует прямой аналог понятия валидности указателя, принятого в языке ACSL, и доступного через конструкции *\valid* и *\valid_read*. Наиболее близкими аналогами понятия валидного указателя в методологии владения являются понятия указателя на открытый объект (которым владеет текущий поток) для неатомарных операций и указателя на закрытый объект для атомарных операций. Объединение этих случаев наиболее близко соответствует семантике предиката *\valid* (точнее говоря, это корректное нижнее приближение этого предиката). Понятие указателя на закрытый объект или на объект, которым владеет текущий поток, аналогично корректно приближает предикат *\valid_read*. В целях упрощения интеграции существующих верифицированных последовательных фрагментов кода в контекст рассматриваемой методологии для параллельных программ предлагается объединить указанные близко соответствующие понятия, выразив их друг через друга в конечных специфицированных программах на языке Why3ML. При этом понятия методологии владения предлагается рассматривать как более примитивные, а предикаты *\valid* и *\valid_read* – как составные, выразимые через понятия методологии владения. В большинстве случаев это должно позволить части ранее верифицированного последовательного кода в параллельном контексте непосредственно, без каких-либо изменений в соответствующих спецификациях. Типичный пример такого использования – вызов функции, требующей валидно-

сти какого-либо указателя в контексте, где объект, адресуемый этим указателем открыт и его владельцем является текущий процесс.

- Подход к спецификации контрактов функций, принятый в языке ACSL, в частности конструкции *assigns* и *allocates*/*frees*, не имеет прямого соответствия в методологии владения (и в языке спецификаций для инструмента VCC). Семантика конструкции *assigns* не соответствует понятию разрешения на запись объекта в языке спецификаций для инструмента VCC, а конструкции для спецификации контекстных ограничений на состояние памяти (аналоги *allocates*/*frees*) полностью отсутствуют в этом языке. Однако по аналогии с предложенным решением для понятия валидности, рассматриваемые конструкции могут быть относительно легко сопоставлены контекстным ограничениям на запись в открытые объекты (текущего процесса) и на изменение множества объектов, которыми владеет текущий процесс соответственно. Таким образом, если объект остается открытым и принадлежащим текущему процессу (фактически это соответствует последовательному доступу) на протяжении всего выполнения функции (в пре- и пост- состояниях, а также в любом состоянии, в котором управление находится в этой функции), такой объект должен быть специфицирован с помощью конструкции *assigns*. Когда же в результате выполнения функции объект изменяет своего владельца на текущий процесс или с текущего процесса на какой-либо другой объект (возможно, процесс), такой объект должен быть специфицирован с помощью конструкций *allocates*/*frees* соответственно. Такое соответствие может показаться неожиданным для пользователей, поэтому для конструкций *allocates*/*frees* могут быть введены более очевидные синтаксически синонимичные обозначения (например *acquires*/*releases*).
- В последней версии языка ACSL (в версии, соответствующей Frama-C Sodium-20150201 [6]) определяется два вида инвариантов типов данных – сильные и слабые инварианты. Двухметочные инварианты, лежащие в основе методологии владения, не являются ни теми, ни другими, поэтому их предлагается непосредственно интегрировать в язык ACSL. Далее в данной статье для обозначения двухметочных инвариантов типов используется ключевое слово 2state.
- Еще одним не обязательным, но весьма желательным в контексте методологии владения, расширением языка ACSL является возможность объявлять теневые структурные типы с заданными инвариантами из любой точки программного кода (а не только в глобальном контексте). Это расширение наиболее важно для упрощения использования динамических утверждений (см. секцию 3.2).

4. Методология владения на примерах

4.1 Спин-блокитровки

Рассмотрим теперь пример достаточно простого механизма синхронизации, формализованного с использованием методологии владения. Намеренно упрощенная спецификация для механизма спин-блокировок, представленная на рис. 1, иллюстрирует предлагаемый подход к интеграции методологии владения в язык ACSL. Представленный пример в основном повторяет аналогичный пример спецификации из документации VCC [25], перенесенный на язык ACSL, однако он также демонстрирует некоторые особенности, специфичные для языка ACSL. Наиболее заметная из них – использование специально вводимого предиката *\new* для различения указателей на открытые и закрытые вновь выделенные объекты. Используемый в ACSL предикат *\fresh*, следуя предлагаемой семантике для валидности указателей, должен быть выполнен в обоих этих случаях. Семантики операций закрытия вновь выделенных и ранее открытых объектов отличаются, так как требуют выполнения различных проверок инвариантов (например, инвариант counter != *\old*(counter) → counter = *\old*(counter) + 1 невозможно доказать для операции закрытия после инициализации объекта). ACSL также требует явной спецификации контекстных ограничений на множество объектов, которыми владеет текущий процесс. В примере также используются вводимые встроенные примитивы *\acquire* и *\release* для изменения значений предопределенных теневых полей *\owner* и *\owns*.

```
volatile_owns struct spinlock {
  volatile unsigned int slock;
  //@ ghost void *resource;
};

/*@ 2state type invariant
  @   same_resource(struct spinlock l) = \old(l.resource) ==
l.resource;
  @*/

/*@ 2state type invariant
  @   ownership(struct spinlock l) =
  @     !l.slock ==> \subset(resource, \owns(&l));
  @
  @*/

/*@ requires \owned(obj) && \closed(obj);
  @ requires \valid(l) && \new(l);
  @ requires \owns(l) == \empty;
  @ frees obj;
  @ ensures \closed(l);
  @ ensures l->resource == obj;
  @*/
void spin_lock_init(struct spinlock *l /*@ ghost void *obj */)
```

```
{
  l->slock = 0;
  /*@ ghost {
    @   l->resource = obj;
    @   \release(obj, l);
    @   \close(l);
    @ }
    @*/
}

/*@ requires \closed(l);
  @ allocates l->resource;
  @ ensures \owned(l->resource) && \closed(l->resource);
  @*/
void spin_lock(struct spinlock *l)
{
  int stop = 0;
  do {
    /*@ atomic (l) */ {
      stop = !cmpxchg(&l->slock, 1, 0);
      /*@ ghost
        @   if (stop)
        @     \acquire(l->resource, l);
        @*/
    }
  } while (!stop);
}

/*@ requires \closed(l);
  @ requires \owned(l->resource) && \closed(l->resource);
  @ frees l->resource;
  @*/
void spin_unlock(struct lock *l)
{
  /*@ atomic (l) */ {
    l->slock = 0;
    //@ ghost \release(l->resource, l);
  }
}
```

Рис. 1. Пример спецификации спин-блокировок.

4.2 Динамические утверждения

Так же как и пример упрощенной спецификации спин-блокировок из документации VCC, его рассмотренная ACSL-версия изначально делает не приемлемые в реальности предположения, закоючающиеся в требовании в предусловиях примитивов синхронизации выполненности предикатов *\closed*(l). Проблема такого предусловия заключается в том, что неясно, как оно может

быть непосредственно доказано для потока, не владеющего объектом *l*. Для решения этой проблемы в инструменте VCC вводится специальное понятие *динамического утверждения*, которое представляет собой объект определенного типа (с соответствующим набором инвариантов), специально предназначенный для того, чтобы им мог владеть некоторый поток управления и, выполняя над этим объектом операции открытия/закрытия, использовать инварианты его типа для доказательства закрытости (и, возможно, некоторых других свойств) других объектов. Для упрощения использования механизма динамических утверждений в VCC вводятся специальное предопределенное теневое поле *\claim_counter*, которое проверяется в операциях открытия объектов и может быть изменено только с помощью специальных конструкций выделения/освобождения динамических утверждений. Пример объявления типа структуры для динамического утверждения приведен на рис. 2. В этом определении используется специальный атрибут *claim*, который позволяет добавить к типу структуры неявное поле *claimed* и двухместочный инвариант, который не является взаимно допустимым без учета дополнительного предусловия вида *\claim_counter* = 0 для операций открытия объектов. Таким образом, поддержка динамических утверждений требует введения дополнительных предусловий для операции открытия объектов. Эти предусловия могут быть либо фиксированными, либо задаваемыми пользователем для каждого типа структур аналогично двухместочным инвариантам. Введение фиксированных предусловий и встроенной поддержки динамических утверждений может быть обосновано большой частотой их использования в спецификациях параллельных программ. В варианте методологии владения, реализованной в инструменте VCC, роль динамических утверждений не ограничивается использованием их для доказательства закрытости объектов, принадлежащих множеству *claimed*. За счет формулирования дополнительных двухместочных инвариантов динамические утверждения в инструменте VCC превращаются в аналоги проверочных утверждений, а также пред- и постусловий в условиях параллельной среды. Поэтому конструкция выделения экземпляров динамических утверждений в VCC позволяет непосредственно указывать произвольный инвариант (предикат над состоянием совокупности закрытых объектов) при создании динамического утверждения. Предполагаемый синтаксис соответствующей конструкции в языке ACSL – *\claim(тег_структуры, указ._на_объект_1, .., указ._на_объект_n, предикат)*. В предикате предполагается возможность добавления полей к типу структуры динамического утверждения с помощью конструкции *выражение \as имя_поля*. Так как создание динамических утверждений выполняется для предикатов над совокупностью закрытых объектов, поля *\claim_counter* являются асинхронно-изменяемыми у всех типов структур. Поддержка конструкции *\claim* требует добавления возможности неявного объявления типов структур динамических утверждений вместе с соответствующими инвариантами, указываемыми при выделении экземпляров этих утверждений. Операции выделения динамических утверждений могут осуществляться только внутри атомарных секций, что в принципе позволяет син-

хронизировать операции изменения теневых счетчиков *\claim_counter* и реальных счетчиков ссылок с помощью соответствующих инвариантов и осуществлять соответствующие операции внутри одних тех же атомарных секций.

```
/*@ ghost claim struct claim {
  @  set<void*> claimed;
  @ };
  @
  @ // 2state type invariant
  @ //  claim(struct claim c) =
  @ //    \forall void *o;
  @ //      \subset(o, c.claimed) ==>
  @ //      \closed(o);
  @*/
```

Рис. 2. Пример спецификации типа структуры динамического утверждения.

4.3 Механизм синхронизации RCU

Read-copy-update (сокращенно RCU [18]) – это механизм синхронизации с поддержкой неблокирующих операций чтения и частично неблокирующей операции записи. В терминах механизма синхронизации RCU запись называется обновлением и осуществляется в две отдельных фазы. Первая фаза, фаза удаления, сводится к защищенному барьером синхронизации памяти атомарному обновлению значения защищенной механизмом синхронизации RCU указательной переменной на значение NULL (или другое предопределенное значение невалидного указателя), либо на значение адреса полностью и корректно инициализированного объекта (и, таким образом, готового к выполнению над ним произвольных операций чтения). Вторая фаза, фаза освобождения, состоит в эксклюзивном (обычно защищаемом блокировками) освобождении памяти, занимаемой ранее удаленным по завершению соответствующей предшествующей первой фазы устаревшим объектом. Две стадии обновления разделяются с помощью специальной блокирующей операции синхронизации, которая гарантирует, что соответствующий устаревший объект (уже удаленный, но еще не освобожденный) останется валидным и доступным для чтения до тех пор, пока все операции чтения этого объекта во всех параллельно выполняющихся потоках не будут гарантированно завершены. Для обеспечения этой гарантии операции чтения из всех защищаемых механизмом RCU указательных переменных должны быть обрамлены парой специальных ограничительных операций начала/конца критической сессии. Кроме этого, когерентность (свежесть) значений защищаемых переменных, из которых происходит чтение внутри критической секции, должна быть явно обеспечена хотя бы однажды внутри любой критической секции, причем строго до первого доступа к этим переменным внутри секции. Выполнение этого условия обеспечивается с помощью специальной операции защищенного разыменования. При

выполнении описанных условий блокирующая операция, осуществляемая в ходе обновления, может просто ожидать окончания всех критических секций, оставшихся открытыми по завершении фазы удаления, что автоматически гарантирует полное завершение всех операций чтения устаревшего объекта к моменту завершения блокирующей операции ожидания. Таким образом, сущность реализации механизма синхронизации RCU заключается в предоставлении следующих пяти основных примитивов: 1) защищенная барьером операция атомарного обновления значения для первой фазы обновления (в ядре Linux соответствующий примитив называется *rcu_assign_pointer*); 2) блокирующая операция, разделяющая две фазы обновления (в Linux она называется *synchronize_rcu*); 3) операция входа в критическую секцию чтения (*rcu_read_lock*); 4) операция выхода из критической секции чтения (*rcu_read_unlock*); 5) операция защищенного разыменования (*rcu_dereference*). Реализация механизма RCU в ядре Linux позволяет для большинства процессорных архитектур сделать реализацию операций 3–5 пустой, переложив таким образом все накладные расходы на синхронизацию на операции 1 и 2. Кроме этого, фаза освобождения вместе с предшествующей ей блокирующей операцией ожидания может быть выполнена в параллельном процессе. Это обеспечивает высокую эффективность и частоту использования механизма RCU в ядре Linux [26].

На рис. 3 показан пример упрощенной спецификации примитивов синхронизации механизма RCU, предназначенной для проверки корректности использования этого механизма в модулях ядра Linux. Пример служит для предварительной иллюстрации применимости расширенной (атомарными операциями) методологии владения для верификации модулей ядра Linux, активно [26] использующих предоставляемый основной частью ядра интерфейс механизма синхронизации RCU. Внутренняя (для основной части ядра) реализация механизма RCU, вполне вероятно, не может быть верифицирована в рамках существующих расширений методологии владения, так как в ней существенную роль играет слабая консистентность, не моделируемая в рамках методологии владения. Наибольшую сложность при спецификации внешнего интерфейса механизма RCU представляет формализация понятия защищенной указательной переменной, доступной для разыменования только внутри соответствующей критической секции, а также обеспечение эксклюзивности при выполнении операции освобождения.

```
/*@ axiomatic rcu {
  @   type rcu_section = integer;
  @
  @   2state type invariant
  @     rcu_secion(rcu_section s) = \owned(&s);
  @
  @   logic struct lock *rcu_lock(void **loc, void *obj);
  @   predicate rcu_reclaimable(void *obj);
  @ }
  @
  @ ghost rcu_section current_section = 0;
```

```
  @ ghost set<void *> rcu_dereferenced = \empty;
  @*/

/*@ requires current_section == 0;
  @ assigns current_section;
  @ ensures current_section != 0;
  @*/
void rcu_read_lock(void);

/*@ requires current_section != 0;
  @ assigns current_section;
  @ assigns rcu_dereferenced;
  @ frees rcu_dereferenced;
  @ ensures current_section == 0;
  @ ensures rcu_dereferenced == \empty;
  @*/
void rcu_read_unlock(void);

/*@ requires rcu_lock(loc, obj) == l;
  @ requires rcu_reclaimable(obj);
  @ allocates l;
  @ ensures \closed(l);
  @*/
void synchronize_rcu(void /*@ ghost void **loc, void *obj, struct lock
*l */);

/*@ requires \owned(l) && \closed(l);
  @ requires l->resource == obj;
  @ requires \claim_count(l) == 0;
  @ frees l;
  @ ensures rcu_lock(loc, obj) == l;
  @ ensures rcu_reclaimable( *loc);
  @*/
void rcu_protect(void **loc, void *obj, struct lock *l);

#define rcu_assign_pointer(p, v, l) \
  {\
    rcu_protect(&p, v, l);\
    p = v;\
    v;\
  }

/*@ requires current_section != 0;
  @ requires rcu_lock(loc, obj) != NULL;
  @ assigns rcu_dereferenced;
  @ allocates obj;
  @ ensures obj != NULL ==> \closed(obj);
  @ ensures rcu_dereferenced =
  @         \union(\old(rcu_dereferenced), obj);
  @*/
void *rcu_deref(void **loc, void *obj)

#define rcu_dereference(p, c) \
  ((typeof(p)) rcu_deref(&p, p));
```

Рис. 3. Упрощенная спецификация интерфейса механизма RCU.

5. Заключение

В статье предложено расширение языка спецификаций ACSL, поддержка которого реализована в наборе инструментов Frama-C–Jessie–Why3, с целью поддержки верификации параллельного кода с использованием методологии владения. В статье также рассмотрены примеры применения предложенного расширения для спецификации спин-блокировок и внешнего интерфейса механизма синхронизации RCU, широко используемого модулями ядра Linux. Предложенная формализация, однако, не была формально верифицирована.

Литература

[1]. http://linuxtesting.org/astraver.
[2]. P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, B. Yakobowski, “FRAMA-C, A Software Analysis Perspective”, Proceedings of International Conference on Software Engineering and Formal Methods 2012 (SEFM’12), October 2012.
[3]. L. Correnson, Z. Dargaye, A. Pacalet, “WP (Draft) Manual”",
[4]. http://frama-c.com/download/frama-c-wp-manual.pdf.
[5]. Y. Moy. Automatic Modular Static Safety Checking for C Programs: Ph.D. Thesis. Université Paris-Sud. – 2009. – January. http://www.lri.fr/~marche/moy09phd.pdf.
[6]. http://kerneis.github.io/cil/doc/html/cil/.
[7]. P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, V. Prevosto, “ACSL: ANSI/ISO C Specification Language. Version 1.7”, http://frama-c.com/download/acsl.pdf.
[8]. F. Bobot, J.-C. Filliâtre, C. Marché, A. Paskevich, “Why3: Shepherd your herd of provers”, Boogie 2011: First International Workshop on Intermediate Verification Languages, 2011.
[9]. J.-C. Filliâtre, A. Paskevich, “Why3 – where programs meet provers", In Programming Languages and Systems, pp. 125–128, Springer Berlin Heidelberg, 2013.
[10]. L. De Moura, N. Bjørner, “Z3: An efficient SMT solver”", In Tools and Algorithms for the Construction and Analysis of Systems, pp. 337—340, Springer Berlin Heidelberg, 2008.
[11]. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, C. Tinelli, “CVC4”, In Computer aided verification, pp. 171—177, Springer Berlin Heidelberg, January, 2011.
[12]. A. Riazanov, A. Voronkov, “The design and implementation of Vampire”, In AI communications, vol. 15(2, 3), pp. 91—110, 2002.
[13]. S. Schulz. “System Description: E 1.8”", In Proceedings of the 19th LPAR, Stellenbosch, pp. 477—483, LNCS 8312, Springer Verlag, 2013.
[14]. Y. Bertot, P. Castéran, “Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions”", Springer Science & Business Media, 2013.
[15]. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, M. Srivas, “PVS: Combining Specification, Proof Checking, and Model Checking”", In proceedings of Computer-Aided Verification '96, pp. 411—414, 1996.
[16]. T. Hubert, C. Marché, “Separation analysis for deductive verification”", In Heap Analysis and Verification, Braga, Portugal, March, 2007.
[17]. J.-P. Talpin, P. Jouvelot, “Polymorphic type region and effect inference”, Technical Report EMP-CRI E/150, 1991.

[18]. M. Mandrykin, A. Khoroshilov, “High level memory model with low level pointer cast support for Jessie intermediate language”, In Programming and Computer Software, Vol. 41, No. 4, pp. 197—208, 2015.
[19]. P. E. McKenney, J. Appavoo, A. Kleen, O. Krieger, R. Russell, D. Sarma, M. Soni, “Read-copy update”, In AUUG Conference Proceedings, p. 175, 2001.
[20]. C. Flanagan, S. N. Freund, S. Qadeer, “Thread-modular verification for shared-memory programs”, In ESOP 2002, Number 2305 in LNCS, Springer, pp. 262—277, 2002.
[21]. E. Cohen, M. Dahlweid, M. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, S. Tobies, “VCC: A practical system for verifying concurrent C”, In Theorem Proving in Higher Order Logics, pp. 23—42, Springer Berlin Heidelberg, 2009.
[22]. E. Cohen, M. Moskal, S. Tobies, W. Schulte, “A precise yet efficient memory model for C”, In Electronic Notes in Theoretical Computer Science, vol. 254, pp. 85—103. 2009.
[23]. E. Cohen, M. Moskal, W. Schulte, S. Tobies, “Local Verification of Global Invariants in Concurrent Programs”, In Computer Aided Verification, Springer Berlin Heidelberg, pp. 480—494, January, 2010.
[24]. E. Cohen, M. Moskal, W. Schulte, S. Tobies. “A practical verification methodology for concurrent programs”, Tech. Rep. MSR-TR-2009-15, Microsoft Research, 2009. (http://research.microsoft.com/pub)
[25]. J.-C. Filliâtre, C. Marché, “The Why/Krakatoa/Caduceus platform for deductive program verification”, In Proceedings of the 19th International Conference on Computer Aided Verification, Lecture Notes in Computer Science, Springer, 2007.
[26]. M. Moskal, W. Schulte, E. Cohen, M. A. Hillebrand, S. Tobies, “Verifying C programs: a VCC tutorial”', MSR Redmond, EMIC Aachen, 2012.
[27]. P.E. McKenney, S. Boyd-Wickizer, J. Walpole, “RCU usage in the Linux kernel: one decade later”, Technical report, 2013.

Towards Deductive Verification of C Programs with Shared Data

[1, 2, 3, 4]A.V. Khoroshilov <khoroshilov@ispras.ru>
[1]M.U. Mandrykin <mandrykin@ispras.ru>
[1] Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia
[2]Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
[3]Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4]Higher School of Economics, National Research University, 20 Myasnitskaya Ulitsa, Moscow 101000, Russia

Abstract. The paper takes a look at the problem of deductive verification of Linux kernel code that is concurrent and involves accesses to shared data and interactions with highly concurrent environment. The presence of shared data does not allow to apply traditional deductive verification techniques based solely on function contracts and loop invariants, so we

consider verification of such code using type invariants and a particular object-oriented methodology. For Linux kernel modules one of the usual goals of deductive verification is a formal proof of the code's compliance to a specification of a synchronization discipline. We propose formalizing both the synchronization discipline and the required properties of the code in terms of ownership methodology with locally checked invariants (LCI) that was previously successfully applied for verifying the Microsoft Hyper-V Hypervisor with VCC deductive verification tool. However to maintain good compatibility with the various specific C features and extensions used in the Linux kernel code, efficiently handle data type representations with many large nested structure definitions and provide a reacher specification language we propose using Frama-C static analysis platform with its ACSL specification language and Jessie plugin for deductive verification as these tools have shown a good applicability to verification of sequential Linux kernel code fragments in the course of the Astraver project. The paper presents preliminary discussion of issues arising from integration of support for the ownership methodology and LCI into both the ACSL specification language and its underlying toolset. In particular, the issue of reusing existing ACSL specifications in sequential context and the related issue of establishing a correspondence between ACSL notion of validity and LCI notions of owned, wrapped and closed object are discussed. The overall approach to specification of concurrent kernel code using ownership methodology, LCI and ACSL specification language is demonstrated on examples of spinlock specification and a simplified specification of RCU (Read-copy-update) API.

Keywords: Verification, concurrency, invariants, C programming language.

DOI: 10.15514/ISPRAS-2015-27(4)-4

For citation: Khoroshilov A.V., Mandrykin M.U. Towards Deductive Verification of C Programs with Shared Data. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 49-68 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-4.

References

[1]. http://linuxtesting.org/astraver.
[2]. P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, B. Yakobowski, “FRAMA-C, A Software Analysis Perspective”, Proceedings of International Conference on Software Engineering and Formal Methods 2012 (SEFM’12), October 2012.
[3]. L. Correnson, Z. Dargaye, A. Pacalet, “WP (Draft) Manual”",
[4]. http://frama-c.com/download/frama-c-wp-manual.pdf.
[5]. Y. Moy. Automatic Modular Static Safety Checking for C Programs: Ph.D. Thesis. Université Paris-Sud. – 2009. – January. http://www.lri.fr/~marche/moy09phd.pdf.
[6]. http://kerneis.github.io/cil/doc/html/cil/.
[7]. P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, V. Prevosto, “ACSL: ANSI/ISO C Specification Language. Version 1.7”, http://frama-c.com/download/acsl.pdf.
[8]. F. Bobot, J.-C. Filliâtre, C. Marché, A. Paskevich, “Why3: Shepherd your herd of provers”, Boogie 2011: First International Workshop on Intermediate Verification Languages, 2011.
[9]. J.-C. Filliâtre, A. Paskevich, “Why3 – where programs meet provers", In Programming Languages and Systems, pp. 125–128, Springer Berlin Heidelberg, 2013.

[10]. L. De Moura, N. Bjørner, “Z3: An efficient SMT solver”', In Tools and Algorithms for the Construction and Analysis of Systems, pp. 337—340, Springer Berlin Heidelberg, 2008.
[11]. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, C. Tinelli, “CVC4”, In Computer aided verification, pp. 171—177, Springer Berlin Heidelberg, January, 2011.
[12]. A. Riazanov, A. Voronkov, “The design and implementation of Vampire”, In AI communications, vol. 15(2, 3), pp. 91—110, 2002.
[13]. S. Schulz. “System Description: E 1.8”', In Proceedings of the 19th LPAR, Stellenbosch, pp. 477—483, LNCS 8312, Springer Verlag, 2013.
[14]. Y. Bertot, P. Castéran, “Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions”', Springer Science & Business Media, 2013.
[15]. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, M. Srivas, “PVS: Combining Specification, Proof Checking, and Model Checking”', In proceedings of Computer-Aided Verification '96, pp. 411—414, 1996.
[16]. T. Hubert, C. Marché, “Separation analysis for deductive verification”', In Heap Analysis and Verification, Braga, Portugal, March, 2007.
[17]. J.-P. Talpin, P. Jouvelot, “Polymorphic type region and effect inference”, Technical Report EMP-CRI E/150, 1991.
[18]. M. Mandrykin, A. Khoroshilov, “High level memory model with low level pointer cast support for Jessie intermediate language”, In Programming and Computer Software, Vol. 41, No. 4, pp. 197—208, 2015.
[19]. P. E. McKenney, J. Appavoo, A. Kleen, O. Krieger, R. Russell, D. Sarma, M. Soni, “Read-copy update”, In AUUG Conference Proceedings, p. 175, 2001.
[20]. C. Flanagan, S. N. Freund, S. Qadeer, “Thread-modular verification for shared-memory programs”, In ESOP 2002, Number 2305 in LNCS, Springer, pp. 262—277, 2002.
[21]. E. Cohen, M. Dahlweid, M. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, S. Tobies, “VCC: A practical system for verifying concurrent C”, In Theorem Proving in Higher Order Logics, pp. 23—42, Springer Berlin Heidelberg, 2009.
[22]. E. Cohen, M. Moskal, S. Tobies, W. Schulte, “A precise yet efficient memory model for C”, In Electronic Notes in Theoretical Computer Science, vol. 254, pp. 85—103. 2009.
[23]. E. Cohen, M. Moskal, W. Schulte, S. Tobies, “Local Verification of Global Invariants in Concurrent Programs”, In Computer Aided Verification, Springer Berlin Heidelberg, pp. 480—494, January, 2010.
[24]. E. Cohen, M. Moskal, W. Schulte, S. Tobies. “A practical verification methodology for concurrent programs”, Tech. Rep. MSR-TR-2009-15, Microsoft Research, 2009. (http://research.microsoft.com/pub)
[25]. J.-C. Filliâtre, C. Marché, “The Why/Krakatoa/Caduceus platform for deductive program verification”, In Proceedings of the 19th International Conference on Computer Aided Verification, Lecture Notes in Computer Science, Springer, 2007.
[26]. M. Moskal, W. Schulte, E. Cohen, M. A. Hillebrand, S. Tobies, “Verifying C programs: a VCC tutorial”', MSR Redmond, EMIC Aachen, 2012.
[27]. P.E. McKenney, S. Boyd-Wickizer, J. Walpole, “RCU usage in the Linux kernel: one decade later”, Technical report, 2013.

Библиотека ограничений для спецификации индустриальных моделей данных

[1, 2]С.В. Морозов <serg@ispras.ru>
[1]Д.В. Ильин <denis.ilyin@ispras.ru>
[1, 3]В.А. Семенов <sem@ispras.ru>
[1, 2]О.А. Тарлапан <oleg@ispras.ru>
[1] Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2] Московский государственный университет имени М.В. Ломоносова 119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК
[3]Московский физико-технический институт (государственный университет), 141700, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. В статье проводится анализ спецификаций индустриально значимого семейства объектно-ориентированных моделей данных на языке EXPRESS, на основе которого выделяются паттерны ограничений целостности, используемые в них. Разрабатывается библиотека обобщенных функций на языке EXPRESS для представления каждого из паттернов, которая может применяться как при рефакторинге существующих моделей, так и при разработке новых. Использование паттернов ограничений в спецификациях моделей позволяет улучшить их наглядность, облегчить их дальнейшее сопровождение и развитие и, в целом, ускорить их разработку. Кроме того, появляется возможность их анализа автоматизированными средствами программной инженерии. Обсуждается возможность применения данной библиотеки для решения задачи верификации моделей. Работа поддержана РФФИ (грант 13-07-00390).

Ключевые слова: объектно-ориентированное моделирование, EXPRESS, STEP, IFC, CIS/2, паттерны ограничений, верификация моделей

DOI: 10.15514/ISPRAS-2015-27(4)-5

Для цитирования: Морозов С.В., Ильин Д.В., Семенов В.А., Тарлапан О.А. Библиотека ограничений для спецификации индустриальных моделей данных. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 69-110. DOI: 10.15514/ISPRAS-2015-27(4)-5.

1. Введение

Модельно-ориентированный подход к разработке программного обеспечения (ПО) является одним из перспективных направлений программной инженерии. В его рамках предполагается активное использование моделей на всех этапах жизненного цикла сложных программных систем, включая их проектирование, разработку, сопровождение и развитие.

Важное место в данной области занимает методология MDA/MDD [1] и составляющие ее основу стандарты OMG: UML [2], MOF [3], XMI [4], CWM [5]. Их применение позволяет улучшить мобильность ПО за счет использования платформно-независимых моделей. В то же время, интерпретируемость моделей UML обеспечивает возможность их тестирования на ранних стадиях разработки, что позволяет повысить качество и надежность ПО.

При проектировании и производстве высокотехнологичной продукции в различных индустриальных областях с применением информационных технологий возникает проблема поддержки интероперабельности используемого ПО на различных этапах ее жизненного цикла, то есть способности одной программной системы открыто взаимодействовать с остальными без каких-либо ограничений. Подобная проблема решается путем разработки и стандартизации единой модели данных о продукции и единого интерфейса доступа к ним в рамках методологии STEP (STandard for Exchange of Product model data) [6], принятой в 1994 году в качестве международного стандарта по интероперабельности ПО. Таким образом, для поддержки интероперабельности ПО в различных индустриальных областях также используется модельно-ориентированный подход.

Согласно методологии STEP модель данных специфицируется на языке EXPRESS [7], а сами данные представляются в формате кодирования открытым текстом, известном также как STEP Physical File Format (SPFF) [8], в XML [9] или же в бинарном формате [10]. Форматы SPFF и STEP XML являются взаимозаменяемыми и преобразуются один в другой согласно установленным в стандарте правилам. STEP определяет также единый интерфейс доступа к данным [11]. Примерами таких моделей служат прикладные протоколы STEP для машиностроения [12], электроники [13], электротехники [14], автомобилестроения [15], судостроения [16–18], системотехники [19], производства мебели [20], а также стандартизованные независимо от STEP модели данных IFC для архитектуры и строительства [21], CIS/2 для строительства с использованием стальных конструкций [22], ISO 15926 для нефтегазодобывающей промышленности [23].

Прикладные протоколы STEP и независимые модели данных, разработанные на основе методологии STEP, представляют собой одну или несколько информационных схем на языке EXPRESS. Каждая схема включает описания объектных типов данных (ENTITY), каждый из которых характеризуется именем, уникальным в пределах схемы, набором атрибутов и ограничений

целостности. Данные представляют собой множество экземпляров объектных типов, специфицируемых в схемах EXPRESS. Ограничения целостности определяют условия соответствия этих данных семантике информационной модели. Часть ограничений, охватывающих несколько объектных типов, описываются в виде глобальных правил на уровне схемы. Остальные элементы, специфицируемые в схемах EXPRESS, носят вспомогательный характер. Так, определяемые пользователем типы данных, наряду с предопределенными типами языка EXPRESS, используются при описании атрибутов в объектных типах. Константы, процедуры и функции применяются в выражениях для вычисления значений производных атрибутов или проверки ограничений целостности данных. Детальное описание языка EXPRESS и его основных возможностей, а также классификацию его типов данных и ограничений можно найти в работах [24–26].

Обсуждаемые индустриальные модели данных являются масштабными и включают спецификации сотен или тысяч объектных типов данных и ограничений целостности. Они разрабатываются крупными консорциумами участников в течение продолжительного времени и требуют периодического пересмотра с целью их развития либо обнаружения и коррекции возможных ошибок. В связи с этим данные спецификации должны быть удобными как для чтения человеком, так и для их интерпретации и анализа автоматизированными средствами программной инженерии. С этой целью спецификации данных и ограничений целостности должны выражаться унифицированным образом.

Однако на практике даже семантически эквивалентные ограничения целостности, представляются, как правило, различными способами. Подобные неоднозначности вызываются богатством конструкций языка EXPRESS и могут проявляться даже в пределах одной модели, над которой трудились независимые группы разработчиков. Таким образом, систематизация ограничений, используемых в индустриальных моделях данных, и реализация их в виде единой библиотеки является актуальной задачей.

Целью проводимого исследования является анализ индустриально значимого семейства спецификаций, принятых в качестве международных или индустриальных стандартов на предмет возможности унификации представления в них ограничений целостности данных. В разделе 2 проводится аналитический обзор моделей данных, основанных на методологии STEP. В разделе 3 выделяются паттерны ограничений целостности данных, наиболее часто используемые в этих моделях. Для представления каждого из паттернов разрабатываются обобщенные функции на языке EXPRESS, организуемые в виде универсальной библиотеки. В разделе 4 приводятся рекомендации по использованию данной библиотеки. Детально обсуждаются возможности ее применения для решения задачи верификации моделей. В заключении подводятся итоги проведенного исследования.

2. Аналитический обзор моделей данных, основанных на методологии STEP

В данном разделе сначала рассматривается организация стандарта STEP, затем приводится пример архитектуры IFC как одной из независимых моделей, основанных на методологии STEP. Далее обсуждаются ограничения целостности данных, применяемые в моделях, специфицируемых на языке EXPRESS, и основные проблемы, связанные с их представлением и использованием.

2.1 Организация стандарта STEP

Стандарт STEP (ISO 10303), разрабатываемый и поддерживаемый техническим комитетом ISO TC 184, изначально предназначался для обеспечения интероперабельности САПР, применяемых в машиностроении. Однако впоследствии он распространился и на другие предметные области. Стандарт состоит из множества (нескольких сотен) томов, разрабатываемых и принимаемых по отдельности. Он имеет трехуровневую архитектуру, представленную на рис.1. Каждый из томов относится к одной из групп, представленных на данном рисунке.

<table>
<tr><td colspan="6">Application Protocols & Associated Abstract Test Suites</td></tr>
<tr><td colspan="6">Common Resources</td></tr>
<tr><td>Integrated generic resources</td><td>Integrated application resources</td><td>application-interpreted constructs</td><td>application modules</td><td>ISO 13584-20: logical model of expressions</td><td>ISO 15531-42: time model</td></tr>
<tr><td colspan="6">Methodology</td></tr>
<tr><td colspan="2">description methods</td><td colspan="2">implementation methods</td><td colspan="2">conformance testing methodology & framework</td></tr>
</table>

Рис. 1. Архитектура стандарта STEP (ISO 10303)

Методологическую часть стандарта STEP составляют три группы: методы описания, реализации и тестирования. К первой группе относятся том 1, описывающий структуру стандарта и определяющий основные понятия и термины, используемые в нем, том 11, содержащий основное руководство по языку моделирования данных EXPRESS, и том 14, определяющий язык EXPRESS-X [27] для преобразования данных, соответствующих одним схемам, специфицируемым на EXPRESS, в другие (аналогично языку QVT [28] для UML моделей или XSLT [29] для XML документов). Методы реализации (тома 21-29) описывают форматы данных STEP, стандартный интерфейс доступа к данным (SDAI) для организации STEP-совместимых баз данных и методы его связывания с языками программирования C, C++, Java, а также способы преобразования моделей на EXPRESS в UML/XMI. Методология тестирования (тома 31-35) описывает процедуры проверки соответствия прикладных систем стандарту, в частности определяет основные требования по тестированию, организацию данного процесса, абстрактные методы тестирования реализаций прикладных протоколов STEP и SDAI.

Общие ресурсы STEP — это спецификации наиболее общих элементов моделей данных, которые могут совместно использоваться несколькими прикладными протоколами. Фактически, они являются основными строительными блоками стандарта и существенно облегчают разработку новых моделей данных за счет повторного использования общих определений. К данной группе относятся интегрированные основные ресурсы (тома 41-61), интегрированные прикладные ресурсы (тома 101-112), прикладные компоненты (тома 501-523), прикладные модули (тома с нумерацией выше 1000), а также спецификации логической модели выражений и модели времени, заимствованные из других стандартов ISO.

Высший уровень иерархии архитектуры STEP составляют прикладные протоколы (тома серии 200) — модели данных, свойственные конкретной предметной области. Каждый прикладной протокол по требованиям стандарта должен состоять как минимум на 85% из объектных типов, наследуемых от общих ресурсов, и сопровождаться набором тестов для проверки соответствия ПО данному протоколу (тома серии 300). Детально архитектура STEP представлена в [24, 30].

2.2 Организация стандарта IFC

Industry Foundation Classes (IFC) — это стандартизованный формат данных с открытой спецификацией, предложенный, разработанный и поддерживаемый International Alliance for Interoperability (IAI) с целью обеспечения интероперабельности разнородного программного обеспечения в области архитектуры и строительства (Architecture, Engineering and Construction, AEC). Альянс был создан в 1995 году американскими и европейскими фирмами, работающими в AEC индустрии, вместе с фирмами-

производителями ПО для данной отрасли. Отделения IAI существуют в США, Великобритании, Франции, Германии, Финляндии, Испании, Японии, Сингапуре, Корее и Австралии. В 2005 году IAI был преобразован в buildingSMART International [31].

Рассмотрим организацию текущей версии стандарта IFC 4, которая принята также в качестве международного стандарта ISO 16739 [32]. Спецификация IFC включает определение схемы данных на языке EXPRESS, а также нормативно-справочной информации, в которую входят определения качественных (property) и количественных (quantity) характеристик объектов. Архитектура IFC представлена на рис. 2. Схема данных условно подразделяется на несколько подсхем, каждая из которых принадлежит одному из четырех концептуальных уровней:

1. Ресурсы (Resource) — включает множество подсхем, содержащих определения структур данных для общих ресурсов, совместно используемых на более высоких уровнях;
2. Ядро (Core) — определяет основную структуру иерархии, фундаментальные взаимоотношения и наиболее общие понятия объектной модели IFC как абстрактные типы данных, от которых прямо или косвенно наследуются объектные типы, объявленные на более высоких концептуальных уровнях;
3. Интероперабельность (Interoperability) — включает подсхемы, содержащие определения типов данных, которые используются в архитектурно-строительных процессах, продуктах и ресурсах на междисциплинарном уровне; объектные типы, объявленные в них, являются специализациями типов ядра IFC, но могут быть конкретизированы на уровне прикладных дисциплин либо использоваться в ассоциативных связях с объектными типами высшего концептуального уровня;
4. Сфера деятельности (Domain) — включает автономные подсхемы, содержащие определения типов данных, которые являются специфичными для конкретных прикладных дисциплин и применяются при обмене совместно используемой информацией между приложениями, относящимися к одной и той же области знания; объектные типы данных на этом уровне являются окончательными специализациями и не могут использоваться на прочих более низких уровнях.

Все объектные типы данных уровня ядра и выше прямо или косвенно наследуются от корневого объектного типа иерархии `IfcRoot`. В данном типе появляется определение глобального идентификатора (GUID) и, как следствие, понятие объектной идентичности (как совпадение значений GUID). В объектных типах общих ресурсов GUID не определен, и их экземпляры не могут существовать независимо от экземпляров высокоуровневых объектов. Таким образом, они обязаны прямо или косвенно

участвовать в ассоциациях с объектными типами, унаследованными от классов ядра IFC. Заметим, что часть схем ресурсов IFC использует спецификации основных ресурсов STEP для представления фундаментальной информации о продукте [33] (включая сведения о персонах и организациях, измерениях величин), геометрии и топологии [34], формы [35], материалов [36], визуальной информации [37]. В IFC эти спецификации были упрощены с учетом специфики прикладной области архитектуры и строительства.

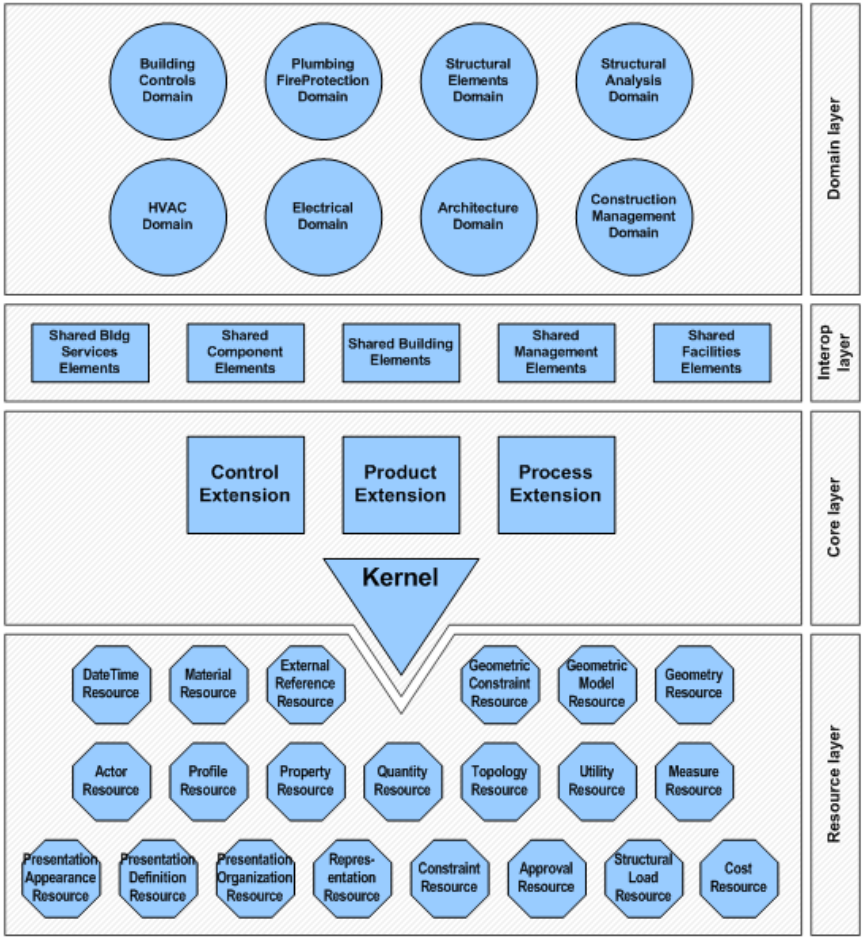

Рис. 2. Архитектура стандарта IFC 4

Заметим, что большинство механизмов, принятых в модели IFC, свойственно также известным онтологическим подходам [38]. Во-первых, это отдельные определения для объектных типов (наследуемых от абстрактного типа ядра

`IfcTypeObject`) и экземпляров или индивидов (наследуемых от абстрактного типа ядра `IfcObject`). Такой механизм позволяет легко расширить семантику модели путем создания новых экземпляров объектных типов и их ассоциации с соответствующими индивидами. Он широко применяется для уточнения типов различных управляющих элементов или же специализированного оборудования.

Во-вторых, в IFC используется объектификация взаимоотношений, которая позволяет сохранять их специфичные характеристики отдельно от типов взаимосвязанных объектов и расширить семантику языка EXPRESS путем определения иерархии типов взаимоотношений, наследуемых от абстрактного типа ядра `IfcRelationship`. Данная иерархия включает не только ассоциативные связи между объектами одного концептуального уровня, но и агрегации, композиции, группировки, определения и прочие зависимости, отсутствующие в языке EXPRESS.

В-третьих, каждому объекту IFC (`IfcObject`) и объектному типу (`IfcTypeObject`) могут быть назначены наборы качественных и количественных характеристик (`IfcPropertySetDefinition`), определяющие дополнительные статические свойства объектов и их типов наряду со стандартными атрибутами. Часть наборов характеристик для конкретных объектов и типов предопределена в стандарте. Любое из программных приложений, работающих с моделью IFC, может определять собственные динамически расширяемые наборы. В последнем случае стандарт предоставляет лишь метамодель для их описания: шаблон `IfcPropertyTemplateDefinition`, определяющий имена и описания характеристик, их типы данных, синтаксис для представления их значений, включая единицы измерения, и правила их назначения определенным объектам или типам. Приложение может предоставлять такой шаблон, но это является необязательным требованием стандарта.

2.3 Ограничения целостности данных, используемые в индустриальных моделях

Ограничения целостности данных, используемые в индустриальных моделях, специфицируемых на языке EXPRESS, можно условно подразделить на следующие группы:

- ограничения кардинальности (длин строковых и двоичных последовательностей, размеров коллекций, кратности прямых и инверсных связей);
- ограничения уникальности элементов в коллекциях или значений атрибутов объектов в наборе данных;
- требования наличия определенного значения для обязательных атрибутов объектов или элементов массивов;

- ограничения области определения значений, представляемые произвольными логическими выражениями или функциями.

Первые три группы ограничений представляются тривиальными синтаксическими конструкциями. В связи с этим, наибольший интерес для проводимого исследования составляет последняя группа ограничений, к которой относятся локальные правила WHERE, определяемые в простых и объектных типах данных и налагаемые на их индивидуальные экземпляры, а также глобальные правила RULE, определяемые на уровне схемы данных и налагаемые на экстенты одного или нескольких объектных типов.

Для спецификации локальных и глобальных правил могут использоваться разнообразные конструкции: определяемые пользователем функции, возвращающие результат логического типа, а также логические выражения. Несмотря на то, что язык EXPRESS является декларативным, в теле функций могут применяться операторы, свойственные императивным языкам, включая присвоения, условные переходы и циклы. Выражения языка EXPRESS также отличаются богатством синтаксических элементов (см. рис. 3). Используемые символы для записи операций в большинстве случаев соответствуют применяемым в других декларативных и императивных языках. Отдельно укажем только специфические для EXPRESS операции:

- `:=:` и `:<>:` — идентичность и неидентичность экземпляров объектного типа;
- `||` — конструирование экземпляра сложного объектного типа (конкатенирует конструкторы простых типов);
- `\` — групповая ссылка (возвращает ссылку на значение подтипа внутри экземпляра сложного объекта, включающую значения всех атрибутов, определенных в данном подтипе);
- `<=` и `>=` для типов коллекций — подмножество и надмножество (проверка, является ли первая коллекция, соответственно, подмножеством или надмножеством второй);
- `+`, `*`, `-` для типов коллекций — теоретико-множественные операции.

Таким образом, разработчик спецификаций на языке EXPRESS может использовать богатый репертуар сложных семантических конструкций для определения локальных и глобальных правил. Как результат, одно и то же ограничение целостности данных может быть выражено несколькими альтернативными способами. Это может быть продемонстрировано на конкретном примере.

```
expression := simple_expression [ rel_op simple_expression ]
```

```
rel_op := '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' |
':=:' | IN | LIKE
simple_expression := term { add_like_op term }
term := factor { multiplication_like_op factor }
factor := simple_factor [ '**' simple_factor ]
simple_factor := aggregate_initializer | entity_constructor |
                    enumeration_reference | interval |
query_expression |
                    ( [ unary_op ] ( '(' expression ')' |
primary ) )
unary_op := '+' | '-' | NOT
primary := literal | (qualifiable_factor { qualifier } )
multiplication_like_op := '*' | '/' | DIV | MOD | AND | '||'
add_like_op = '+' | '-' | OR | XOR
interval := '{' simple_expression interval_op
simple_expression interval_op
                    simple_expression '}'
interval_op := '<' | '<='
query_expression := QUERY '(' variable_id '<*'
aggregate_source '|'
                          logical_expression ')'
aggregate_source := simple_expression
aggregate_initializer := '[' [ element { ',' element } ] ']'
element := expression [ ':' numeric_expression ]
enumeration_reference := [ type_id '.' ] enumeration_id
entity_constructor := entity_id '(' [ expression { ','
expression } ] ')'
qualifiable_factor := attribute_id | built_in_constant |
constant_id |
                    function_call | parameter_id | variable_id
| entity_id
qualifier := attribute_qualifier | group_qualifier |
index_qualifier
```

```
attribute_qualifier := '.' atttibute_id
group_qualifier := '\' entity_id
index_qualifier := '[' numeric_expression [ ':'
numeric_expression ] ']'
function_call := ( built_in_function | function_id ) [
actual_parameter_list ]
actual_parameter_list := '(' expression { ',' expression } ')'
```

Рис. 3. Синтаксис выражений языка EXPRESS

В модели IFC наиболее часто используемым ограничением является уточнение типов объектов, участвующих в ассоциациях. Рассмотрим спецификации двух объектных типов данных, принадлежащих подсхеме `Geometric Model Resources`: `IfcBooleanClippingResult`, представляющий твердотельный объект, получаемый в результате отсечения булевскими операциями и `IfcPolygonalBoundedHalfSpace`, представляющий полупространство, ограниченное полигоном (см. рис. 4). В первом типе ограничиваются типы данных операндов для булевской операции. Так, первый операнд может быть твердым телом, получаемым либо путем перемещения произвольного плоского сечения в пространстве, либо путем перемещения окружности вдоль трехмерной директрисы, либо в результате другого отсечения (см. правило `FirstOperandType`). Во втором типе устанавливается, что граница полупространства может быть либо ломаной линией, либо композитной кривой (см. правило `BoundaryType`). При этом первое ограничение выражается через дизъюнкцию операций принадлежности имен отдельных допустимых типов множеству названий типов данных, членом которых является объект. Второе — через функцию проверки мощности множества, получаемого в результате пересечения множеств имен допустимых типов и названий типов данных, членом которых является объект.

```
ENTITY IfcBooleanClippingResult SUBTYPE OF (IfcBooleanResult);
 WHERE
       FirstOperandType : ('IFC4.IFCSWEPTAREASOLID' IN
TYPEOF(FirstOperand)) OR
                          ('IFC4.IFCSWEPTDISCSOLID' IN
TYPEOF(FirstOperand)) OR
                          ('IFC4.IFCBOOLEANCLIPPINGRESULT' IN
TYPEOF(FirstOperand));
```

```
       SecondOperandType : ('IFC4.IFCHALFSPACESOLID' IN
TYPEOF(SecondOperand));
       OperatorType : Operator = DIFFERENCE;
END_ENTITY;


ENTITY IfcPolygonalBoundedHalfSpace SUBTYPE OF
(IfcHalfSpaceSolid);
       Position : IfcAxis2Placement3D;
       PolygonalBoundary : IfcBoundedCurve;
 WHERE
       BoundaryDim : PolygonalBoundary.Dim = 2;
       BoundaryType : SIZEOF(TYPEOF(PolygonalBoundary) * [
                        'IFC4.IFCPOLYLINE',
                        'IFC4.IFCCOMPOSITECURVE']
                      ) = 1;
END_ENTITY;
```

Рис. 4. Примеры спецификации локальных правил в модели IFC

Подобные неоднозначности представления эквивалентных ограничений целостности данных затрудняют их интерпретацию как человеком, так и автоматизированными средствами. В связи с этим, является важной систематизация ограничений и унификация их представления.

3. Паттерны ограничений в индустриальных моделях данных

Проведенный анализ моделей данных IFC, CIS/2, а также общих ресурсов STEP показал, что, несмотря на большое количество ограничений, используемых в этих моделях (несколько сотен в каждой), все они соответствуют небольшому числу типовых случаев (паттернов). В перечисленных моделях возможно выделить 17 паттернов, каждый из которых соответствует определенному семантическому ограничению, но может быть выражено разными синтаксическими формами. Тем не менее, паттерны обычно легко распознаются по наличию тех или иных конструкций языка EXPRESS в логическом выражении ограничения.

Ниже детально обсуждается каждый из выделенных паттернов ограничений по следующей схеме. Выделяются типовые конструкции языка EXPRESS, которые соответствуют данному паттерну. Проводится математическая

формализация и по возможности предлагается обобщенная функция на языке EXPRESS для выражения ограничений данного типа.

3.1 Set Pattern

Ограничения данного вида устанавливают принадлежность значения переменной некоторому конечному множеству: $v \in S$. На языке EXPRESS они записываются с помощью оператора принадлежности IN, применяемого к атрибутам перечисляемого или строкового типа либо в виде дизъюнкции операций равенства атрибута допустимым литеральным значениям. Конечные множества строк наиболее часто используются в индустриальных моделях данных для задания названий предопределенных цветов, стилей оформления, семейств шрифтов, форматов представления данных и т.п. В обобщенном виде подобные ограничения можно представить следующей функцией:

```
FUNCTION BelongsToSet(v: GENERIC:t; s: AGGREGATE OF GENERIC:t)
: LOGICAL;
RETURN (v IN s);
END_FUNCTION;
```

Заметим, что к данному паттерну также можно отнести условия неравенства переменных перечисляемого типа некоторому литералу: $v \neq const$. Они легко преобразуются к виду $v \in S'$, где $S' \cup \{const\} = S, const \notin S'$. В качестве примера рассмотрим спецификацию объектного типа `IfcStructuralCurveReaction` для представления реакций (сил, смещений, отклонений и т.п.), распределенных вдоль кривой. Здесь правило `SuitablePredefinedType` запрещает синусоидальные и параболические типы распределений.

```
TYPE IfcStructuralCurveActivityTypeEnum = ENUMERATION OF
       (CONST, LINEAR, POLYGONAL, EQUIDISTANT, SINUS,
PARABOLA, DISCRETE,
             USERDEFINED, NOTDEFINED);
END_TYPE;


ENTITY IfcStructuralCurveReaction SUBTYPE OF
(IfcStructuralReaction);
       PredefinedType : IfcStructuralCurveActivityTypeEnum;
 WHERE
```

```
        HasObjectType : (PredefinedType <>
IfcStructuralCurveActivityTypeEnum.USERDEFINED) OR
EXISTS(SELF\IfcObject.ObjectType);
        SuitablePredefinedType : (PredefinedType <>
IfcStructuralCurveActivityTypeEnum.SINUS) AND (PredefinedType
<> IfcStructuralCurveActivityTypeEnum.PARABOLA);
END_ENTITY;
```

Данное правило может быть легко переформулировано с использованием предложенной обобщенной функции:

```
        SuitablePredefinedType :
BelongsToSet(SELF.PredefinedType,
                                    [
IfcStructuralCurveActivityTypeEnum.CONST,

        IfcStructuralCurveActivityTypeEnum.LINEAR,

        IfcStructuralCurveActivityTypeEnum.POLYGONAL,

        IfcStructuralCurveActivityTypeEnum.EQUIDISTANT,

        IfcStructuralCurveActivityTypeEnum.DISCRETE,

        IfcStructuralCurveActivityTypeEnum.USERDEFINED,

        IfcStructuralCurveActivityTypeEnum.NOTDEFINED ]);
```

3.2 Range Pattern

Данные условия ограничивают область определения значений числовых переменных (целых или вещественных) интервалом или полуинтервалом: замкнутым ($m \le v \le n$, или $v \le n$, или $v \ge m$), открытым ($m < v < n$, или $v < n$, или $v > m$) или полуоткрытым ($m < v \le n$ или $m \le v < n$). На языке EXPRESS они выражаются с помощью операторов сравнения или оператора интервала. В обобщенном виде их можно представить с помощью следующей функции, в которой значение переменной и границы интервала имеют общий

числовой тип NUMBER, а для определения замкнутости границ дополнительно вводятся два формальных параметра булевского типа:

```
FUNCTION LiesInRange(v, m, n : NUMBER; closedM, closedN :
BOOLEAN) : LOGICAL;
LOCAL
	resultLow : LOGICAL := TRUE;
	resultHigh : LOGICAL := TRUE;
END_LOCAL;
IF (EXISTS(m)) THEN
	IF (closedM = TRUE) THEN
		resultLow := m <= v;
	ELSE
		resultLow := m < v;
	END_IF;
END_IF;
IF (EXISTS(n)) THEN
	IF (closedN = TRUE) THEN
		resultHigh := v <= n;
	ELSE
		resultHigh := v < n;
	END_IF;
END_IF;
RETURN (resultLow AND resultHigh);
END_FUNCTION;
```

Неравенства вида $v \neq const$, где v — числовая переменная, представляются в виде дизъюнкции двух открытых полуинтервалов: $v < const \vee v > const$. Следовательно, их можно выразить в виде комбинации двух паттернов интервала, связанных логической функцией OR: `LiesInRange(v, ?, const, FALSE, FALSE) OR LiesInRange(v, const, ?, FALSE, FALSE)`, где символ '?' соответствует неопределенному значению.

Если ограничения интервала определены для коллекции чисел, то их можно переформулировать в виде следующей обобщенной функции:

```
FUNCTION AggregateLiesInRange(v: AGGREGATE OF NUMBER; m, n :
NUMBER; closedM, closedN : BOOLEAN): LOGICAL;
```

```
RETURN (SIZEOF(QUERY(temp <* v | LiesInRange(temp, m, n,
closedM, closedN))) = SIZEOF(v));
END_FUNCTION;
```

3.3 Constantation Pattern

Ограничения данного вида применяются с целью уточнения значения атрибута константой: $v = const$. Такое уточнение обычно используется в производных объектных типах, когда специализация допускает единственное конкретное значение для унаследованного атрибута. Также уточнение константой может происходить и для атрибутов ассоциируемых объектов при установлении соответствующих связей. Последнее наиболее часто используется при определении геометрических типов данных, когда требуется установить одинаковую размерность для геометрических объектов, связанных с некоторыми другими. Подобные ограничения можно представить в виде следующей обобщенной функции:

```
FUNCTION EqualsToConstant(v: GENERIC:t; const: GENERIC:t) :
LOGICAL;
RETURN (v = const);
END_FUNCTION;
```

Заметим, что подобные случаи близки по смыслу к уточнению атрибута в специализации объектного типа данных. Однако при генерации объектов уточненные атрибуты пропускаются и в результирующем наборе данных при его представлении в формате SFF на месте их значений ставится символ ‘*’. При наличии же ограничения константации значение соответствующего атрибута генерируется и представляется требуемым литералом в результирующем наборе.

3.4 Multiplicity Pattern

Ограничения данного вида уточняют размер атрибутов-коллекций, устанавливая либо их минимально допустимый ($|c| \geq size$), либо точный размер ($|c| = size$) в специализациях объектных типов данных или при установлении ассоциативных связей между объектами геометрических типов (аналогично ограничениям константации). Паттерн распознается по использованию функции SIZEOF для определения размера коллекции. В обобщенном виде это можно выразить с помощью следующей функции, где формальный параметр булевского типа `exact` устанавливает, задается ли точный или минимальный размер:

```
FUNCTION IsSizePermissible(c: AGGREGATE OF GENERIC; size:
INTEGER, exact: BOOLEAN): LOGICAL;
```

```
IF (exact = TRUE) THEN
       RETURN (SIZEOF(c) = size);
ELSE
       RETURN (SIZEOF(c) >= size);
END_IF;
END_FUNCTION;
```

Заметим, что иногда в спецификациях вместо функции SIZEOF при определении размеров коллекций используется функция HIINDEX, которая эквивалентна SIZEOF для списков, множеств и мультимножеств. В тех случаях, когда допустимый размер определяется строгим неравенством ($|c| > size$), его следует преобразовать в нестрогое ($|c| \geq size + 1$) для возможности выражения с помощью вышеприведенной обобщенной функции.

Для строковых данных, минимально допустимая или точная длина которых в EXPRESS определяется с помощью функции LENGTH, обобщенная функция может быть представлена в следующем виде:

```
FUNCTION IsStringSizePermissible(v: STRING; size: INTEGER,
exact: BOOLEAN): LOGICAL;
IF (exact = TRUE) THEN
       RETURN (LENGTH(v) = size);
ELSE
       RETURN (LENGTH(v) >= size);
END_IF;
END_FUNCTION;
```

К этому же паттерну следует отнести ограничения длин двоичных последовательностей, определяемые в EXPRESS с помощью функции BLENGTH и устанавливающие требования их кратности восьми или целому числу байтов:

```
FUNCTION IsIntegerNumberOfBytes(v: BINARY) : LOGICAL;
RETURN (BLENGTH(v) MOD 8 = 0);
END_FUNCTION;
```

3.5 Existence Pattern

Ограничения данного вида либо устанавливают обязательность существования значения опционального атрибута, наследуемого от базового объектного типа ($\exists v$), либо контролируют существование значений для

нескольких атрибутов одного объектного типа, определяемое некоторой логической функцией. В первом случае логическое выражение на языке EXPRESS состоит из единственной функции EXISTS и его можно представить в следующем обобщенном виде:

```
FUNCTION IsExist(v: GENERIC) : LOGICAL;
RETURN (EXISTS(v));
END_FUNCTION;
```

Во втором случае функции EXISTS для проверки существования значений индивидуальных атрибутов связываются допустимыми в языке логическими операциями: AND, OR, XOR, NOT. Подобные ограничения можно рассматривать как комбинацию паттернов существования.

3.6 Existence on Determinator Pattern

Ограничения данного вида устанавливают требование существования значения для опционального атрибута в зависимости от состояния другого атрибута этого объектного типа данных, который называется определителем и имеет перечисляемый тип: $d = const \rightarrow \exists v$. Иногда определитель может также являться опциональным. Тогда спецификация ограничения включает дополнительную проверку существования его значения: $\exists d \wedge d = const \rightarrow \exists v$.

Заметим, что данные правила можно представить в виде комбинации паттернов существования и константации. Однако это один из наиболее часто встречаемых видов ограничений в индустриальных моделях данных, в связи с чем было решено выделить их в отдельный паттерн. Подобные правила используются, например, в вышеописанной схеме расширения семантики IFC для определяемых пользователем типов объектов. Рассмотрим тип данных `IfcBeam` для представления строительных балок. В нем атрибут `PredefinedType` является определителем, а строковый атрибут `ObjectType`, унаследованный от `IfcObject` и задающий имя определяемого пользователем типа балки — зависимым. Имя типа обязано существовать, если `PredefinedType` установлен как `USERDEFINED` (см. правило `CorrectPredefinedType`).

```
TYPE IfcBeamTypeEnum = ENUMERATION OF
       (BEAM, JOIST, HOLLOWCORE, LINTEL, SPANDREL, T_BEAM,
USERDEFINED,
              NOTDEFINED);
END_TYPE;
```

```
ENTITY IfcBeam SUPERTYPE OF (ONEOF (IfcBeamStandardCase))
SUBTYPE OF (IfcBuildingElement);
       PredefinedType : OPTIONAL IfcBeamTypeEnum;
 WHERE
       CorrectPredefinedType : NOT(EXISTS(PredefinedType)) OR
              (PredefinedType <> IfcBeamTypeEnum.USERDEFINED)
OR
              ((PredefinedType = IfcBeamTypeEnum.USERDEFINED)
AND EXISTS (SELF\IfcObject.ObjectType));
       CorrectTypeAssigned : (SIZEOF(IsTypedBy) = 0) OR
         ('IFC4.IFCBEAMTYPE' IN
TYPEOF(SELF\IfcObject.IsTypedBy[1].RelatingType));
END_ENTITY;
```

Подобные ограничения можно выразить следующей обобщенной функцией, где формальный параметр булевского типа `isDExists` устанавливает необходимость проверки существования значения определителя:

```
FUNCTION ExistsOnDeterminator(v: GENERIC:obj; d: GENERIC:t;
const: GENERIC:t; isDExist: BOOLEAN) : LOGICAL;
IF (isDExist = TRUE AND NOT(EXISTS(d))) THEN
       RETURN (TRUE);
END_IF;
IF (d <> const) THEN
       RETURN (TRUE);
END_IF;
RETURN (d = const AND EXISTS(v));
END_FUNCTION;
```

3.7 Type Refinement Pattern

Правила данного вида уточняют типы объектов, участвующих в ассоциациях. Если ассоциация определена на некотором обобщенном объектом типе, то с помощью данных правил можно ограничить количество специализаций, имеющих право участвовать в ней. Подобное уточнение обычно используется, если ассоциация унаследована от базового объектного типа. Формализовать данный тип ограничений можно следующим образом. Пусть функция $typeof(o)$ возвращает множество названий типов данных, членом

которых является объект o. То есть, это множество включает название собственного объектного типа и всех его обобщений вплоть до корня иерархии. Пусть множество T_A включает названия типов данных, допустимых для участия в ассоциации, а множество T_D — названия недопустимых типов. Тогда ограничения формулируются следующим образом: $|typeof(o) \cap T_A| = 1 \wedge |typeof(o) \cap T_D| = 0$. На языке EXPRESS это можно представить следующей обобщенной функцией, где дополнительный параметр `isAllowed` устанавливает, задается ли параметром t множество допустимых или недопустимых типов:

```
FUNCTION RefinesType(o: GENERIC; t: SET OF STRING; isAllowed:
BOOLEAN) : LOGICAL;
LOCAL
       allowedNumber : INTEGER;
END_LOCAL;
IF (isAllowed = TRUE) THEN
       allowedNumber := 1;
ELSE
       allowedNumber := 0;
END_IF;
RETURN (SIZEOF(TYPEOF(o) * t) = allowedNumber);
END_FUNCTION;
```

Для уточнения типов во множественных ассоциациях может использоваться следующая обобщенная функция:

```
FUNCTION RefinesTypeInCollection(c: AGGREGATE OF GENERIC; t:
SET OF STRING; isAllowed: BOOLEAN) : LOGICAL;
RETURN (SIZEOF(QUERY(temp <* c | RefinesType(temp, t,
isAllowed)) = SIZEOF(c));
END_FUNCTION;
```

3.8 Type Refinement on Determinator Pattern

Ограничения данного вида уточняют типы объектов, участвующих в ассоциациях, в зависимости от состояния атрибута-определителя. Их можно представить в виде комбинации паттернов константации и уточнения типа: $d = const \rightarrow |typeof(o) \cap T_A| = 1$. В связи с тем, что они также часто встречаются в индустриальных моделях данных, было решено выделить их в отдельный паттерн.

Как правило, подобные ограничения выражаются в виде логической функции, тело которой состоит из единственного оператора альтернативы CASE или же последовательности условных операторов, которые ставят в соответствие множества допустимых типов различным значениям атрибута-определителя. В анализируемых моделях данных ограничения подобного типа, как правило, применяются к коллекциям объектов. Тогда обобщенная функция на языке EXPRESS может быть представлена следующим образом:

```
FUNCTION RefinesTypeOnDeterminator(c: AGGREGATE OF
GENERIC:obj; d: GENERIC:dt; const: LIST OF GENERIC:dt; t: LIST
OF SET OF STRING) : LOGICAL;
REPEAT i := LOINDEX(const) TO HIINDEX(const);
       IF (d = const[i]) THEN
              IF (SIZEOF(t[i]) = 0) THEN
                     RETURN (TRUE);
              ELSE
                     RefinesTypeInCollection(c, t[i], TRUE);
              END_IF;
       END_IF;
END_REPEAT;
RETURN (?);
END_FUNCTION;
```

3.9 Subset Pattern

Ограничения данного вида устанавливают, является ли одна коллекция подмножеством другой: $v_1 \subseteq v_2$. На языке EXPRESS они формулируются либо с помощью операции <= над коллекциями, либо путем выполнения операции QUERY к элементам первой коллекции с условием проверки принадлежности каждого из ее элементов второй коллекции с помощью операции IN. Это можно выразить с помощью следующей обобщенной функции:

```
FUNCTION IsSubset(v1: AGGREGATE OF GENERIC:t; v2: AGGREGATE OF
GENERIC:t) : LOGICAL;
RETURN SIZEOF(QUERY(temp <* v1 | temp IN v2)) = SIZEOF(v1)));
END_FUNCTION;
```

3.10 Correlated Size Pattern

Ограничения данного вида устанавливают равенство размеров двух атрибутов-коллекций в объектном типе данных: $|c_1| = |c_2|$. На языке EXPRESS для определения размеров коллекций используются функции SIZEOF или HIINDEX, и в обобщенном виде данный паттерн можно представить с помощью следующей функции:

```
FUNCTION AreSizesEqual(c1: AGGREGATE OF GENERIC; c2: AGGREGATE
OF GENERIC): LOGICAL;
RETURN (SIZEOF(c1) = SIZEOF(c2));
END_FUNCTION;
```

К этому же паттерну следует отнести правила, устанавливающие равенство длин строк или двоичных последовательностей. Поскольку для определения этих длин в EXPRESS используются функции LENGTH и BLENGTH соответственно, для их представления в обобщенном виде вводятся индивидуальные функции:

```
FUNCTION AreStringSizesEqual(v1: STRING; v2: STRING) :
LOGICAL;
RETURN (LENGTH(v1) = LENGTH(v2));
END_FUNCTION;

FUNCTION AreBinarySizesEqual(v1: BINARY; v2: BINARY) :
LOGICAL;
RETURN (BLENGTH(v1) = BLENGTH(v2));
END_FUNCTION;
```

3.11 Correlated Type Pattern

Подобные ограничения определяются для селективных типов данных. Согласно этим правилам атрибуты должны быть установлены либо в одинаковые, либо в разные допустимые типы данных по списку выбора соответствующих селективных типов. Это можно формализовать следующим образом: $typeof(v_1) = typeof(v_2)$ или $typeof(v_1) \neq typeof(v_2)$, где функция $typeof(v)$ возвращает текущий тип данных для переменной селективного типа. На языке EXPRESS подобные ограничения можно представить с помощью следующей обобщенной функции, где дополнительный параметр `isEqual` устанавливает, должны ли совпадать типы данных:

```
FUNCTION AreTypesCorrelate(v1: GENERIC:t; v2: GENERIC:t;
isEqual:BOOLEAN) : LOGICAL;
IF (isEqual = TRUE) THEN
       RETURN (TYPEOF(v1) = TYPEOF(v2));
ELSE
       RETURN (TYPEOF(v1) <> TYPEOF(v2));
END_IF;
END_FUNCTION;
```

Аналогичные ограничения могут быть также определены для коллекций селективных элементов. Тогда обобщенная функция имеет следующий вид:

```
FUNCTION AreTypesCorrelateInCollection(c: AGGREGATE OF
GENERIC; isEqual:BOOLEAN) : LOGICAL;
IF (isEqual = TRUE) THEN
       RETURN (SIZEOF(QUERY(temp <* c | TYPEOF(temp) <>
TYPEOF(c[1]))) = 0);
ELSE
       RETURN (SIZEOF(QUERY(temp <* c | TYPEOF(temp) <>
TYPEOF(c[1]))) = SIZEOF(c));
END_IF;
END_FUNCTION;
```

3.12 Alternative Reference Pattern

Данные ограничения запрещают связь с одним и тем же экземпляром из двух однотипных ассоциаций в некотором объектном типе или же рекурсивную связь (ассоциацию объекта с самим собой). Они распознаются по наличию операций идентичности или неидентичности в логическом выражении. Требования единственности роли ассоциируемых объектов используются в наследуемых от `IfcRelationship` типах данных. Рекурсивные связи запрещаются в типах данных сложных качественных и количественных характеристик, которые конструируются как множества других характеристик. Если обе ассоциации являются одиночными, то ограничение представляется следующей обобщенной функцией на языке EXPRESS:

```
FUNCTION AreDifferentInstances(r1: GENERIC:t; r2: GENERIC:t) :
LOGICAL;
RETURN (r1 :<>: r2);
```

```
END_FUNCTION;
```

Для проверки отсутствия экземпляра во множественной ассоциации можно использовать следующую обобщенную функцию:

```
FUNCTION IsInstanceMissingInMultipleAssociation(r: AGGREGATE
OF GENERIC:t; o: GENERIC:t) : LOGICAL;
RETURN (SIZEOF(QUERY(temp <* r | o :=: temp)) = 0);
END_FUNCTION;
```

3.13 Unique Name Pattern

Данные ограничения устанавливают требования уникальности имен объектов, участвующих во множественной ассоциации с некоторым другим объектом: $\forall o_i \in O, o_j \in O \rightarrow o_i.name \neq o_j.name, i \neq j$, где O — множество ассоциируемых объектов. В IFC 4 такие требования распространяются, например, на имена качественных и количественных характеристик в пределах одного набора или сложной характеристики. На языке EXPRESS подобные правила можно представить в виде следующей обобщенной функции:

```
FUNCTION AreNamesUnique(O: AGGREGATE OF GENERIC) : LOGICAL;
LOCAL
       Names : SET OF STRING := [];
END_LOCAL;
REPEAT i := LOINDEX(O) TO HIINDEX(O);
       Names := Names + O[i].Name;
END_REPEAT;
RETURN (SIZEOF(Names) = SIZEOF(O));
END_FUNCTION;
```

3.14 Equivalence Pattern

Ограничения данного вида устанавливают равенство значений двух атрибутов в одном объектном типе или же атрибутов в объектах, связанных ассоциацией друг с другом или третьим объектом: $v_1 = v_2$. Обобщенная функция на языке EXPRESS для выражения подобных правил выглядит следующим образом:

```
FUNCTION AreAttributesEqual(v1: GENERIC:t; v2: GENERIC:t) :
LOGICAL;
RETURN (v1 = v2);
```

```
END_FUNCTION;
```

Еще одной разновидностью ограничений, которые можно отнести к данному паттерну, являются требования равенства какого-либо атрибута в объектах, участвующих во множественной ассоциации. Например, в IFC 4 эти правила устанавливают равенство размерностей ассоциируемых геометрических объектов или совпадение типов двухмерных сечений, что можно представить следующими обобщенными функциями:

```
FUNCTION AreDimensionsEqual(c: AGGREGATE OF GENERIC) :
LOGICAL;
RETURN (SIZEOF(QUERY(temp <* c | temp.Dim <> c[1].Dim)) = 0);
END_FUNCTION;


FUNCTION AreProfileTypesEqual(c: AGGREGATE OF GENERIC) :
LOGICAL;
RETURN (SIZEOF(QUERY(temp <* c | temp.ProfileType <>
c[1].ProfileType)) = 0);
END_FUNCTION;
```

3.15 Sequence Patterns

Ограничения данного вида устанавливают принадлежность значения атрибута некоторой аналитически определяемой последовательности. Можно выделить несколько типовых случаев в зависимости от специфики заданной последовательности.

3.15.1 Sequence Pattern: Loop

В данном случае устанавливается наличие циклической зависимости между элементами упорядоченной коллекции C, при которой ее первый элемент совпадает с последним: $c_1 = c_n,\ c_1, c_n \in C, |C| = n$. Для множественных ассоциаций требуется, чтобы первый и последний элемент указывали на один и тот же экземпляр объекта, что задается с помощью оператора идентичности :=: языка EXPRESS. Характерным примером использования данного вида ограничений являются объектные типы замкнутых ломаных или кривых. Ограничение представляется следующей обобщенной функцией:

```
FUNCTION IsLoop(c: AGGREGATE OF GENERIC) : LOGICAL;
RETURN c[1]:=:c[SIZEOF(c)];
END_FUNCTION;
```

3.15.2 Sequence Pattern: Regular Identity

Ограничение устанавливает условие идентичности соседних элементов коллекции, повторяющееся с некоторым шагом. К данному типу можно отнести, например, ограничения непрерывности топологических объектов типа путей или циклов, то есть требование совпадения вершин, соответствующих концу предыдущего и началу следующего ребра в этих объектах. В обобщенном виде этот тип ограничений возможно представить с помощью следующей функции:

```
FUNCTION IsRegularIdentity(c: AGGREGATE OF GENERIC, step:
INTEGER) : LOGICAL;
LOCAL
      N : INTEGER := 0;
      P : LOGICAL := TRUE;
END_LOCAL;
N := SIZEOF(c)-step;
REPEAT i := step TO N BY step;
      P := P AND (c[i] :=: c[i+1]);
END_REPEAT;
RETURN (P);
END_FUNCTION;
```

3.15.3 Sequence Pattern: Distinction

Данные правила устанавливают требование неравенства хотя бы одного из элементов коллекции заданному литеральному значению: $\exists c \in C: c \neq const$. Их можно выразить с помощью следующей обобщенной функции:

```
FUNCTION HasDistinctionFromGiven(c: AGGREGATE OF GENERIC:t,
const: GENERIC:t) : LOGICAL;
RETURN (SIZEOF(QUERY(v <* c | v <> const)) > 0);
END_FUNCTION;
```

3.16 Dependency Patterns

Данные ограничения выражают зависимость между значениями нескольких атрибутов некоторого объектного типа в виде произвольных неравенств: $f(x_1, \dots, x_n) \geq 0$. Они могут разрешаться на основе предварительно определенных правил вида: $x_1 = \tilde{f}_1(x_2, \dots, x_n) + C_1$,..., $x_n = \tilde{f}_n(x_1, \dots, x_{n-1}) + C_n$, где $C_i, i = 1, \dots, n$ — некоторые константы. Эти правила могут задаваться

вручную на основе расширения синтаксиса языка EXPRESS либо генерироваться автоматически по сигнатуре ограничений с помощью специальных методов.

В качестве примера рассмотрим объектный тип `IfcCircleHollowProfileDef` для представления круглых пустотелых сечений. В нем установлено ограничение `WR1` на соотношение радиуса и толщины стенки отверстия. Тогда решающие правила могут быть заданы в виде синтаксической конструкции RULES FOR, расширяющий синтаксис языка EXPRESS. Зафиксировав значение одного из атрибутов, можно сгенерировать значение для другого с помощью данных правил. В приведенном ниже примере правила устанавливают, что толщина стенки отверстия составляет 1% от его радиуса.

```
ENTITY IfcCircleProfileDef SUPERTYPE OF (ONEOF
(IfcCircleHollowProfileDef))
 SUBTYPE OF (IfcParameterizedProfileDef);
       Radius : IfcPositiveLengthMeasure;
END_ENTITY;

ENTITY IfcCircleHollowProfileDef SUBTYPE OF
(IfcCircleProfileDef);
       WallThickness : IfcPositiveLengthMeasure;
 WHERE
       WR1 : WallThickness < SELF\IfcCircleProfileDef.Radius;
 RULES FOR WR1 :
       WallThickness = SELF\IfcCircleProfileDef.Radius * 0.01;
       SELF\IfcCircleProfileDef.Radius = WallThickness / 0.01;
 END_RULES;
END_ENTITY;
```

3.17 Complex Patterns

Подобные правила устанавливают ограничения целостности данных, задаваемые в виде сложных логических функций, сформулированных на языке EXPRESS. Обобщенное представление для данного семейства паттернов отсутствует, поскольку каждая такая функция является индивидуальной. В качестве примера рассмотрим объектный тип `IfcBSplineCurveWithKnots` для представления B-сплайн кривых с узлами, где правило корректной параметризации B-сплайна

ConsistentBSpline определяется функцией IfcConstraintsParamBSpline.

```
ENTITY IfcBSplineCurveWithKnots SUPERTYPE OF (ONEOF
       (IfcRationalBSplineCurveWithKnots)) SUBTYPE OF
(IfcBSplineCurve);
       KnotMultiplicities : LIST [2:?] OF INTEGER;
       Knots : LIST [2:?] OF IfcParameterValue;
       KnotSpec : IfcKnotType;
DERIVE
       UpperIndexOnKnots : INTEGER := SIZEOF(Knots);
 WHERE
       ConsistentBSpline : IfcConstraintsParamBSpline(Degree,
UpperIndexOnKnots,
                    UpperIndexOnControlPoints,
KnotMultiplicities, Knots);
       CorrespondingKnotLists : SIZEOF(KnotMultiplicities) =
UpperIndexOnKnots;
END_ENTITY;

FUNCTION IfcConstraintsParamBSpline( Degree, UpKnots, UpCp :
INTEGER;
       KnotMult : LIST OF INTEGER; Knots : LIST OF
IfcParameterValue ) : BOOLEAN;
LOCAL
       Result : BOOLEAN := TRUE;
       K, Sum : INTEGER;
END_LOCAL;
       Sum := KnotMult[1];
       REPEAT i := 2 TO UpKnots;
              Sum := Sum + KnotMult[i];
       END_REPEAT;
       IF (Degree < 1) OR (UpKnots < 2) OR (UpCp < Degree) OR
```

```
                        (Sum <> (Degree + UpCp + 2))
    THEN
            Result := FALSE;
            RETURN(Result);
    END_IF;
    K := KnotMult[1];
    IF (K < 1) OR (K > Degree + 1) THEN
            Result := FALSE;
            RETURN(Result);
    END_IF;
    REPEAT i := 2 TO UpKnots;
            IF (KnotMult[i] < 1) OR (Knots[i] <= Knots[i-1])
    THEN
                    Result := FALSE;
                    RETURN(Result);
            END_IF;
            K := KnotMult[i];
            IF (i < UpKnots) AND (K > Degree) THEN
                    Result := FALSE;
                    RETURN(Result);
            END_IF;
            IF (i = UpKnots) AND (K > Degree + 1) THEN
                    Result := FALSE;
                    RETURN(Result);
            END_IF;
    END_REPEAT;
    RETURN(result);
END_FUNCTION;
```

4. Использование библиотеки ограничений при решении задачи верификации моделей данных

Разработанная библиотека на языке EXPRESS, в состав которой входят перечисленные выше обобщенные функции для представления паттернов ограничений, может использоваться как при рефакторинге существующих

моделей, так и при разработке новых. Использование паттернов ограничений в спецификациях новых моделей позволяет улучшить их наглядность, облегчить их дальнейшее сопровождение и развитие и, в целом, ускорить их разработку. Кроме того, появляется возможность их анализа автоматизированными средствами программной инженерии.

Табл. 1. Методы разрешения паттернов ограничений целостности данных

Паттерн	Метод разрешения
Set	генерация значения, принадлежащего множеству допустимых
Range	генерация значения, лежащего в заданном интервале или полуинтервале
Constantation	генерация константного значения
Multiplicity	установка размера коллекции в соответствии с уточненным интервалом
Existence	генерация значений соответствующих атрибутов в обязательном порядке
Existence on Determinator	генерация значения определителя в первую очередь, затем в зависимости от него генерация значения для зависимого атрибута в обязательном или необязательном порядке
Type Refinement	выбор кандидатов ассоциируемых объектов из ограниченного соответствующим образом множества
Type Refinement on Determinator	генерация значения определителя в первую очередь, затем выбор кандидатов ассоциируемых объектов из ограниченного соответствующим образом множества
Subset	генерация значений элементов второй коллекции, затем заполнение первой коллекции некоторой случайной выборкой элементов из второй
Correlated Size	установка одинакового допустимого размера для обеих коллекций
Correlated	установка одного и того же типа данных из списка выбора

Type	селективного типа
Alternative Reference	установка одной из ассоциаций, затем выбор кандидатов для другой из ограниченного соответствующим образом множества
Unique Name	выбор кандидатов с уникальным именем либо генерация уникального имени при установлении ассоциативной связи
Equivalence	присвоение соответствующим атрибутам одинаковых значений
Sequence	генерация значений элементов коллекции в соответствии с заданной последовательностью
Dependency	поиск значения на основе интервальной арифметики и локального распространения
Complex	специальные методы генерации значений

Обсудим возможности применения этой библиотеки при решении задачи верификации моделей данных. В работе [39] был предложен комбинированный метод, применимый к анализу масштабных моделей данных. Он сначала разрешает сильно связанные ограничения кардинальности, уникальности и определяет необходимый размер семантически корректной коллекции объектов, а затем, изолированно обрабатывая индивидуальные ограничения, позволяет установить корректные значения атрибутов объектов и ассоциативные связи между ними, удовлетворяющие каждому из установленных ограничений.

На этапе генерации значений атрибутов и установления ассоциативных связей задача верификации моделей редуцируется к задаче удовлетворения ограничений (Constraint Satisfaction Problem, CSP) [40]. Методы ее решения предполагают наличие разрешающих правил для ограничений модели. Ручное создание подобных правил не представляется возможным вследствие масштабности индустриальных моделей данных, а их автоматизированное построение в общем случае является затруднительным, поскольку ограничения целостности данных, в особенности ограничения области определения значений, в языке EXPRESS могут представляться разнообразными синтаксическим конструкциями, включая определяемые пользователем функции и произвольные логические выражения. Таким образом, предлагается задействовать разработанную библиотеку ограничений и использовать предопределенные разрешающие правила для паттернов. Для существующих моделей данных следует предварительно провести

рефакторинг определенных в них ограничений с целью унификации их представления.

Для каждого из выделенных паттернов ограничений были разработаны методы разрешения. Примечательно, что для большинства из паттернов (15 из 17) эти методы оказались тривиальными и связанными с ограничением области определения при генерировании значений зависимых атрибутов (см. табл. 1).

Рассмотрим в качестве примера псевдокод функции для установки значений атрибутов в объектах типа `IfcPolygonalBoundedHalfSpace`, спецификация которого представлена на рис. 4:

```
function setIfcPolygonalBoundedHalfSpaceAttributes( o :
       IfcPolygonalBoundedHalfSpace )
begin_function
       if( not exists( o.Position ) )
       then
              IfcAxis2Placement3D pos :=
                     generate( "IFC4.IFCAXIS2PLACEMENT3D" );
              setIfcAxis2Placement3DAttributes( pos );
              o.Position := pos;
       end_if;
       if( not exists( o.PolygonalBoundary ) )
       then
              IfcBoundedCurve bound;
              (* select the type according to Where Rule
"BoundaryType" *)
              int typeN = rnd( 0, 1 );
              if( typeN = 0 )
              then
                     bound := generate( "IFC4.IFCPOLYLINE" );
                     (* set the value according to Where Rule
"BoundaryDim" *)
                     bound.Dim := 2;
                     setIfcPolylineAttributes( bound );
              else
```

```
                    bound := generate(
"IFC4.IFCCOMPOSITECURVE" );
                    (* set the value according to Where Rule
"BoundaryDim" *)
                    bound.Dim := 2;
                    setIfcCompositeCurveAttributes( bound );
            end_if;
            o.PolygonalBoundary := bound;
      end_if;
end_function;
```

Для разрешения ограничений, принадлежащих семейству паттернов Dependency, возможно применить методы локального распространения [41] при условии, что зависимости между переменными являются ациклическими. Способы конструирования и примеры разрешающих правил, необходимых для данных методов, были приведены в разделе 3.16.

Наиболее сложно разрешаются семейства паттернов ограничений Complex, представляемые произвольными логическими функциями. Однако в рассмотренных индустриальных моделях данных они не являются широко распространенными и встречаются в определениях геометрических объектов, которые в значительной степени стандартизированы. Несложно определить специальные методы генерации корректного геометрического представления, которые будут применимы к любой из моделей. Способы разрешения сложных ограничений в классах модели IFC 4 приведены в табл. 2.

Табл. 2. Способы разрешения сложных геометрических ограничений в модели IFC 4

Класс	Ограничение	Способ разрешения
IfcAxis2Placement3D (локальные системы координат)	AxisToRefDirPosition (оси не должны быть коллинеарными)	Создание взаимно перпендикулярных векторов ненулевой длины
IfcBSplineCurveWithKnots IfcBSplineSurfaceWithKnots (B-сплайн кривые и поверхности)	ConsistentBSpline UDirectionConstraints VDirectionConstraints (проверяется	Генерация B-сплайнов с помощью известных

	корректность параметризации B-сплайнов)	алгоритмов [42]
IfcExtrudedAreaSolid (3D объект, получаемый путем перемещения сечения в заданном направлении на фиксированное расстояние)	ValidExtrusionDirection (заданное направление не должно быть перпендикулярно оси Z в локальной системе координат)	Выбор направления перемещения сечения вдоль локальной оси Z
IfcExtrudedAreaSolidTapered (расширяет IfcExtrudedAreaSolid указанием концевой поверхности) IfcRevolvedAreaSolidTapered (3D объект, получаемый путем вращения сечения вдоль дуги с указанием концевой поверхности)	CorrectProfileAssignment (если тип концевой поверхности — IfcDerivedProfileDef, то она должна быть получена путем трансформации исходного сечения, в противном случае их типы должны совпадать и наследоваться от IfcParametrizedProfileDef)	Генерация концевой поверхности путем трансформации исходного сечения

Заметим, что ряд ограничений в индустриальных моделях данных может быть представлен в виде комбинации однотипных или различных паттернов, связанных логическими операциями, допустимыми в языке EXPRESS (AND, OR, XOR, NOT). К простейшим случаям относятся, например, вышеупомянутые неравенства вида $v \neq const$ (см. раздел 3.2) или комбинированный паттерн существования (см. раздел 3.5).

Логические выражения подобных комбинированных ограничений легко преобразуются к ДНФ. Тогда необходимо и достаточно разрешить паттерны в одном из дизъюнктов. Рассмотрим это на конкретном примере.

```
ENTITY IfcFixedReferenceSweptAreaSolid SUBTYPE OF
(IfcSweptAreaSolid);
```

```
        Directrix : IfcCurve;
        StartParam : OPTIONAL IfcParameterValue;
        EndParam : OPTIONAL IfcParameterValue;
        FixedReference : IfcDirection;
 WHERE
        DirectrixBounded : (EXISTS(StartParam) AND
EXISTS(EndParam)) OR
        (SIZEOF(['IFC4.IFCCONIC',
'IFC4.IFCBOUNDEDCURVE']*TYPEOF(Directrix)) = 1);
END_ENTITY;
```

Объектный тип `IfcFixedReferenceSweptAreaSolid` используется для представления твердых тел, получаемых путем перемещения сечений вдоль директриссы. В нем локальное правило `DirectrixBounded` устанавливает, что либо следует установить параметры начала и конца перемещения по директриссе, либо она должна быть кривой с конечной длиной или коническим сечением. Логическое выражение данного правила может быть переформулировано с использованием разработанной библиотеки следующим образом:

```
IsExist(StartParam) AND IsExist(EndParam) OR
RefinesType(Directrix, ['IFC4.IFCCONIC',
'IFC4.IFCBOUNDEDCURVE'], TRUE);
```

Тогда при установлении ассоциативной связи `Directrix` необходимо и достаточно ограничить множество объектов-кандидатов классами конических сечений или кривых с конечной длиной. При наличии таких кандидатов параметры `StartParam` и `EndParam` могут иметь неопределенные значения. При отсутствии кривых вышеперечисленных типов ассоциативная связь может быть установлена с произвольной кривой, но тогда необходимо сгенерировать значения для `StartParam` и `EndParam`.

5. Заключение

Таким образом, проведен анализ спецификаций индустриально значимого семейства объектно-ориентированных моделей данных (в частности, IFC, CIS/2, общих ресурсов STEP) и выделены паттерны ограничений целостности, используемые в них. Разработана библиотека обобщенных функций на языке EXPRESS для представления каждого из паттернов. Рассмотрена возможность применения данной библиотеки для решения задачи верификации моделей. Для каждого из выделенных паттернов предложен метод разрешения соответствующих ограничений.

Примечательно, что для большинства из паттернов эти методы оказались тривиальными и связанными с ограничением области определения при генерировании значений зависимых атрибутов.

Результаты проведенного исследования демонстрируют важные возможности унификации спецификаций индустриальных моделей данных, которые позволяют улучшить их наглядность, облегчить их дальнейшее сопровождение и развитие и, в целом, ускорить их разработку и упростить анализ с применением автоматизированных средств программной инженерии. Использование паттернов ограничений в спецификациях моделей может служить важной рекомендацией для индустриальных консорциумов и технических комитетов, занимающихся их разработкой и стандартизацией.

Литература

[1]. Model Driven Architecture: The Architecture of Choice for a Changing World. Executive Overview, February 2014. http://www.omg.org/mda/executive_overview.htm
[2]. Unified Modeling Language (UML), V2.4.1, Release Date: August 2011. http://www.omg.org/spec/UML/2.4.1
[3]. Meta Object Facility (MOF) Core, V2.4.2, Release Date: April 2014. http://www.omg.org/spec/MOF/2.4.2
[4]. XML Metadata Interchange (XMI), V2.4.2, Release Date: April 2014. http://www.omg.org/spec/XMI/2.4.2
[5]. Common Warehouse Metamodel (CWM), V1.1, Release Date: March 2003. http://www.omg.org/spec/CWM/1.1
[6]. ISO 10303-1:1994. Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles.
[7]. ISO 10303-11:2004. Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.
[8]. ISO 10303-21:2002. Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure.
[9]. ISO 10303-28:2007. Industrial automation systems and integration — Product data representation and exchange — Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas.
[10]. ISO/TS 10303-26:2011. Industrial automation systems and integration — Product data representation and exchange — Part 26: Implementation methods: Binary representation of EXPRESS-driven data.
[11]. ISO 10303-22:1998. Industrial automation systems and integration — Product data representation and exchange — Part 22: Implementation methods: Standard data access interface.
[12]. ISO 10303-203:1994. Industrial automation systems and integration — Product data representation and exchange — Part 203: Application protocol: Configuration controlled 3D designs of mechanical parts and assemblies.
[13]. ISO 10303-210:2014. Industrial automation systems and integration — Product data representation and exchange — Part 210: Application protocol: Electronic assembly, interconnect and packaging design.

[14]. ISO 10303-212:2001. Industrial automation systems and integration — Product data representation and exchange — Part 212: Application protocol: Electrotechnical design and installation.
[15]. ISO 10303-214:2001. Industrial automation systems and integration — Product data representation and exchange — Part 214: Application protocol: Core data for automotive mechanical design processes.
[16]. ISO 10303-215:2004. Industrial automation systems and integration — Product data representation and exchange — Part 215: Application protocol: Ship arrangement.
[17]. ISO 10303-216:2003. Industrial automation systems and integration — Product data representation and exchange — Part 216: Application protocol: Ship moulded forms.
[18]. ISO 10303-218:2004. Industrial automation systems and integration — Product data representation and exchange — Part 218: Application protocol: Ship structures.
[19]. ISO 10303-233:2012. Industrial automation systems and integration — Product data representation and exchange — Part 233: Application protocol: Systems engineering.
[20]. ISO 10303-236:2006. Industrial automation systems and integration — Product data representation and exchange — Part 236: Application protocol: Furniture catalog and interior design.
[21]. IFC4 Release Summary, March 2013. http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release/ifc4-release-summary
[22]. Khemlani L. The CIS/2 Format: Another AEC Interoperability Standard. // AECbytes Newsletter, July 27, 2005. http://www.aecbytes.com/buildingthefuture/2005/CIS2format.html
[23]. ISO 15926-1:2004. Industrial automation systems and integration — Integration of life-cycle data for process plants including oil and gas production facilities — Part 1: Overview and fundamental principles.
[24]. CALS-стандарты. // Автоматизация проектирования, № 2, 1997. http://www.osp.ru/ap/1997/02/13031610
[25]. Семенов В.А., Морозов С.В., Тарлапан О.А. Инкрементальная верификация объектно-ориентированных данных на основе спецификации ограничений. // Труды Института системного программирования / под ред. В.П. Иванникова, т. 8, ч. 2, 2004, с. 21–52.
[26]. Семенов В.А., Ерошкин С.Г., Караулов А.А., Энкович И.В. Семантическая реконсиляция прикладных данных на основе моделей. // Труды Института системного программирования / под ред. В.П. Иванникова, т. 13, ч. 2, 2007, с. 141–164.
[27]. ISO 10303-14:2005. Industrial automation systems and integration — Product data representation and exchange — Part 14: Description methods: The EXPRESS-X language reference manual.
[28]. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), V1.2, Release Date: February 2015. http://www.omg.org/spec/QVT/1.2
[29]. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999. http://www.w3.org/TR/xslt
[30]. Nell J. STEP on a page, April 2003. http://www.mel.nist.gov/sc5/soap
[31]. buildingSmart® — International home of openBIM®, October 2014. http://www.buildingsmart.org
[32]. ISO 16739:2013. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries.

[33]. ISO 10303-41:2005. Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resource: Fundamentals of product description and support.
[34]. ISO 10303-42:2003. Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation.
[35]. ISO 10303-43:2000. Industrial automation systems and integration — Product data representation and exchange — Part 43: Integrated generic resource: Representation structures.
[36]. ISO 10303-45:2008. Industrial automation systems and integration — Product data representation and exchange — Part 45: Integrated generic resource: Material and other engineering properties.
[37]. ISO 10303-46:2011. Industrial automation systems and integration — Product data representation and exchange — Part 46: Integrated generic resource: Visual presentation.
[38]. Staab S., Studer R. (eds.). Handbook on Ontologies, Second Edition. Springer-Verlag, Berlin Heidelberg, 2009.
[39]. Семенов В.А., Морозов С.В., Ильин Д.В. Комбинированный метод верификации масштабных моделей данных. // Труды Института системного программирования / под ред. В.П. Иванникова, т. 26, вып. 2, 2014, с. 197–230. DOI: 10.15514/ISPRAS-2014-26(2)-9
[40]. Tsang E. Foundations of constraint satisfaction. Academic Press Limited, London & San-Diego, 1993.
[41]. Семенов В.А., Сидяка О.В. Теоретические и экспериментальные оценки сложности методов локального распространения в задачах программирования в ограничениях. // Труды Института системного программирования / под ред. В.П. Иванникова, т. 19, 2010, с. 117–134.
[42]. Роджерс Д., Адамс Дж. Математические основы машинной графики. М.: Мир, 2001.

A Constraint Library for Specification of Industrial Data Models

[1, 2]*S.V. Morozov <serg@ispras.ru>*
[1]*D.V. Ilyin <denis.ilyin@ispras.ru>*
[1, 3]*V.A. Semenov <sem@ispras.ru>*
[1, 2]*O.A. Tarlapan <oleg@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia*
[2]*Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation*
[3]*Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*

Abstract. The paper is addressed to an analysis of object-oriented data models specified at EXPRESS language and widely used in industrial applications. These models play important role for achievement of software interoperability and system integration in accordance with STEP standard family (ISO 10303). The examples of such models are STEP application protocols for machinery construction, automobile industry, shipbuiling, electronics, electrical engineering, systems engineering, furniture production as well as IFC (ISO 16739) model for architecture, engineering and construction, CIS/2 model for manufacturing using constructional steel work, POSC Caesar (ISO 15296) model for oil and gas producing industry. The purpose of the performed analysis is to unify representation of data integrity constraints typically used in the models by means of identification of constraint patterns. The identified patterns are specified at EXPRESS language as an unified constraint library that can be applied both on refactoring of the existing models and on development of new ones. Utilizing the constraint library users can improve clearness of the specifications, to simplify their maintenance and evolution and, on the whole, to accelerate their development. Besides, CASE tools can be effectively applied to analyze the specifications in highly automatic way. Possibility to apply the library for verification of the data models is also discussed in the paper. Rules for resolving the appropriate constraints have been proposed for each pattern. The constraint library can be recommended to industrial consortiums and technical committees that are engaged in development and standardization of the data models. The work is supported by RFBR (grant 13-07-00390).

Keywords: object-oriented modeling, EXPRESS, STEP, IFC, CIS/2, constraint patterns, model verification

DOI: 10.15514/ISPRAS-2015-27(4)-5

For citation: Morozov S.V., Ilyin D.V., Semenov V.A., Tarlapan O.A. A Constraint Library for Specification of Industrial Data Models. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 69-110 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-5.

References

[1]. Model Driven Architecture: The Architecture of Choice for a Changing World. Executive Overview, February 2014. http://www.omg.org/mda/executive_overview.htm
[2]. Unified Modeling Language (UML), V2.4.1, Release Date: August 2011. http://www.omg.org/spec/UML/2.4.1
[3]. Meta Object Facility (MOF) Core, V2.4.2, Release Date: April 2014. http://www.omg.org/spec/MOF/2.4.2
[4]. XML Metadata Interchange (XMI), V2.4.2, Release Date: April 2014. http://www.omg.org/spec/XMI/2.4.2
[5]. Common Warehouse Metamodel (CWM), V1.1, Release Date: March 2003. http://www.omg.org/spec/CWM/1.1
[6]. ISO 10303-1:1994. Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles.
[7]. ISO 10303-11:2004. Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.
[8]. ISO 10303-21:2002. Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure.
[9]. ISO 10303-28:2007. Industrial automation systems and integration — Product data representation and exchange — Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas.
[10]. ISO/TS 10303-26:2011. Industrial automation systems and integration — Product data representation and exchange — Part 26: Implementation methods: Binary representation of EXPRESS-driven data.
[11]. ISO 10303-22:1998. Industrial automation systems and integration — Product data representation and exchange — Part 22: Implementation methods: Standard data access interface.
[12]. ISO 10303-203:1994. Industrial automation systems and integration — Product data representation and exchange — Part 203: Application protocol: Configuration controlled 3D designs of mechanical parts and assemblies.
[13]. ISO 10303-210:2014. Industrial automation systems and integration — Product data representation and exchange — Part 210: Application protocol: Electronic assembly, interconnect and packaging design.
[14]. ISO 10303-212:2001. Industrial automation systems and integration — Product data representation and exchange — Part 212: Application protocol: Electrotechnical design and installation.
[15]. ISO 10303-214:2001. Industrial automation systems and integration — Product data representation and exchange — Part 214: Application protocol: Core data for automotive mechanical design processes.
[16]. ISO 10303-215:2004. Industrial automation systems and integration — Product data representation and exchange — Part 215: Application protocol: Ship arrangement.
[17]. ISO 10303-216:2003. Industrial automation systems and integration — Product data representation and exchange — Part 216: Application protocol: Ship moulded forms.
[18]. ISO 10303-218:2004. Industrial automation systems and integration — Product data representation and exchange — Part 218: Application protocol: Ship structures.

[19]. ISO 10303-233:2012. Industrial automation systems and integration — Product data representation and exchange — Part 233: Application protocol: Systems engineering.
[20]. ISO 10303-236:2006. Industrial automation systems and integration — Product data representation and exchange — Part 236: Application protocol: Furniture catalog and interior design.
[21]. IFC4 Release Summary, March 2013. http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release/ifc4-release-summary
[22]. Khemlani L. The CIS/2 Format: Another AEC Interoperability Standard. // AECbytes Newsletter, July 27, 2005. http://www.aecbytes.com/buildingthefuture/2005/CIS2format.html
[23]. ISO 15926-1:2004. Industrial automation systems and integration — Integration of life-cycle data for process plants including oil and gas production facilities — Part 1: Overview and fundamental principles.
[24]. CALS-standarty [CALS standards]. // Avtomatizatsiya proektirovaniya [Computer-aided design], no. 2, 1997. http://www.osp.ru/ap/1997/02/13031610 (in Russian).
[25]. Semenov V.A., Morozov S.V., Tarlapan O.A. Inkremental'naya verifikatsiya ob'ektno-orientirovannyh dannyh na osnove spetsifikatsii ogranichenij [Incremental verification of object-oriented data based on specification of constraints]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 8, no. 2, 2004, pp. 21–52 (in Russian).
[26]. Semenov V.A., Eroshkin S.G., Karaulov A.A., Enkovich I.V. Semanticheskaya rekonsilyatsiya prikladnyh dannyh na osnove modelej [Model-based semantic reconciliation of applied data]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 13, no. 2, 2007, pp. 141–164 (in Russian).
[27]. ISO 10303-14:2005. Industrial automation systems and integration — Product data representation and exchange — Part 14: Description methods: The EXPRESS-X language reference manual.
[28]. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), V1.2, Release Date: February 2015. http://www.omg.org/spec/QVT/1.2
[29]. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999. http://www.w3.org/TR/xslt
[30]. Nell J. STEP on a page, April 2003. http://www.mel.nist.gov/sc5/soap
[31]. buildingSmart® — International home of openBIM®, October 2014. http://www.buildingsmart.org
[32]. ISO 16739:2013. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries.
[33]. ISO 10303-41:2005. Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resource: Fundamentals of product description and support.
[34]. ISO 10303-42:2003. Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation.
[35]. ISO 10303-43:2000. Industrial automation systems and integration — Product data representation and exchange — Part 43: Integrated generic resource: Representation structures.
[36]. ISO 10303-45:2008. Industrial automation systems and integration — Product data representation and exchange — Part 45: Integrated generic resource: Material and other engineering properties.

[37]. ISO 10303-46:2011. Industrial automation systems and integration — Product data representation and exchange — Part 46: Integrated generic resource: Visual presentation.

[38]. Staab S., Studer R. (eds.). Handbook on Ontologies, Second Edition. Springer-Verlag, Berlin Heidelberg, 2009.

[39]. Semenov V.A., Morozov S.V., Ilyin D.V. Kombinirovannyj metod verifikatsii masshtabnyh modelej dannyh [A combined method for verification of large-scale data models]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 26, no. 2, 2014, pp. 197–230 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-9.

[40]. Tsang E. Foundations of constraint satisfaction. Academic Press Limited, London & San-Diego, 1993.

[41]. Semenov V.A., Sidyaka O.V. Teoreticheskie i eksperimental'nye otsenki slozhnosti metodov lokal'nogo rasprostraneniya v zadachah programmirovaniya v ogranicheniyah [Theoretical and practical complexity estimates for local propagation methods in constraint-based programming applications]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 19, 2010, pp. 117–134 (in Russian).

[42]. Rogers D.F., Adams J.A. Mathematical Elements of Computer Graphics, Second Edition. McGraw Hill, 1990.

Совместная вероятностная тематическая модель для идентификации проблемных высказываний, связанных нарушением функциональности продуктов

Е.В. Тутубалина <elvtutubalina@kpfu.ru>
Казанский (Приволжский) федеральный университет,
420008, Россия, г. Казань, ул. Кремлевская, дом 18.

Аннотация. В статье исследуется задача автоматического извлечения информации о существовании различных проблем с продуктами из отзывов пользователей. В последние десятилетия на рынке потребительских товаров появилась резкая динамика увеличения количества технически сложных товаров. У покупателей возникают претензии по поводу удобства использования продукта наряду с ненадлежащим техническим качеством. Пользователи публикуют свои мнения о сложностях в использовании продуктов, что может оказывать влияние на процесс принятия решения о покупке продуктов потенциальными потребителями. Для достижения целей исследования предложены две тематические модели на основе латентного размещения Дирихле, позволяющие совместно учитывать несколько типов информации для идентификации проблемных высказываний. Предложенные алгоритмы моделируют распределение слов в документе, учитывая взаимосвязь между скрытыми тематической, тональной и проблемной переменными. Результаты экспериментального исследования анализируются в статье в сравнении с результатами популярных вероятностных моделей для задач анализа мнений, в качестве критериев оценки используются стандартные метрики качества систем анализа текстов и перплексия контрольных данных (perplexity). Для качественной оценки тематических распределений моделей был проведен анализ тем, подтверждающий целесообразность определения тональности для критических высказываний пользователей. Эксперименты показали, что наилучшие результаты классификации фраз о проблемах в использовании продуктов показывают предложенные модели, использующие совместную информацию из отзывов пользователей на русском и английском языках.

Ключевые слова: отзывы пользователей, латентное размещение Дирихле, Latent Dirichlet Allocation, совместная вероятностная модель, извлечение проблемных высказываний

DOI: 10.15514/ISPRAS-2015-27(4)-6

Для цитирования: Тутубалина Е.В. Совместная вероятностная тематическая модель для идентификации проблемных высказываний, связанных нарушением функциональности продуктов. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 87-96. DOI: 10.15514/ISPRAS-2015-27(4)-6.

1. Введение

В последние десятилетия на рынке потребительских товаров появляется все больше технически сложных товаров. Это связано, прежде всего, с развитием технологических инноваций, что приводит к постоянному увеличению конкретных видов компьютерных продуктов, и с концепцией соединения разной функциональности в едином устройстве. В связи с этим у покупателей возникают претензии по поводу удобства использования продукта наряду с ненадлежащим техническим качеством. Многие покупатели осуществляют возврат товаров компаниям даже, если товар работает исправно согласно государственным стандартам и техническим отчетам компаний. Описанные примеры иллюстрируют необходимость извлечения информации из отзывов пользователей о существовании проблем с теми или иными продуктами для обеспечения высокого качества продукции и устранения затруднений, влияющих на работу продуктов. Высокое качество продукции и услуг является свойством, которое определяет их конкурентоспособность в условиях рыночной экономики [1]. В данной работе исследуется корреляция между проблемными индикаторами, эмоционально-окрашенными (тональными) словами пользователя и темами внутри отзывов о товарах.

Описание проблемной ситуации с продуктами может сопровождаться негативными, позитивными или нейтральными высказываниями относительно аспектных терминов (целевых объектов), о которых высказывание было сделано. Описания технических проблем, связанных с нарушением функциональности продуктов, не содержат в себе явных тональных слов, сообщая нейтральную информацию о действительном событии. В отличие от этого, описание проблем с ухудшением эффективности, продуктивности и удобства использования продуктов характерно тем, что содержит некую тональную оценку относительно разных категорий целевых объектов. Например, пользователи электронных устройств могут быть недовольны медленной ответной реакцией дисплея, низкой эффективностью батареи, слишком громким или тихим звуком динамика, избыточным количеством функций, короткими проводами, недостаточно ярким цветом корпуса. Это объясняет необходимость агрегировать совместную информацию, влияющую на распределение слов в отзыве пользователя, о тематической категории целевого объекта, тональном контексте (позитивном, нейтральном, негативном) и знаниях о проблемных высказываниях.

В мировой науке существует небольшое количество работ, посвященных задаче анализа высказываний пользователей, на предмет обнаруженных ими проблем в использовании тех или иных устройств [2-5]. В основном эти

исследования больше сосредоточены над созданием методов, основанных на словарях индикативных конструкций [2, 3], частичном обучении с применением правил или синтаксических конструкциях [4, 5], не анализируя возможность внедрения дополнительных знаний о тональности слов, кроме определения негативных высказываний. В данной работе представлены тематические модели на основе модели латентного размещения Дирихле (Latent Dirichlet Allocation, LDA) [6], направленные на объединение задач идентификации категорий аспектных терминов, определения тональности и идентификации проблем с продуктами в отзывах.

В работе предлагаются две вероятностные модели: (i) совместная модель *тема-тональность-проблема* (topic-sentiment-problem model, TSPM), моделирующая слова в документе в зависимости от нескольких скрытых переменных; (ii) совместная модель *оценка пользователя-тема-тональность-проблема* (rating-aware topic-sentiment-problem model, RTSPM), являющейся модификацией модели TSPM с добавлением известной оценки продукта пользователем. В качестве критериев оценки качества моделей используется перплексия (perplexity). В качестве критериев оценки качества моделей в задаче классификации использованы популярные метрики задач автоматической обработки естественного языка такие, как аккуратность (accuracy), точность (precision), полнота (recall) и F-мера (F-measure). Результаты работы предложенных методов оценены на корпусах текстов с отзывами пользователей о компьютерах, машинах, инструментам для дома и детских товарах.

Статья состоит из следующих разделов: в разделе 2 обсуждается современное состояние исследований по задаче анализа мнений, в разделе 3 приводятся описания вероятностных моделей. Раздел 4 посвящен статистическому оцениванию предложенных моделей с помощью сэмплирования Гиббса. В разделе 5 анализируются результаты экспериментов в сравнении с популярными вероятностными моделями. В разделе 6 обсуждаются выводы, сделанные на основе экспериментов, и направления дальнейшей работы.

2. Современное состояние исследований

В мировой науке подавляющее большинство работ по анализу мнений пользователей посвящено английскому языку и методам определения эмоциональной окраски текста относительно аспектных терминов (aspect-based sentiment analysis), подробный обзор которых описан в работах [7, 8]. Популярными подходами к задаче анализа тональности являются подходы, основанные на знаниях в виде словарей и использующие статистические метрики или машинное обучение для задачи классификации мнений.

В настоящий момент доминирующими методами являются алгоритмы на основе модели латентного размещения Дирихле [6] для задачи определения аспектных терминов, предметно-ориентированных тональных индикаторов и тематической категоризации аспектов продукта. Это связано с тем, что

создание предметно-ориентированной обучающей выборки для классификатора требует больших затрат времени в то время, как вероятностные модели позволяют использовать коллекции неразмеченных документов для нахождения скрытых переменных. В работах [9-12] представлены тематические модели, направленные на объединение задач идентификации аспектных терминов в отзыве и определения тональности для этих объектов. Эти модели используют словарь позитивных и негативных слов для задания гиперпараметра β (априорное распределение Дирихле на мультиномиальном распределении Ф в пространстве слов для темы).

В работе [10] авторы описывают модель *тональность-тема* (joint sentiment-topic model, JST) и модель *тема-тональность* (Reverse-JST), добавляя скрытую тональную переменную для моделей. Авторы предполагают, что в JST распределение тем в каждом документе зависит от тонального распределения, в Reverse-JST верно обратное. В моделях предполагается, что слово в документе порождено некоторой латентной темой и некоторой латентной тональной меткой. Таким образом, каждой тональной метке соответствует мультиномиальное распределение в пространстве тем, парам (тональная метка, тема) соответствует мультиномиальное распределение в пространстве слов. Эксперименты показали, что модели показывают наилучший результат классификации в нескольких доменах (книги, фильмы, электроника). В работе [11] описана объединенная модель *аспект-тональность* (aspect and sentiment unification model, ASUM). Под аспектом понимается тема в отзывах пользователей. Авторы полагают, что каждое предложение отзыва принадлежит одной теме и тональности. ASUM моделирует аспекты из мультиномиального распределения в пространстве тональности для предложения, слово порождено некоторым аспектом и тональностью предложения. Эксперименты показывают, что ASUM показывает лучшие результаты тональной классификации по сравнению с JST. В работе (Yang et al., 2015) представлена модификация LDA, названная User-aware Sentiment Topic Models (USTM), которая включает в распределения метаданные профайлов пользователя (геолокацию, возраст, пол) и словари эмоционально-окрашенной лексики для определения связи между тематическими наборами аспектов и категориями пользователей. Модель показывает наилучшие результаты классификации отзывов о машинах и ресторанах по сравнению с популярными вероятностными моделями JST и ASUM.

Задача анализа высказываний пользователей, на предмет обнаруженных ими проблем в использовании тех или иных устройств, является менее изученной. Существует несколько работ по классификации фраз отзывов об электронных устройствах [2-5]. В авторе [2] использует метод машинного обучения (метод максимальной энтропии) для извлечения аспектов из коротких сообщений о компании AT&T. Автор использует набор признаков, основанных на словарях позитивных и негативных слов, для обучения классификатора, однако не

приводит анализ важности созданных признаков в задаче обнаружения проблем. В работе [3] используются методы, основанные на правилах, для идентификации проблем в предложениях отзывов пользователей. Авторы используют словарь негативных слов для идентификации проблемных высказываний, не анализируя влияние нейтральной или позитивной тональности. В работе [5] авторы описывают модификацию LDA для идентификации целевых объектов и проблемных высказываний: модификация моделирует два распределения в пространстве аспектных терминов и в пространстве проблемных индикаторов. В качестве целевых объектов рассматриваются существительные отзывов, проблемными индикаторами считаются прочие слова (глаголы, наречия, прилагательные и пр.). Каждому документу соответствует два распределение в пространстве тем, где проблемные индикаторы зависят от темы и проблемной метки. В качестве критериев оценки метода использовалась перплексия, эффективность модели в задаче классификации не анализировалась. В работе [4] результаты экспериментов показали, что метод, основанный на частичном обучении с применением правил, может показать сопоставимые результаты относительно методов машинного обучения, для которых требуется размеченная обучающая коллекция. Для категоризации слов автор использует стандартную модель LDA, показывая, что темы из словосочетаний и глагольных групп более информативны, чем отдельные слова. Авторам статьи неизвестны работы, посвященные задаче автоматического извлечения информации о существовании различных проблем с товарами на русском языке, использующие вероятностные тематические модели.

3. Совместные вероятностные тематические модели для задачи идентификации проблем

Определим точную формулировку описанной задачи. Пусть $P = \{P_1, P_2, \dots, P_m\}$ – множество продуктов (сервисов, товаров), выпускаемое компаниями на потребительском рынке. Для каждого продукта $P_i \in P$ задано множество отзывов пользователей $D = \{d_1, d_2, \dots, d_n\}$. Каждый продукт $P_i \in P$ состоит из множества целевых объектов (компонентов, составных частей) $T = \{t_1, t_2, \dots, t_k\}$. В некоторых отзывах пользователи сообщают о дефектах или нарушениях функционирования продуктов. Проблемным высказыванием называются фраза пользователя в отзыве, содержащая явное указание на трудность в использовании тех или иных продуктов, невозможность использования продуктов вследствие ошибки (бага, дефекта) или сложности в использования продукта. Например,

- «Когда машину только купили, *дверь багажника закрывалась плохо*, приходилось очень сильно хлопать».
- «Один минус в салоне, что *нет бортового компьютера*».

- «Телефоном доволен, даже не смотря на то, что *пластиковый корпус потускнел*».
- «В целом все было отлично, единственное, что не понравилось так это то, что в зимние дни *батарея разряжается быстрее* из-за металлической крышки».
- «Когда едешь по не асфальтированной дороге, возникает *шум плюс скрип в салоне*».

В целом, задача автоматического извлечения информации о существовании различных проблем с товарами состоит из трех основных подзадач:

1. идентификация проблемных высказываний из текстов пользователей;
2. извлечение проблемных фраз по отношению к целевым объектам, зависящих от конкретной предметной области;
3. резюмирование целевых объектов продуктов и проблемных индикаторов по тематическим категориям.

В данной работе рассматривается задача резюмирования целевых объектов продуктов по тематическим категориям с учетом информации о тональных и проблемных индикаторах.

После анализа высказываний из отзывов пользователей мы выделили несколько типов фраз с различной тональной окраской. Пользователь может описывать технические дефекты и неполадки в процессе использования продукта в отзыве, который не содержит эмоционально-окрашенных слов (например, *«не могу открыть флэшку», «машине требуется ремонт»*). Проблемное высказывание может сопровождаться негативной или позитивной тональностью, если пользователь описывает затруднения с комфортным использованием продукта относительно разных категорий целевых объектов (например, «в ресторане *отвратительное обслуживание*», «*не слишком чувствительный сенсор*», «батарея держится *долго, но меньше*, чем заявлено в инструкции», «было бы лучше, *если бы не было шумов* от двигателя»). Таким образом, в зависимости от темы (категории) целевого объекта, пользователь может использовать слова с различной тональной степенью, влияющие на общее представление о проблемной ситуации.

Для достижения целей исследования в статье предложены два совместные тематические модели:

1. модель *тема-тональность-проблема* (topic-sentiment-problem model, TSPM)
2. модель *оценка пользователя-тема-тональность-проблема* (rating-aware topic-sentiment-problem Model, RTSPM).

Графические представления TSPM и RTSPM моделей приведены на рис. 1.

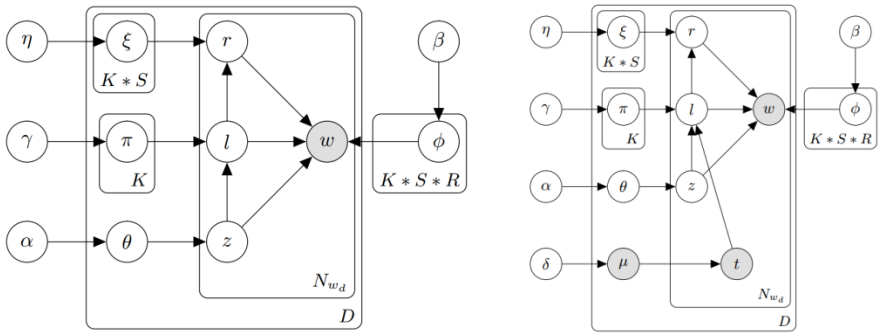

Рис. 1. Совместные вероятностные модели: (1) TSPM и (2) RTSPM

3.1 Вероятностная модель тема-тональность-проблема

В рамках TSPM модели каждой теме соответствует мультиномиальное распределение в пространстве слов. Для каждого слова в документе тема z выбирается из мультиномиального распределения θ, затем выбирается тональная метка l из мультиномиального распределения π, соответствующего теме z, затем выбирается проблемная метка r из мультиномиального распределения ε, соответствующего паре (тема z, тональная метка l). Наконец, слово w выбирается из распределения Φ соответствующего теме z. тональной метке l, проблемной метке r. Таким образом, слова в документах порождаются в зависимости от некоторой латентной темы, латентной тональной и проблемной меток. Графическое представление RTSPM приведено на рис.1 (1). В таблице 1 приводится список основных обозначений, используемых в моделях.

Табл. 1. Основные обозначения в TSPM и RTSPM моделях.

Символ	Описание
D	число документов
V	размер словаря слов в коллекции
N	число слов в коллекции
K	число тем
S	Число тональных классов
R	Число проблемных классов
w_d	вектор слов документа d
N_{w_d}	число слов в документе d
θ_d	мультиномиальное распределение в пространстве тем с параметром α
π_z	мультиномиальное распределение в пространстве тональных меток для темы z с параметром β

$\varepsilon_{z,l}$	мультиномиальное распределение в пространстве проблемных меток для пары (тема *z*, тональная метка l) с параметром β
$\Phi_{z,l,r}$	мультиномиальное распределение в пространстве слов для троек (тема *z*, тональная метка *l*, проблемная метка *r*) с параметром β
t_{di}	множество оценочных тэгов, присвоенных *i*-му слову в документе *d*
z_{di}	множество тем, присвоенных *i*-му слову в документе *d*
l_{di}	множество тональных меток, присвоенных *i*-му слову в документе *d*
r_{di}	множество проблемных меток, присвоенных *i*-му слову в документе *d*
α	априорное распределение Дирихле на параметры θ
β	априорное распределение Дирихле на параметры Φ
η	априорное распределение Дирихле на параметры ξ
γ	априорное распределение Дирихле на параметры π
δ	априорное распределение Дирихле на параметры ψ

В данной работе число тональных классов равно 3 (позитивный, нейтральный, негативный), число проблемных классов равно 2 (проблемный класс и класс с отсутствием указаний на проблемы). Совместная вероятность слов, тем, тональных и проблемных меток для TSPM могут быть посчитана следующим образом:

$$p(w,z,r,l) = p(w|z,l,r)\, p(r|z,l)\, p(l|z)\, p(z)$$

TSPM характерен следующий порождающий процесс:

- для каждой тройки (тема *z*, тональная метка *l*, проблемная метка *r*) выбирается распределение слов в каждой теме $\Phi_{z,l,r} \sim Dir\,(\beta_{zlk})$ $(l \in \{neu, pos, neg\}, r \in \{pr, no-pr\})$
- для каждого документа (отзыва) *d*:
  - выбирается случайный вектор $\theta_d \sim Dir\,(\alpha)$
  - для каждой темы *z* выбирается вектор тональных меток $\pi_d^z \sim Dir\,(\gamma)$
  - для каждой пары (*z,l*) выбирается вектор проблемных меток $\xi_d^{z,l} \sim Dir\,(\eta)$
  - для каждого слова w_i в документе *d*:
    - выбирается тема $z_{d,w_i} \sim Mult\,(\theta_d)$
    - для каждой темы выбирается тональная метка $l_{d,w_i} \sim Mult\,(\pi_d^z)$
    - для каждой пары (тема, тональная метка) выбирается проблемная метка $r_{d,w_i} \sim Mult\,(\xi_d^{z,l})$
    - для каждой тройки (тема, тональная метка, проблемная метка) выбирается слово w_i из распределения в пространстве слов с параметром β, зависящее от комбинации $(z_{d,w_i}, l_{d,w_i}, r_{d,w_i})$

3.2 Вероятностная модель оценка пользователя-тема-тональность-проблема

Модель *оценка пользователя-тема-тональность-проблема* (rating-aware topic-sentiment-problem Model, RTSPM) является модификацией предложенной модели TSPM. Графическое представление RTSPM приведено на рис. 1 (2). В рамках RTSPM добавляется переменная, связанная с оценкой пользователя по отношению к продукту, который описывается в отзыве. На многих онлайн-ресурсах пользователи оценивают продукт с точки зрения того, как хорошо он работает по пятизвездочному рейтингу качества. Затем пользователи выражают свои мнения, используя эмоционально-окрашенные слова, которые коррелируют с выставленным значением.

RTSPM модель рассматривает документ с известной оценкой рейтинга: каждое слово документа взаимосвязано с оценочным тэгом *t*, эквивалентного пятизвездочному рейтингу. Для каждого слова в документе тема *z* выбирается из мультиномиального распределения θ, затем для пары (тэг *t*, тема *z*) выбирается тональная метка l из мультиномиального распределения π, затем выбирается проблемная метка *r*, соответствующая паре (тэг *t*, тема *z*, тональная метка *l*). *С*лово *w* выбирается из распределения Φ соответствующего тэгу *t*, теме *z*, тональной метке *l*, проблемной метке *r*.

RTSPM характерен следующий порождающий процесс:

- для каждой тройки (тема *z*, тональная метка l, проблемная метка r) выбирается распределение слов в каждой теме $\Phi_{z,l,r} \sim Dir\ (\beta_{zlk})$ ($l \in \{neu, pos, neg\}, r \in \{pr, no-pr\}$)
- для каждого документа (отзыва) d с известным рейтингом документа:
  - выбирается случайный вектор $\theta_d \sim Dir\ (\alpha)$
  - для каждой темы z и рейтинга t выбирается вектор тональных меток $\pi_d^{t,z} \sim Dir\ (\gamma)$
  - для каждой пары (*t,z,l)* выбирается вектор проблемных меток $\xi_d^{t,z,l} \sim Dir\ (\eta)$
  - для каждого слова w_i с оценкой *t* в документе *d*:
    - присваивается оценочный тэг t в соответствии с рейтингом документа
    - выбирается тема $z_{d,w_i} \sim Mult\ (\theta_d\ )$
    - для каждой темы выбирается тональная метка $l_{d,w_i} \sim Mult\ (\pi_d^{t,z})$
    - для каждой пары (оценочный тэг, тема, тональная метка) выбирается проблемная метка $r_{d,w_i} \sim Mult\ (\xi_d^{t,z,l})$
    - для каждой тройки (тема, тональная метка, проблемная метка) выбирается слово w_i из распределения в пространстве слов с параметром β, зависящее от комбинации $(z_{d,w_i}, l_{d,w_i}, r_{d,w_i})$ и *t*.

4. Статистическое оценивание предложенных моделей

Данный раздел описывает алгоритм статистического оценивания предложенных TSPM и RTSPM моделей. Для решения задачи статистического оценивания применительно к тематической модели LDA существует несколько алгоритмов: сэмплирование Гиббса (Gibbs sampling), вариационный вывод (variational inference), распространение математического ожидания (expectation propagation) [6, 13, 14]. В данной работе применяется сэмплирование Гиббса для оценивания параметров модели, поскольку такой подход позволяет эффективно находить скрытые темы в корпусах текстов [15].

Используя сэмплирование Гиббса, присвоенные скрытые параметры в модели TSPM могут быть выбраны по следующей формуле:

$$P(\mathbf{z}_{d,i}=k,\mathbf{l}_{d,i}=l,\mathbf{r}_{d,i}=r|\mathbf{w}_{d,i}=w,\mathbf{z}_{-(d,i)},\mathbf{l}_{-(d,i)},\mathbf{r}_{-(d,i)},\alpha,\beta,\gamma,\eta)\propto \frac{n_{k,l,r,w}^{-(d,i)}+\beta_{l,r}^{w}}{n_{k,l,r}^{-(d,i)}+\sum_{j=1}^{V}*\beta_{l,r}^{j}}\frac{n_{d,k,l,r}^{-(d,i)}+\eta}{n_{d,k,l}^{-(d,i)}+R*\eta}\frac{n_{d,k,l}^{-(d,i)}+\gamma}{n_{d,k}^{-(d,i)}+L*\gamma}\frac{n_{d,k}^{-(d,i)}+\alpha}{n_{d}^{-(d,i)}+K*\alpha} \quad (1)$$

где $n_{k,l,r,w}$ означает количество раз, когда слову *w* присвоена тема *k*, тональная метка l и проблемная метка *r* в коллекции документов, $n_{k,l,r}$ определяет общее количество слов, которым присвоена тройка *(k,l,r)*. Общее число $n_{d,k,l,r}$ обозначает количество слов в документе *d*, присвоенных теме *k*, тональной метке l и проблемной метке *r*, $n_{d,k,l}$ определяет количество слов документа *d*, присвоенных теме *k* и тональной метке l. Общее число $n_{d,k}$ обозначает количество слов в документе *d*, присвоенных теме *k*, n_d обозначает общее число слов в документе *d*. Индекс *–(d,i)* обозначает количество элементов, исключая текущие значения слова *w* в документе *d*.

Используя схожие обозначения в сэмплировании Гиббса, присвоенные скрытые параметры могут быть выбраны в модели RTSPM по следующей формуле:

$$P(\mathbf{z}_{d,i}=k,\mathbf{l}_{d,i}=l,\mathbf{r}_{d,i}=r|\mathbf{w}_{d,i}=w,\mathbf{t}_{d,i}=t,\mathbf{z}_{-(d,i)},\mathbf{l}_{-(d,i)},\mathbf{r}_{-(d,i)},\alpha,\beta,\gamma,\eta)\propto \frac{n_{t,k,l,z,w}^{-(d,i)}+\beta_{l,r}^{w}}{n_{t,k,l,r}^{-(d,i)}+\sum_{j=1}^{V}*\beta_{l,r}^{j}}\frac{n_{d,t,k,l,z}^{-(d,i)}+\eta}{n_{d,t,k,l}^{-(d,i)}+R*\eta}\frac{n_{d,t,k,l}^{-(d,i)}+\gamma}{n_{d,t,k}^{-(d,i)}+L*\gamma}\frac{n_{d,k}^{-(d,i)}+\alpha}{n_{d}^{-(d,i)}+K*\alpha} \quad (2)$$

где $n_{t,k,l,r,w}$ означает количество раз, когда слову *w* с оценочным тэгом *t* присвоены тема *k*, тональная метка l и проблемная метка *r* в коллекции документов, $n_{d,t,k,l,r,w}$ означает число раз, когда слову *w* с оценкой *t* присвоены тема *k*, тональная метка l и проблемная метка *r* в документе *d*, $\beta_{l,r}^{w}$ обозначает априорное распределение Дирихле на *Ф* для слова *w* с тональной меткой l и проблемной меткой *r*.

5. Эксперименты и результаты

Для наших экспериментов мы использовали отзывы об автомобилях на русском языке, опубликованные в рамках дорожки анализа тональности соревнования SentiRuEval-2015 [16], и отзывы пользователей на английском языке компании Amazon[1] в четырех различных предметных областях. В качестве тестового корпуса использовались размеченные предложения пользователей из работ [3, 17]. Тестовый корпус содержит бинарную классификацию: предложения без упоминания проблем (no-problem класс) и с проблемными высказываниями (problem класс). Морфологическая обработка текста осуществлялась с помощью библиотеки NLTK[2] для английского языка и Mystem[3] для русского языка: на этапе предварительной обработки текстов была выполнена лемматизация и стемминг всех слов, удалены стоп-слова. Дополнительно из отзывов на русском языке были удалены слова, встречающиеся в корпусе менее двух раз. К словам рядом с отрицанием поставлен префикс *neg*. Таблица 2 содержит статистику по обучающей и тестовой коллекции.

Табл. 2. Статистика обучающей и тестовой коллекций.

Предметная область отзывов	**Количество отзывов с оценкой *r***						**Тестовый корпус**	
	r=1	*r=2*	*r=3*	*r=4*	*r=5*	размер словаря V	# problem	# no-problem
Электроника	820	598	922	2211	5450	35610	498	222
Детские товары	327	195	258	570	1498	9452	780	363
Инструменты для дома	873	503	965	19115	5745	23706	611	239
Машины (анг.)	399	239	361	1135	3760	13314	827	171
Машины (рус.)	40	150	634	3041	4061	22958	774	2824

Для сравнения результатов классификации были выбраны популярные вероятностные модели: модель тональность-тема (joint sentiment-topic model, JST) [10], модель тема-тональность (Reverse-JST) [10], модель аспект-тональность (aspect and sentiment unification model, ASUM) [11], модель User-aware Sentiment Topic Models (USTM) [12]. Эти модели объединяют тематическую и тональную переменные, USTM так же включает в распределения метаданные профайлов пользователей. Для обучения USTM в качестве метаданных используется информация о месте жительства пользователей из их личных профилей Amazon. Чтобы избежать

[1] Корпус отзывов доступен по ссылке https://snap.stanford.edu/data/web-Amazon.html.
[2] http://www.nltk.org/
[3] https://tech.yandex.ru/mystem/

разреженности тематик в USTM, текстовая коллекция собрана с учетом двадцати пяти наиболее часто встречающихся местоположений, указанных в профилях пользователей, для каждой предметной области. Поскольку корпус об автомобилях на русском языке не содержит информацию о пользователях, модель USTM не применима.

В качестве тонального лексикона (SL) используется словарь MPQA[4], часто использующийся в работах по анализу мнений для английского языка; для русского языка используется словарь позитивных и негативных слов, расширенный синонимами и родственными словами из Викисловаря[5] и описанный в статье [18]. В качестве словаря проблемных индикаторов (PL) используются слова, описанные в [3, 17]. На основе выбранных словарей задаются асимметричные априорные распределения Дирихле с гипепараметром β. Начальные значения гиперпараметров β_w для всех слов равны 0.1. Затем значения гиперпараметров β_{lw} для эмоционально-окрашенных слов определяются следующим, характерным для современных работ, образом: если существует вхождение слова в лексикон SL с позитивной пометкой, то $\beta_{*w} = (1{,}0.1, 0.01)$ (1 для позитивных меток (pos), 0.1 для нейтральных (neu), 0.01 для негативных (neg)); для слов с негативной пометкой $\beta_{*w} = (0.01{,}0.1{,}1)$. Аналогично определяются гиперпараметры β_{rw} для проблемных индикаторов на английском языке: если существует вхождение слова в словарь PL с проблемной пометкой, то $\beta_{*w} = (1, 0.01)$ (1 для проблемных меток (pr), 0.01 для меток, указывающие на отсутствие проблемы (no-pr)); проблемный индикатор с префиксом отрицания получает значения β_{*w}= $(0.01, 1)$. Для русского языка гиперпараметры β_{rw} задаются следующим образом: если существует вхождение слова в словарь PL с проблемной пометкой, то $\beta_{*w} = (0.001, 0.0001)$, для проблемного индикатора с префиксом отрицания $\beta_{*w} = (0.00001, 0.001)$. Гиперпараметр β_{lrw} определяется как сумма β_{lw} и β_{rw} ($l \in \{\text{neu}, \text{pos}, \text{neg}\}, r \in \{\text{pr}, \text{no} - \text{pr}\}$).

В качестве критерия качества построенных моделей используется перплексия контрольных данных (perplexity) [19]. 90% отзывов использовано в качестве обучающей выборки для вероятностных моделей, 10% отзывов использованы для тестирования. Для всех моделей при решении задачи статистического оценивания использовалось сэмплирование Гиббса, число итераций равно 1000. Все эксперименты проводились при следующих параметрах: α = 50/K, β=0.01, γ = 0.01 ∗ AvgLen/S, η = 0.01 ∗ AvgLen/R, где AvgLen обозначает среднее количество слов в отзыве, R=2, S=3, K=5. Для USTPM число различных пользовательских метаданных о географическом местоположении пользователя (T) равно 25. Результаты экспериментов представлены в таблице 3. Модель TSPM наименьшие значения перплексии среди моделей JST, Reverse-JST и USTM в коллекциях о детских товарах и инструментах, где

[4] http://mpqa.cs.pitt.edu/

[5] https://ru.wiktionary.org/wiki

любое слово документа зависит от скрытой тональной переменной. Предложенная модель RTSPM показывает наименьшие значения перплексии среди всех тематических моделей, что характеризует лучшую способность RTSPM предсказывать появление слов w в документах коллекции в зависимости от оценочного тэга, темы, тонального и проблемного контекста.

Табл. 3. Перплексия вероятностных моделей.

Метод	**Коллекции отзывов пользователей о продуктах разных категорий**				
	Электроника	Инструменты	Детские товары	Машины (анг.)	Машины (рус.)
JST+SL	6139.432	2934.682	1899.184	4114.005	4293.087
Reverse-JST+SL	5833.777	3345.068	2322.585	4405.159	4691.698
ASUM+SL	2544.348	2321.319	1327.732	1837.371	2047.021
USTM+SL	5543.127	4963.680	2525.952	3434.839	-
JST+PL	4191.591	2768.168	1878.961	3934.514	4203.726
Reverse-JST+PL	4643.836	2913.586	1902.013	4661.959	4470.233
ASUM+PL	2346.913	2580.232	1368.797	2294.031	2044.301
USTM+PL	5346.548	4603.271	1959.309	2919.317	-
TSPM	4006.628	2524.141	1572.788	4314.725	4296.813
RTSPM	2081.009	1433.861	759.591	1354.284	1824.547

Для каждой JST, Reverse-JST, ASUM, USTM мы использовали оба словаря SL и PL независимо: префикс ‘+SL’ свидетельствует, что модель учитывает только тональные метки слов и задает гиперпараметры β_{lw} (S=3) на основе словаря SL; префикс ‘+PL’ указывает на то, что модель учитывает только проблемные метки слов и задает гиперпараметры β_{rw} (R=2) на основе словаря PL. Для моделей, учитывающих только тональные метки слов (с префиксом ‘+SL’), используется следующее предположение: высказывание считается проблемным, если вероятность негативного класса $p(\mathrm{l}_{neg}|d)$ выше, чем вероятность позитивного и нейтрального классов: $p(\mathrm{l}_{pos}|d)$ и $p(\mathrm{l}_{neu}|d)$; аналогично высказывание не содержит проблем с продуктами, если $p(\mathrm{l}_{pos}|d)$ выше $p(\mathrm{l}_{neg}|d)$ и $p(\mathrm{l}_{neu}|d)$. Вероятности проблемного и тонального классов вычисляются на основе мультиномиального распределения в пространстве слов $\Phi_{z,l,r}$ по схожей формуле, описанной в [12].

Результаты классификации представлены в табл. 4 и табл. 5, в качестве критериев использовались стандартные метрики качества систем анализа текстов: аккуратность (accuracy), точность (precision) и полнота (recall) и F-мера (F1-measure). Поскольку мета-данные об авторе отзывов отсутствуют для русского языка, результаты USTM для отзывов для русского языка не описаны. Модели JST+SL, Reverse-JST+SL, ASUM+SL, USTM+SL показали наименьшие значения F-меры и аккуратности классификации по сравнению с JST+PL, Reverse-JST+PL, ASUM+PL, USTM+PL, что опровергает взаимно-однозначное соответствие негативного класса и класса проблемных высказываний. Наилучшие результаты F-меры достигают предложенные

модели TSPM и RTSPM на корпусе отзывов на английском языке, что показывает эффективность порождения слова в документах в зависимости от некоторой скрытой темы, тональной и проблемной информации. TSPM и RTSPM достигают сравнимые с другими вероятностными моделями результаты на корпусе отзывов на русском языке, что может быть связано с размером словарей позитивных и негативных слов (словарь SL для английского содержит 7629 слов, словарь SL для русского языка содержит около 3800 слов) или с необходимостью более сложной линейной комбинации гиперпараметров (полученной с помощью EM алгоритма (expectation-maximization) или линейного классификатора).

Табл. 4. Результаты классификации предложений отзывов пользователей о детских товарах и инструментах для дома.

Метод	**Инструменты для дома**				**Детские товары**			
	A	P	R	F1	A	P	R	F1
JST+SL	.442	.718	.368	.487	.569	.788	.481	.597
Reverse-JST+SL	.526	.747	.513	.609	.508	**.797**	.348	.485
ASUM+SL	.481	.741	.427	.542	.581	.794	.499	.613
USTM+SL	.536	.766	.511	.612	.500	.797	.328	.465
JST+PL	.587	.728	.677	.702	.497	.665	.490	.564
Reverse-JST+PL	.647	**.763**	.738	.750	.533	.665	.598	.630
ASUM+PL	.572	.731	.641	.684	**.621**	.742	.658	.698
USTM+PL	.514	.678	.617	.646	.480	.616	.574	.595
TSPM	**.664**	.724	**.857**	**.786**	.577	.698	635	.665
RTSPM	.622	.714	.792	.751	.618	.691	**.765**	**.726**

Табл. 5. Результаты классификации предложений отзывов пользователей об электронике и машинах.

Метод	**Электроника**				**Машины (анг)**				**Машины (рус.)**			
	A	P	R	F1	A	P	R	F1	A	P	R	F1
JST+SL	.408	.71	.41	.523	.244	**.92**	.10	.176	.614	.27	.47	.345
Reverse-JST+SL	.457	.70	.38	.494	.294	.89	.17	.285	.617	**.28**	.49	.354
ASUM+SL	.523	.71	.53	.606	.457	.85	.42	.563	.666	.30	.42	.352
USTM+SL	.564	.72	.61	.657	.373	.88	.28	.429	-	-	-	-
JST+PL	.689	.71	.93	.804	.671	.84	.74	.789	**.547**	.25	.55	.343
Reverse-JST+PL	.689	.72	.91	.802	.678	.83	.77	.798	.546	.26	.62	.368
ASUM+PL	.588	.74	.63	.679	.604	.84	.65	.729	.498	.23	.58	.331
USTM+PL	.575	.67	.77	.715	.529	.81	.57	.667	-	-	-	-
TSPM	**.693**	.71	.95	**.811**	.699	.83	**.80**	**.814**	.517	.26	.67	**.376**
RTSPM	.654	.71	.83	.769	**.701**	.83	**.80**	**.815**	.380	.23	**.78**	.351

В таблице 6 приведены примеры тем с различными проблемными метками, полученные с помощью моделей JST и TSPM. Слова из словарей PL и SL

выделены курсивом. По каждой теме представлены наиболее вероятностные слова. Согласно тональным меткам, TSPM по сравнению с JST+PL различает темы, где возможные неполадки или дискомфорт в использовании автомобилей связаны с внешними факторами (*зима, покупка, впечатление*) в отличие от тем, где существующие проблемные фразы связаны с составными элементами или функциями автомобилей (*крыло, ездить, ремонт двигателя*).

Табл. 6. Примеры тематических слов из отзывов о машинах на русском языке.

JST+PL		**TSPM**					
no-pr.	**probl.**	**negative**		**positive**		**neutral**	
		no-pr.	**probl.**	**no-pr.**	**probl.**	**no-pr.**	**probl.**
автомобиль	машина	зима	кузов	автомобиль	Машина	машина	масло
хороший	купить	машина	бампер	**хороший**	ездить	купить	поменять
качество	новый	заводиться	дверь	общий	расход	новый	замена
цена	деньги	мороз	краска	впечатление	менять	решать	двигатель
класс	машинка	печка	порог	авто	трасса	месяц	менять
модель	продавать	лето	крыло	салон	город	продавать	**проблема**
кузов	сразу	**проблема**	покрытие	двигатель	купить	деньги	приходится
надежный	решать	быстро	удар	расход	литр	становиться	задний
дизайн	тысяча	ездить	коррозия	**отличный**	масло	покупка	**ремонт**

6. *Заключение*

Исследование в рамках статьи рассматривает задачу анализа мнений пользователей о продуктах на русском и английском языках. Целью исследования является резюмирование слов в отзывах пользователей по тематическим отзывам, используя взаимосвязи между проблемными индикаторами, эмоционально-окрашенными (тональными) словами пользователя и категориями аспектных терминов продуктов при моделировании слов в отзывах. Для достижения целей исследования в статье представлены тематические модели на основе модели латентного размещения Дирихле: (i) модель *тема-тональность-проблема (topic-sentiment-problem model)* и (ii) модель *оценка пользователя-тема-тональность-проблема* (rating-aware topic-sentiment-problem model). Предложенные модели объединяют знания на основе словарей тональных слов и проблемных индикаторов с помощью асимметричных гиперпараметров для всех слов документов. В статье оценка качества предложенных методов анализируются в сравнении с

результатами популярных модификаций латентного размещения Дирихле для задач анализа мнений. Предложенные модели достигают наилучшие результаты F-меры и сравнимые значения перплексии в сравнении с другими вероятностными моделями. В последующих исследованиях планируется улучшить предложенные модели за счет изменения линейной комбинации гиперпараметров с помощью EM алгоритма или методов машинного обучения.

Список литературы

[1]. Сабирова И.М. Качество–ключевой фактор обеспечения конкурентности продуктов и услуг в условиях рыночной экономики. Автоматизация и управление в технических системах, №1, 2015, С. 181-190.

[2]. Gupta N. K. Extracting descriptions of problems with product and services from twitter data. Proceedings of the 3rd Workshop on Social Web Search and Mining (SWSM2011). Beijing, China, 2011.

[3]. Solovyev V., Ivanov V. Dictionary-Based Problem Phrase Extraction from User Reviews. Text, Speech and Dialogue, Springer International Publishing, 2014, P. 225-232.

[4]. Moghaddam S. Beyond Sentiment Analysis: Mining Defects and Improvements from Customer Feedback. Advances in Information Retrieval, Springer International Publishing, 2015, P. 400-410.

[5]. Tutubalina E. Target-Based Topic Model for Problem Phrase Extraction. Advances in Information Retrieval, 2015, P. 271-277.

[6]. Blei D. M., Ng A. Y., Jordan M. I. Latent dirichlet allocation. The Journal of machine Learning research., T. 3, 2003, P. 993-1022.

[7]. Liu B. Sentiment analysis and opinion mining. Synthesis Lectures on Human Language Technologies, T. 5. , 2012, P. 1-167.

[8]. Martínez-Cámara E, Martín-Valdivia M. T., Urena-López L. A., Montejo-Ráe, A. R. Sentiment analysis in twitter. Natural Language Engineering, T. 20(1), 2014, P. 1-28.

[9]. Moghaddam S., Ester M. On the design of LDA models for aspect-based opinion mining. Proceedings of the 21st ACM international conference on Information and knowledge management. – ACM, 2012., P. 803-812.

[10]. Lin C., He Yu., Everson R., Ruger S. Weakly supervised joint sentiment-topic detection from text. Knowledge and Data Engineering, IEEE Transactions on, T. 24(6), 2012, P. 1134-1145.

[11]. Jo Y., Oh A. H. Aspect and sentiment unification model for online review analysis. Proceedings of the fourth ACM international conference on Web search and data mining, ACM, 2011, P. 815-824.

[12]. Z. Yang, A. Kotov, A. Mohan S. Lu. Parametric and Non-parametric User-aware Sentiment Topic Models. Proceedings of the 38th ACM SIGIR, 2015.

[13]. Heinrich G. Parameter estimation for text analysis. Technical report, 2005.

[14]. Minka T., Lafferty J. Expectation-propagation for the generative aspect model. Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence. – Morgan Kaufmann Publishers Inc., 2002., P. 352-359.

[15]. Griffiths T. L., Steyvers M. Finding scientific topics. Proceedings of the National Academy of Sciences, T. 101 (1), 2004, P. 5228-5235.

[16]. Loukachevitch N., Blinov P. , Kotelnikov E., Rubtsova Y., Ivanov V., Tutubalina E. SentiRuEval: testing object-oriented sentiment analysis systems in Russian. Proceedings of International Conference Dialog-2015, Moscow, Russia, 2015.

[17]. Тутубалина Е. В. Извлечение проблемных высказываний, связанных с неисправностями и нарушением функциональности продуктов, на основании отзывов пользователей. «Вестник КГТУ им. А.Н.Туполева», Т. 3, 2015.
[18]. Ivanov V., Tutubalina E., Mingazov N., Alimova I. Extracting aspects, sentiment and categories of aspects in user reviews about restaurants and cars. Proceedings of International Conference "Dialog-2015", Moscow, Russia, 2015.
[19]. Воронцов К. В., Потапенко А. А. Регуляризация, робастность и разреженность вероятностных тематических моделей. Компьютерные исследования и моделирование, Т. 4 (4), 2012, С. 693-706.

Sentiment-Based Topic Model for Mining Usability Issues and Failures with User Products

E.V. Tutubalina <ElVTutubalina@kpfu.ru>
Kazan (Volga Region) Federal University,
18 Kremlyovskaya Str., Kazan, 420008, Russian Federation

Abstract. This paper describes an approach to problem phrase extraction from texts that contain user experience with products. During the last decades, consumer products have grown in complexity, and consumer dissatisfaction is increasingly caused by usability problems, in addition to problems with technical failures. Moreover, user reviews from online resources, that describe actual difficulties with products experienced by users, affect on other people's purpose decisions. In this paper, we present two probabilistic graphical models which aim to extract problems with products. We modify Latent Dirichlet Allocation (LDA) to incorporate information about problem phrases with words' sentiment polarities (negative, neutral or positive). The proposed models learn a distribution over words, associated with topics, both sentiment and problem labels. Topic models were evaluated on reviews of different domains collected from online consumer review platforms. The algorithms achieve a better performance in comparison to several state-of-the-art models in terms of the likelihood of a held-out test and in terms of an accuracy of classification results. Qualitative analysis of the topics discovered using the proposed models indicates the utility of considering sentiment information in users' critical feedback. Our contribution is that incorporating sentiment and problem information about words with reviews' topics by the model's asymmetric priors gives an improvement for problem phrase extraction from user reviews published in English and Russian.

Keywords: opinion mining, problem phrases, user reviews, mining defects, topic modeling, LDA, problem phrase extraction.

DOI: 10.15514/ISPRAS-2015-27(4)-6

For citation: Tutubalina E.V. Sentiment-Based Topic Model for Mining Usability Issues and Failures with User Products. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 111-128 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-6.

References

[1]. Sabirova I.M. [Quality-key factor of ensuring competition of products and services in the conditions of market economy]. Avtomatizaciya i upravlenie v tehnicheskih sistemah [Automation and management in technical systems], vol. 1, 2015, pp. 181-190. (In Russian)

[2]. Gupta N. K. Extracting descriptions of problems with product and services from twitter data. Proceedings of the 3rd Workshop on Social Web Search and Mining. Beijing, China, 2011.

[3]. Solovyev V., Ivanov V. Dictionary-Based Problem Phrase Extraction from User Reviews. Text, Speech and Dialogue, Springer International Publishing, 2014, vol. 225-232.

[4]. Moghaddam S. Beyond Sentiment Analysis: Mining Defects and Improvements from Customer Feedback. Advances in Information Retrieval, Springer International Publishing, 2015, PP. 400-410.

[5]. Tutubalina E. Target-Based Topic Model for Problem Phrase Extraction. Advances in Information Retrieval, 2015, pp. 271-277.

[6]. Blei D. M., Ng A. Y., Jordan M. I. Latent dirichlet allocation. The Journal of machine Learning research., vol. 3, 2003, pp. 993-1022.

[7]. Liu B. Sentiment analysis and opinion mining. Synthesis Lectures on Human Language Technologies, T. 5. , 2012, pp. 1-167.

[8]. Martínez-Cámara E, Martín-Valdivia M. T., Urena-López L. A., Montejo-Ráe, A. R. Sentiment analysis in twitter. Natural Language Engineering, T. 20(1), 2014, PP. 1-28.

[9]. Moghaddam S., Ester M. On the design of LDA models for aspect-based opinion mining. Proceedings of the 21st ACM international conference on Information and knowledge management. – ACM, 2012., pp. 803-812.

[10]. Lin C., He Yu., Everson R., Ruger S. Weakly supervised joint sentiment-topic detection from text. Knowledge and Data Engineering, IEEE Transactions on, vol. 24(6), 2012, pp. 1134-1145.

[11]. Jo Y., Oh A. H. Aspect and sentiment unification model for online review analysis. Proceedings of the fourth ACM international conference on Web search and data mining, ACM, 2011, pp. 815-824.

[12]. Z. Yang, A. Kotov, A. Mohan S. Lu. Parametric and Non-parametric User-aware Sentiment Topic Models. Proceedings of the 38th ACM SIGIR, 2015.

[13]. Heinrich G. Parameter estimation for text analysis. Technical report, 2005.

[14]. Minka T., Lafferty J. Expectation-propagation for the generative aspect model. Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence. – Morgan Kaufmann Publishers Inc., 2002., pp. 352-359.

[15]. Griffiths T. L., Steyvers M. Finding scientific topics. Proceedings of the National Academy of Sciences, vol. 101 (1), 2004, pp. 5228-5235.

[16]. Loukachevitch N., Blinov P., Kotelnikov E., Rubtsova Y., Ivanov V., Tutubalina E. SentiRuEval: testing object-oriented sentiment analysis systems in Russian. Proceedings of International Conference Dialog-2015, Moscow, Russia, 2015.

[17]. Tutubalina E.V. [Extracting problem phrases about product defects and malfunctions in user reviews about cars]. Vestnik KGTU im. A.N.Tupoleva [Proceeding of KGTU im. A.N.Tupoleva], vol. 3, 2015. (in Russian)

[18]. Ivanov V., Tutubalina E., Mingazov N., Alimova I. Extracting aspects, sentiment and categories of aspects in user reviews about restaurants and cars. Proceedings of International Conference "Dialog-2015", Moscow, Russia, 2015.

[19]. Vorontsov K.V., Potapenko A.A. [Regularization, robustness and sparsity of probabilistic topic models]. Komp'yuternyye issledovaniya i modelirovaniye [Computer research and modeling], vol. 4 (4), 2012, pp. 693-706. (in Russian).

Методы построения социо-демографических профилей пользователей сети Интернет

[1, 2]А.Г. Гомзин <gomzin@ispras.ru>
[1, 2, 3]С.Д. Кузнецов <kuzloc@ispras.ru>
[1] Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2] Московский государственный университет имени М.В. Ломоносова, 119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК
[3]Московский физико-технический институт (государственный университет), 141700, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. Работа посвящена методам построения социально-демографического профиля пользователей Интернета. Примерами демографических атрибутов являются пол, возраст, политические и религиозные взгляды, район проживания, состояние отношений с другими людьми. Эта работа представляет собой обзор методов, которые обнаруживают демографические атрибуты из профиля пользователя и сообщений. Большинство известных работ посвящены выявлению пола. Возраст, политические взгляды и области также интересуют исследователей.

Самыми популярными источниками данных для извлечения демографических атрибутов являются социальные сети, такие как Facebook, Twitter, Youtube. Большинство решений основано на машинном обучении с учителем. Машинное обучение позволяет найти целевые значения (демографические атрибуты) в зависимости от входных данных и использовать их, чтобы предсказать значение целевого атрибута для новых данных. в работе анализируются следующие шаги решения задачи: сбор данных, извлечение признаков, отбор информативных признаков, методы обучения классификаторов, оценка качества.

Исследования используют различные виды данных, чтобы предсказать демографические атрибуты. Самым популярным источником данных является текст. Последовательности слов (n-граммы), части речи, смайлики, особенности относящиеся к конкретным ресурсам (например, @ и # в Twitter) извлекаются и используются в качестве входных данных для алгоритмов машинного обучения. Социальные графы также используются в качестве исходных данных. Сообщества пользователей, которые автоматически извлекаются из социального графа пользователем в качестве признаков для прогнозирования атрибутов. Текстовые данные дает много возможностей. Алгоритмы выбора признаков необходимы для снижения признакового пространства.

В статье исследуются функции выбора, классификации и регрессии алгоритмы, показатели оценки.

Ключевые слова: демографические атрибуты; демографические характеристики; социальные сети; обработка текстов на естественном языке; машинное обучение

DOI: 10.15514/ISPRAS-2015-27(4)-7

Для цитирования: Гомзин А.Г., Кузнецов С.Д. Методы построения социо-демографических профилей пользователей сети Интернет. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 129-144. DOI: 10.15514/ISPRAS-2015-27(4)-7.

1. Введение

Многие ресурсы в сети Интернет позволяют своим пользователям принимать активное участие в создании контента. К таким ресурсам относятся блоги, форумы, социальные сети. Кроме того, многие интернет-магазины, новостные сайты и другие подобные сервисы позволяют пользователям оставлять комментарии, отзывы. Как правило, помимо сообщений, комментариев, отзывов, оценок пользователь оставляет на ресурсе некоторую информацию о себе. Эта информация включает в себя имя, пол, возраст, интересы, контактные и другие данные. Такая информация о пользователе, как правило, помещается на отдельную страницу ресурса. Набор таких атрибутов называется профилем пользователя.

Контент, генерируемый пользователями, отражает его интересы, взгляды. Так, например, лексика, используемая в социальных сетях подростками и взрослыми людьми, различается. В качестве контента, генерируемого пользователями, можно рассматривать тексты, изображения, аудио- и видеоконтент. В данном обзоре рассматриваются только работы, посвященные анализу текстового контента пользователей.

В статье рассматривается задача составления социо-демографических профилей пользователей сети Интернет. Далеко не все пользователи полностью заполняют свой профиль. Кроме того, в некоторых случаях пользователи преднамеренно указывают неверные данные. В связи с этим возникает задача предсказания неизвестных социо-демографических атрибутов, таких как пол, возраст, политические предпочтения, по имеющейся информации о пользователе. Обычно анализируется только находящийся в публичном доступе контент пользователя, т.е. его сообщения, комментарии, отзывы.

Методы автоматического определения демографических атрибутов пользователей могут использоваться для исследования определенных групп пользователей, даже если не все пользователи указывают значения атрибутов.

Полученные с помощью таких методов значения атрибутов могут применяться в рекомендательных системах [1], для таргетированной рекламы [2], а также в других приложениях.

Абсолютное большинство работ по определению пола, возраста и других атрибутов основано на методах машинного обучения. Решение задачи разбивается на несколько этапов:

1. сбор данных для построения модели
2. построение (обучение) модели
3. классификация с использованием полученной модели и оценка качества модели

В первом разделе рассматриваются исходные данные и особенности их сбора. Второй раздел описывает решения, где не используется машинное обучение. Третий раздел посвящен решению задачи с использованием методов машинного обучения. Затем следуют выводы и заключение.

2. Данные

Наибольший интерес для исследователей представляют активно развивающиеся социальные сети, такие как Facebook, Twitter и другие.

Facebook – самая крупная по количеству зарегистрированных пользователей социальная сеть [3]. В ней зарегистрировано более 1,2 млн пользователей. В профилях пользователей можно встретить различные демографические атрибуты: пол, возраст, семейное положение, политические и религиозные взгляды и т.д. Классификаторы, полученные с использованием контента пользователей и их профилей, позволяют с высокой точностью предсказывать значения атрибутов, которые не указаны в профиле других пользователей.

Другой ресурс, не менее популярный у исследователей демографических атрибутов, – социальная сеть Twitter. Это сервис микроблоггинга, в котором длина каждого сообщения не превышает 140 символов. Такие тексты имеют свои особенности, которые описаны в работе [4]. На данном ресурсе в профилях пользователей отсутствуют демографические атрибуты, что усложняет этап сбора обучающей выборки. Об этом подробнее будет написано в разделе 2.1.

Кроме данных ресурсов, в некоторых исследованиях анализируются комментарии на Youtube [5], новости и электронные письма [6].

В нашем обзоре представлены работы, в которых решаются задачи определения следующих атрибутов: пол, возраст, политические взгляды, регион проживания.

2.1 Сбор данных

Первым этапом решения задачи определения демографических атрибутов является сбор данных. Данные содержат текстовый контент пользователей, а также истинные значения атрибутов. Значения атрибутов используются алгоритмами машинного обучения с учителем и оценки качества.

Социальная сеть представляется в виде графа, в котором вершины соответствуют пользователям, а ребра – наличию социальной связи между пользователями (отношения дружбы, подписки и т.д.). Возникает задача обхода вершин графа с целью получения репрезентативной выборки. Такой процесс обхода называется сэмплингом. Исследования [7, 8] показали, что наиболее репрезентативные выборки получаются при использовании алгоритмов сэмплирования «Лесного пожара» (Forest Fire) и Метрополиса-Гастингса (Metropolis–Hasting).

При решении задач определения демографических атрибутов в обучающую выборку попадают только пользователи с явно указанными атрибутами. Такие данные можно собрать, например, с использованием сервиса поиска друзей в социальных сетях. Например, в социальной сети Вконтакте сервис поиска пользователей[1] позволяет явно указать значения атрибутов интересующих пользователей. Встречаются также работы, в которых значения целевых атрибутов определяются экспертами [9, 10, 11, 12].

Не всегда на анализируемом ресурсе профиль пользователей содержит нужный атрибут. Примером такого ресурса со скудными профилями является сервис микроблогинга Twitter. В профиле Twitter не указывается пол, возраст, семейное положение и другие атрибуты. Найти значение нужного атрибута можно, например, если знать где находится профиль этого же человека на другом ресурсе. В профиле Twitter имеется поле URL, в котором пользователи часто указывают гиперссылку на свой профиль в другой социальной сети. Перейдя по гиперссылке, можно найти значения целевых атрибутов. Такой подход используется в работах [13, 14].

Даже если данных с атрибутами достаточно для построения решений и реализации методов, могут возникать технические проблемы, связанные с ограничениями на ресурсах. Например, API Twitter позволяет выдавать не более 300 запросов на скачивание страницы сообщений пользователя в течение 15 минут для одного приложения[2]. Кроме того, на многих ресурсах имеются неявные ограничения. Например, при слишком частом обращении к сервису, он может заблокировать запросы и не возвращать данные. Частоту, с которой нужно делать запросы, не получая блокировку, можно определить только эмпирически.

3. Методы, не использующие машинное обучение

Как правило, задачи определения демографических атрибутов пользователей решаются с использованием методов машинного обучения. Но в некоторых задачах высокая точность достигается и при использовании более простых решений.

[1] https://vk.com/people

[2] https://dev.twitter.com/rest/reference/get/statuses/user_timeline

Например, пол человека с высокой точностью определяется по его имени. В работе [15] использовались различные словари имен. Такие методы предполагают, что пользователи указывают свое настоящее имя.

Для определения остальных атрибутов требуются более сложные методы. В большинстве исследований, посвященных определению демографических атрибутов пользователей, используется машинное обучение с учителем.

4. Машинное обучение

Машинное обучение позволяет найти зависимость целевых значений от исходных данных и использовать ее для предсказания значения целевого атрибута для новых данных. В нашем случае целевые данные – это демографические атрибуты, а исходные данные – информация о пользователе. Информация о пользователе варьируется от ресурса к ресурсу, но, как правило, включает в себя сообщения, автором которых он является, и профиль – набор атрибутов и их значений. В социальных сетях также доступна информация об отношениях между пользователями, а также отношениях между пользователями и объектами: так называемые социальные связи. В некоторых работах [5, 9] эти связи используются для определения демографических атрибутов.

Различаются машинное обучение без учителя и машинное обучение с учителем. При обучении без учителя алгоритм находит закономерности в исходных данных, с помощью которых разбивает эти данные на группы (кластеры). При обучении с учителем, помимо исходных данных, для алгоритма требуются значения целевых атрибутов. В процессе обучения строится модель, с помощью которой предсказываются целевые значения для новых исходных данных, в которых значения целевых атрибутов неизвестны. В задачах определения демографических атрибутов, как правило, имеется выборка, в которой значения атрибутов указаны явно. Соответственно, применяется обучение с учителем.

При использовании машинного обучения процесс решения задачи включает в себя четыре этапа:

1. извлечение признаков;
2. отбор признаков (опционально);
3. обучение модели;
4. оценка качества алгоритма.

На рис. 1 изображена схема обучения с учителем. В качестве целевого атрибута рассматривается пол пользователя.

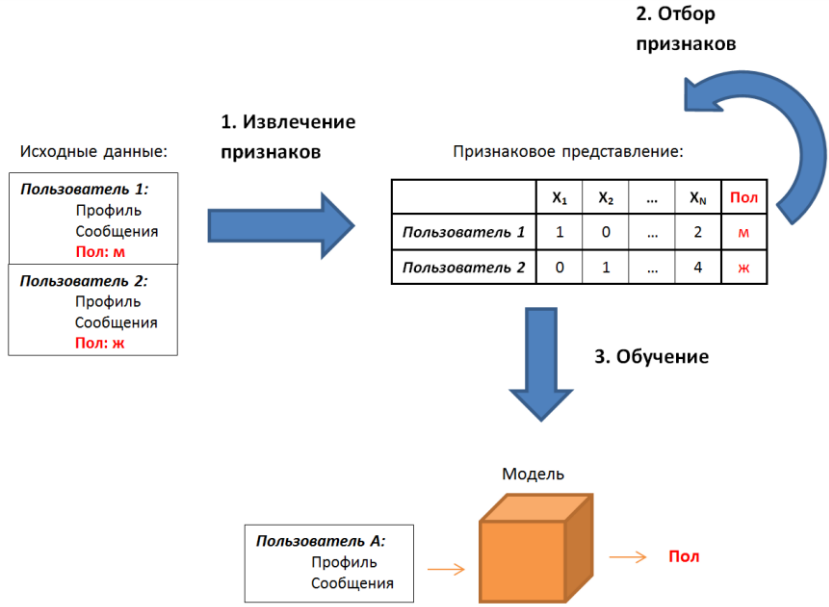

Рис. 1. Машинное обучение с учителем

4.1 Извлечение признаков

Данный подраздел посвящен признакам, используемым при обучении классификаторов пользователей по демографическим атрибутам. Сначала опишем признаки, а затем исследования, в которых используются эти признаки.

Исходные данные в задачах определения атрибутов представляют собой профиль пользователя и набор текстовых сообщений, автором которого он является. Но алгоритмы машинного обучения работают с признаковыми представлениями объектов (рис. 1), где каждый объект представляет собой набор векторов в пространстве признаков. Если применяются методы машинного обучения с учителем, значение целевого атрибута также входит в объект. Соответствующий этап машинного обучения называется извлечением признаков (рис. 1).

Для начала необходимо определить, какие признаки можно выделить из текстов и профилей. Текст состоит из слов. В качестве признаков используются слова и последовательности подряд идущих слов. Такие последовательности называются n-граммами. Здесь n – длина последовательности. В качестве значений признаков можно использовать бинарное значение: 1, если слово встретилось в тексте, 0 – иначе. Кроме того,

значением признака может быть частота встречаемости слова в текст, мера TF-IDF и т.д.

Часто в определении пола, возраста и других атрибутов помогают части речи в последовательности слов. Части речи последовательностей подряд идущих слов могут рассматриваются как признаки (POS n-граммы).

С другой стороны, текст можно рассматривать как последовательность символов. Соответственно, в качестве признаков могут использоваться символьные n-граммы, т.е. последовательности подряд идущих символов.

Существует также подход, при котором в качестве признаков рассматриваются последовательности фонем – единиц звука речи. В этом случае ключевую роль играет не написание слов, а их произношение.

В некоторых работах используются такие признаки, как сокращения, эмотиконы, признаки, специфичные для определенных ресурсов (например, @упоминания и #хэштэги в Твиттере).

Еще одна группа признаков – статистические. Числовые значения таких признаков – некоторые статистические значения, полученные из тестов. Примерами статистических признаков являются средняя длина одного сообщения пользователя, частота эмотиконов, частота знаков препинания в сообщениях и т.д.

Помимо текстов сообщений пользователя, анализируется его профиль. Значения атрибутов профиля могут использоваться как признаки. В некоторых случаях из значений атрибутов извлекаются описанные выше признаки (например, n-граммы). Например, для определения пола пользователя полезным бывает имя. Встречаются также работы, где используется цвет страницы пользователя, заданный им в личных настройках.

В социальных сетях, помимо текстов и профилей пользователей, существуют социальные связи. Социальный граф, в котором вершины соответствуют пользователям, а ребра – наличию отношений между пользователями, также используется для предсказания атрибутов. Например, в таком графе с помощью специальных алгоритмов [16] могут выделяться сообщества. Набор сообществ, в которые входит пользователь, используется в качестве признаков.

Как правило, из исходных данных извлекаются комбинации различных типов признаков. Но важно помнить, что не всегда использование большего числа признаков улучшает качество получаемых классификаторов. Может возникнуть переобучение. Способы избежать переобучения описываются в следующем разделе.

Универсального рецепта по выбору признаков нет. Выбор признаков существенно зависит от задачи (т.е. какой атрибут определяется) и исходных данных. Далее описываются некоторые работы и признаки, используемые авторами этих работ.

В работе [5] определяется пол пользователей Youtube по комментариям и графу пользователи-видео, где ребро между пользователем и видео означает факт просмотра видео пользователем. В качестве признаков рассматриваются статистические признаки, такие как средняя длина комментария в символах/словах/предложениях, словесные n-граммы, возраст пользователя, распределение пола, полученное с помощью модели распространения атрибута «пол» в графе пользователи-видео.

Авторы работы [14] определяют пол пользователей Twitter. В этом исследовании извлекаются признаки из профиля, в том числе цвета фона/текста/ссылок, которые пользователь указал в настройках своей страницы. Также выделяются признаки из имен пользователей. При этом имя преобразовывается в последовательность фонем.

В работе [13] пол пользователей Twitter определяется по текстам их твитов. Используются символьные и словесные n-граммы. Для решения той же задачи авторы [12] использовали символьные n-граммы.

В работе [17] рассматривалась задача определения возраста пользователей, пишущих на голландском языке. Возраст пользователей разбивался на 2 или 4 интервала разными способами (до 16, после 16 лет; до 16, после 18 лет; до 16, после 25 лет; то же самое, но для каждого значения пола). В качестве признаков использовались символьные и словесные 1,2 и 3-граммы.

Помимо решений, в которых множество значений возраста пользователей разбивается на несколько интервалов, существуют методы, которые предсказывают числовое значение возраста [18]. В [18] методы тестировались на нескольких наборах исходных данных. При этом в качестве признаков использовались словесные юниграммы (1-граммы), юниграммы и биграммы (2-граммы) частей речи слов, статистические признаки.

Пол и возраст – атрибуты, наиболее популярные среди исследователей. Существуют также и работы, в которых рассматриваются другие атрибуты, такие как политические взгляды и регион проживания.

Авторы [9] определяют политические предпочтения пользователей социальной сети Twitter. Рассматриваются три класса: демократы, республиканцы, неявная политическая позиция. В качестве признаков используются словесные юниграммы, хэштеги, сообщества пользователей (полученные с помощью алгоритма, основанного на распространении меток в социальном графе пользователей).

В работе [10] решаются задачи определения пола, возраста (до 30, после 30), региона (юг и северо-восток США) и политических взглядов (республиканцы, демократы). Авторы рассматривали в качестве признаков юниграммы и биграммы слов, а также социолингвистические признаки – эмотиконы, аббревиатуры, повторяющиеся знаки препинания и др.

Мы затронули лишь некоторые исследования, посвященные определению демографических атрибутов. В большинстве работ в качестве признаков присутствуют n-граммы, извлеченные из текстов сообщений пользователей.

При использовании n-грамм пространство признаков велико, в связи с чем возникает задача отбора информативных признаков.

4.2 Отбор признаков, уменьшение размерности

В процессе извлечения признаков из текста каждый пользователь представляется большим количеством признаков. Если рассматривать все различные признаки, встречающиеся у всех пользователей, то их получается на порядки больше, чем самих пользователей. Например, в работе [13] обучающая выборка содержит 180000 пользователей, при этом в ней 15000000 различных признаков. При таких данных высока вероятность возникновения переобучения. При переобучении модели классификаторов получаются очень сложными, так как в них присутствуют все признаки. В этом случае классификатор правильно предсказывает результат для тех данных, на которых он обучился, и неправильно – для новых данных. Чтобы избавиться от переобучения, нужно уменьшить количество признаков.

Один из способов – избегать большого числа признаков на этапе выбора признаков. Например, в работе [14] рассматриваются последовательности фонем, извлеченных из имени пользователя. В данной работе используется до 16000 различных признаков при том, что набор данных содержит около 180000 пользователей.

В случае, когда анализируются тексты сообщений, нельзя обойтись без этапа отбора признаков. С помощью специальных методов, о которых будет сказано далее, выбираются наиболее информативные признаки для анализируемого набора данных и у каждого объекта оставляются только те признаки, которые считаются информативными.

Среди методов отбора признаков существуют такие, которые не рассматривают значения целевых атрибутов. Одним из простых примеров является фильтрация признаков по частоте. Для каждого признака вычисляется количество объектов, в которых данный признак присутствует. Выбираются признаки с наибольшим значением частоты, а редкие признаки не рассматриваются. Другой способ – оставлять признаки с высокой дисперсией. В этом случае удаляются признаки, значение которых несущественно варьируется у объектов.

В этих методах не учитываются значения целевых атрибутов у объектов обучающей выборки. Информативность признака должна оцениваться в контексте целевого атрибута. Например, значение признака «окончание имени пользователя» коррелирует с полом соответствующего пользователя: как правило, имена женщин заканчиваются на гласную букву, имена мужчин – на согласную.

В работе [12] используются несколько методов отбора признаков: Хи-квадрат (Chi-Square), прирост информации (Information Gain), отношение прироста информации (Information Gain Ratio), Relief, симметричная неопределенность (Symmetrical Uncertainty), Filtered Attribute Evaluation. Все эти методы

оценивают, насколько хорошо значения признаков разделяют выборку по классам.

Некоторые алгоритмы машинного обучения имеют встроенную возможность отбора признаков. Как правило, алгоритмы подбирают параметры модели, чтобы минимизировалась ошибка на обучающей выборке. Для уменьшения количества признаков в модели используется регуляризация. Суть ее заключается в том, что сложность модели (пропорциональная количеству признаков с ненулевыми весами) минимизируется одновременно с ошибкой на тренировочных данных. Примером такой регуляризации, уменьшающей размерность данных, является регуляризации LASSO. К функционалу ошибки прибавляется сумма модулей весов признаков. Таким образом, если признак используется в модели, функционал ошибки увеличивается на модуль его веса.

Рассмотренные способы уменьшают размерность исходных данных путем удаления неинформативных признаков. В этом случае в качестве признаков на вход обучающему алгоритму приходит некоторое подмножество исходных признаков. Существуют другие подходы уменьшения размерности данных, в которых признаковое пространство полностью меняется.

Для текстовых данных применимы методы тематического моделирования [19]. В результате применения таких методов каждый текст (сообщение пользователя) представляется в виде распределения над темами.

Исходные данные можно представить в виде матрицы (объекты × признаки). Существуют различные методы матричных разложений, где исходная матрица представляется в виде произведения двух других: $A=W\times H$. Здесь A – исходная матрица (размерности $n\times m$) признаковых представлений объектов (n объектов, m признаков) раскладывается на произведение двух других матриц: W (размерности $n\times t$) и H (размерности $t\times m$). Матрицу W можно рассматривать как новое признаковое представление объектов в пространстве t неявных признаков.

Отбору признаков как отдельной проблеме посвящены работы [20, 21].

4.3 Используемые алгоритмы машинного обучения

Множество значений исследуемых демографических атрибутов, как правило, состоит из нескольких элементов. Например, пол принимает два значения: мужской, женский. В некоторых случаях атрибут принимает числовое значение (например, возраст). Таким образом, исходная задача сводится к задаче классификации или регрессии.

Одним из самых простых алгоритмов классификации является Наивный байесовский классификатор. Он предполагает, что все признаки независимы. В его основе лежит теорема Байеса. Данный алгоритм используется в работах [12, 13, 14] для определения пола. Преимущество данного метода состоит в том, что он поддерживает онлайн-обучение, т.е. при добавлении в обучающую

выборку новых объектов модель классификатора не пересчитывается заново, а обновляется согласно новым данным.

При классификации на два класса часто используется линейный классификатор. Одним из популярных алгоритмов обучения линейного классификатора является метод опорных векторов (в англоязычной литературе также известный как SVM – Support Vector Machine). Основная идея метода – поиск разделяющей гиперплоскости с максимальным зазором до объектов классов. Метод опорных векторов использовался в работах [6, 9, 10, 13, 17, 18].

Метод опорных векторов не является онлайновым. При необходимости «дообучения» линейного классификатора нужно использовать другие алгоритмы, такие как Перцептрон и Balanced Winnow. Такие алгоритмы используются в работах [11, 12, 13].

В работах также встречаются решающие деревья [6, 14], логистическая регрессия [6].

4.4 Оценка качества

Оценка качества нужна, чтобы иметь представление о том, насколько хорошо работает полученный алгоритм. Для этого измеряются такие значения, как точность (accuracy), достоверность (precision), полнота (recall) и F-1 мера.

Точность определяется как доля объектов, по которым классификатор принял правильное решение:

$$Accuracy = \frac{P}{N}$$

Достоверность и полнота рассматриваются в пределах одного класса, который обозначается *положительным*. Сначала составляется таблица (табл. 1)

Таблица 1. Обозначение результатов классификатора по отношению к истинным значениями

		Истинное значение	
		положительное	*отрицательное*
Результат классификатора	*Положительный*	TP	FP
	Отрицательный	FN	TN

Тогда достоверность определяется как:

$$Precision = \frac{TP}{TP + FP}$$

Полнота:

$$Recall = \frac{TP}{TP + FN}$$

F-мера представляет собой гармоническое среднее между достоверностью и полнотой:

$$F = 2\frac{Precision \times Recall}{Precision + Recall}$$

Как правило, качество алгоритмов оценивается с использованием кросс-валидации. Набор размеченных данных разбивается на несколько частей. Затем для каждой части происходит обучение на оставшихся частях и проверка на выбранной части данных. Измерения значений точности или других параметров усредняются по всему набору данных.

Для задач регрессии используется, как правило, метрика MAE – средняя абсолютная ошибка:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Здесь y_i – предсказанное значение, $\hat{y}_i$ – истинное значение.

Результаты исследований показывают большой разброс в зависимости от данных, извлекаемых признаков и алгоритмов машинного обучения. Здесь приводятся интервалы значений точности для различных атрибутов:

Пол: 75-95 %;

Политические взгляды: 79-95%;

Возраст (MAE): 5-17.

5. Заключение

В данной статье рассматриваются методы построения социо-демографичского профиля пользователей Интернета. Большинство обозреваемых работ посвящено анализу сообщений социальных сетей. Особый интерес представляет социальная сеть Twitter, так как профили пользователей не содержат явно указанных демографических атрибутов. Кроме того, сообщения имеют свои стилистические особенности.

Много работ посвящено определению пола. Также встречаются статьи, посвященные определению возраста, политических взглядов, региона проживания.

Абсолютное большинство решений основано на использовании методов машинного обучения с учителем. В статье рассмотрен каждый этап решения: сбор данных, извлечение признаков, отбор информативных признаков, методы обучения классификаторов, оценка качества.

В заключение можно выделить направления дальнейших улучшений работы алгоритмов. Первое направление – определять все атрибуты вместе, учитывая зависимость между этими атрибутами. Второе направление – исследовать

возможность построения классификаторов, не зависящих от источника исходных данных.

Список литературы

[1]. Li Q., Kim B. M. Constructing user profiles for collaborative recommender system //Advanced Web Technologies and Applications. – Springer Berlin Heidelberg, 2004. – С. 100-110.

[2]. Bharat K., Lawrence S., Sahami M. Generating user information for use in targeted advertising : заяв. пат. 10/750,363 США. – 2003.

[3]. Список социальных сетей. [электронный ресурс] https://ru.wikipedia.org/wiki/Список_социальных_сетей

[4]. Коршунов А. и др. Определение демографических атрибутов пользователей микроблогов //Труды Института системного программирования РАН. – 2013. – Т. 25, стр. 179-194. DOI: 10.15514/ISPRAS-2013-25-10

[5]. Filippova K. User demographics and language in an implicit social network //Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. – Association for Computational Linguistics, 2012. – С. 1478-1488.

[6]. Cheng N., Chandramouli R., Subbalakshmi K. P. Author gender identification from text //Digital Investigation. – 2011. – Т. 8. – №. 1. – С. 78-88.

[7]. Leskovec J., Faloutsos C. Sampling from large graphs //Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. – ACM, 2006. – С. 631-636.

[8]. Gjoka M. et al. Walking in Facebook: A case study of unbiased sampling of OSNs //INFOCOM, 2010 Proceedings IEEE. – IEEE, 2010. – С. 1-9.

[9]. Conover M. D. et al. Predicting the political alignment of twitter users //Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on. – IEEE, 2011. – С. 192-199.

[10]. Rao D. et al. Classifying latent user attributes in twitter //Proceedings of the 2nd international workshop on Search and mining user-generated contents. – ACM, 2010. – С. 37-44.

[11]. Deitrick W. et al. Gender identification on Twitter using the modified balanced winnow. – 2012

[12]. Miller Z., Dickinson B., Hu W. Gender prediction on twitter using stream algorithms with N-gram character features. – 2012.

[13]. Burger J. D. et al. Discriminating gender on Twitter //Proceedings of the Conference on Empirical Methods in Natural Language Processing. – Association for Computational Linguistics, 2011. – С. 1301-1309.

[14]. Alowibdi J. S., Buy U. A., Yu P. Empirical evaluation of profile characteristics for gender classification on twitter //Machine Learning and Applications (ICMLA), 2013 12th International Conference on. – IEEE, 2013. – Т. 1. – с. 365-369.

[15]. Sloan L. et al. Knowing the tweeters: Deriving sociologically relevant demographics from Twitter //Sociological Research Online. – 2013. – Т. 18. – №. 3. – С. 7.

[16]. Fortunato S. Community detection in graphs //Physics Reports. – 2010. – Т. 486. – №. 3. – С. 75-174.

[17]. Peersman C., Daelemans W., Van Vaerenbergh L. Predicting age and gender in online social networks //Proceedings of the 3rd international workshop on Search and mining user-generated contents. – ACM, 2011. – С. 37-44.
[18]. Nguyen D., Smith N. A., Rosé C. P. Author age prediction from text using linear regression //Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities. – Association for Computational Linguistics, 2011. – С. 115-123.
[19]. Коршунов А., Гомзин А. Тематическое моделирование текстов на естественном языке //Труды Института системного программирования РАН. – 2012. – Т. 23, стр. 215-244. DOI: 10.15514/ISPRAS-2012-23-13
[20]. Molina L. C., Belanche L., Nebot À. Feature selection algorithms: A survey and experimental evaluation //Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on. – IEEE, 2002. – С. 306-313.
[21]. Zheng Z., Wu X., Srihari R. Feature selection for text categorization on imbalanced data //ACM Sigkdd Explorations Newsletter. – 2004. – Т. 6. – №. 1. – С. 80-89.

Methods for Construction of Socio-Demographic Profile of Internet Users

[1,2]A.G. Gomzin <gomzin@ispras.ru>
[1, 2, 3]S.D. Kuznetsov <kuzloc@ispras.ru>
[1] Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia
[2]Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
[3]Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia

Abstract. The paper is devoted to methods for construction of socio-demographic profile of Internet users. Gender, age, political and religion views, region, relationship status are examples of demographic attributes. This work is a survey of methods that detect demographic attributes from user's profile and messages. The most of surveyed works are devoted to gender detection. Age, political views and region are also interested researches. The most popular data sources for demographic attributes extraction are social networks, such as Facebook, Twitter, Youtube.

The most of solutions are based on supervised machine learning. Machine learning allows to find target values (demographic attributes) dependencies from input data and use them to predict the value of the target attribute for the new data. The following problem solving steps are surveyed in the paper: feature extraction, feature selection, model training, evaluation. Researches use different kind of data to predict demographic attributes. The most popular data source is text. Words sequences (n-grams), parts of speech, emoticons, features specific to particular resources (eg, @ mentions and # Hashtags on Twitter) are extracted and used as input for machine learning algorithms. Social graphs are also used as source data.

Communities of users that are automatically extracted from social graph are user as features for attributes prediction.
Text data produces a lot of features. Feature selection algorithms are needed to reduce feature space.
The paper surveys feature selection, classification and regression algorithms, evaluation metrics.

Keywords: demographic attributes; social networks; text processing; machine learning

DOI: 10.15514/ISPRAS-2015-27(4)-7

For citation: Gomzin A.G., Kuznetsov S.D. Methods for Construction of Socio-Demographic Profile of Internet Users. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 129-144 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-7.

References

[1]. Li Q., Kim B. M. Constructing user profiles for collaborative recommender system. Advanced Web Technologies and Applications. – Springer Berlin Heidelberg, 2004. – С. 100-110.
[2]. Bharat K., Lawrence S., Sahami M. Generating user information for use in targeted advertising : patent. 10/750,363 США. – 2003.
[3]. Spisok social'nyh setej. Wikipedia. List of social networks https://ru.wikipedia.org/wiki/Список_социальных_сетей
[4]. Koeshunov A. et al., Opredelenie demograficheskih atributov pol'zovatelej mikroblogov [Microblogs' users' demographic attributes detection]. *Trudy ISP RAN* [*The Proceedings of ISP RAS*], 2013. – T. 25, pp. 179-194. DOI: 10.15514/ISPRAS-2013-25-10
[5]. Filippova K. User demographics and language in an implicit social network. Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. – Association for Computational Linguistics, 2012. – P. 1478-1488.
[6]. Cheng N., Chandramouli R., Subbalakshmi K. P. Author gender identification from text. Digital Investigation. – 2011. – Т. 8. – №. 1. – P. 78-88.
[7]. Leskovec J., Faloutsos C. Sampling from large graphs. Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. – ACM, 2006. – P. 631-636.
[8]. Gjoka M. et al. Walking in Facebook: A case study of unbiased sampling of OSNs. INFOCOM, 2010 Proceedings IEEE. – IEEE, 2010. – P. 1-9.
[9]. Conover M. D. et al. Predicting the political alignment of twitter users. Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on. – IEEE, 2011. – P. 192-199.
[10]. Rao D. et al. Classifying latent user attributes in twitter. Proceedings of the 2nd international workshop on Search and mining user-generated contents. – ACM, 2010. – P. 37-44.
[11]. Deitrick W. et al. Gender identification on Twitter using the modified balanced winnow. – 2012
[12]. Miller Z., Dickinson B., Hu W. Gender prediction on twitter using stream algorithms with N-gram character features. – 2012.

[13]. Burger J. D. et al. Discriminating gender on Twitter. Proceedings of the Conference on Empirical Methods in Natural Language Processing. – Association for Computational Linguistics, 2011. – P. 1301-1309.

[14]. Alowibdi J. S., Buy U. A., Yu P. Empirical evaluation of profile characteristics for gender classification on twitter. Machine Learning and Applications (ICMLA), 2013 12th International Conference on. – IEEE, 2013. – T. 1. – P. 365-369.

[15]. Sloan L. et al. Knowing the tweeters: Deriving sociologically relevant demographics from Twitter. Sociological Research Online. – 2013. – T. 18. – №. 3. – P. 7.

[16]. Fortunato S. Community detection in graphs. Physics Reports. – 2010. – T. 486. – №. 3. – P. 75-174.

[17]. Peersman C., Daelemans W., Van Vaerenbergh L. Predicting age and gender in online social networks. Proceedings of the 3rd international workshop on Search and mining user-generated contents. – ACM, 2011. – P. 37-44.

[18]. Nguyen D., Smith N. A., Rosé C. P. Author age prediction from text using linear regression. Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities. – Association for Computational Linguistics, 2011. – P. 115-123.

[19]. Korshunov A., Gomzin A. Tematicheskoe modelirovanie tekstov na estestvennom yazyke [Topic modeling of natural language texts]. *Trudy ISP RAN* [*The Proceedings of ISP RAS*] – 2012. – T. 23. pp. 215-244. DOI: 10.15514/ISPRAS-2012-23-13

[20]. Molina L. C., Belanche L., Nebot À. Feature selection algorithms: A survey and experimental evaluation. Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on. – IEEE, 2002. – P. 306-313.

[21]. Zheng Z., Wu X., Srihari R. Feature selection for text categorization on imbalanced data. ACM Sigkdd Explorations Newsletter. – 2004. – T. 6. – №. 1. – P. 80-89.

Применение алгоритмов проверки эквивалентности для оптимизации программ

[1, 2, 3, 4]В.А.Захаров <zakh@cs.msu.su>
[2, 4]В.В. Подымов <valdus@yandex.ru>
[1] Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2] Московский государственный университет имени М.В. Ломоносова
119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК
[3]Московский физико-технический институт (государственный университет),
141700, Московская область, г. Долгопрудный, Институтский пер., 9
[4]НИУ Высшая школа экономики, Россия, Москва, 101000, ул. Мясницкая, д. 20

Аннотация. На примере двух моделей программ показано, что задача оптимизации размера программ может быть эффективно решена при помощи процедур проверки эквивалентности программ в рассматриваемых моделях. Основной результат работы – полиномиальные по времени алгоритмы минимизации конечных детерминированных автоматов-преобразователей над конечно порожденными разрешимыми группами и схем последовательных программ, семантика которых определяется конечно порожденными разрешимыми упорядоченными левосократимыми полугруппами. Предложенные алгоритмы можно использовать в качестве теоретической основы для построения эффективных процедур глобальной оптимизации императивных и реагирующих программ.

Ключевые слова: эквивалентность программ; оптимизирующие преобразования; реагирующая программа; схема программ; разрешающий алгоритм.

DOI: 10.15514/ISPRAS-2015-27(4)-8

Для цитирования: Захаров В.А., Подымов В.В. Применение алгоритмов проверки эквивалентности для оптимизации программ. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 145-174. DOI: 10.15514/ISPRAS-2015-27(4)-8.

1. Введение

В программировании задача проверки эквивалентности программ – для двух заданных программ выяснить, имеют ли эти программы одинаковое

поведение, – играет двоякую роль. С одной стороны, эта задача служит индикатором сложности модели вычислений, в которой формализовано понятие программы: оценивая сложность проблемы эквивалентности, можно узнать, насколько выбранная формальная модель программ пригодна для практического применения при решении других задач анализа поведения программ. Именно с этой целью проблема эквивалентности изучается в теории схем программ (см. [41,50,51]). С другой стороны, к задаче проверке эквивалентности программ сводятся многие задачи верификации и оптимизации программ. Если проблема эквивалентности имеет эффективный алгоритм решения, то его можно использовать как универсальную вспомогательную процедуру для решения многих других задач системного программирования. Цель данной статьи – продемонстрировать, как можно использовать быстрые алгоритмы проверки эквивалентности в разных формальных моделях программ для эффективной оптимизации программ.

Алгоритмы проверки эквивалентности программ имеют довольно широкую область применения в системном программировании. С их помощью удается решать некоторые задачи

- рефакторинга программ, связанные с выявлением фрагментов программного кода, реализующих известные алгоритмы [3,38];
- регрессионного анализа кода [5,11,16,21] для обнаружения ошибок, которые могут возникать в процессе модернизации программ;
- верификации и валидации оптимизирующих процедур в компиляторах [13,17,18,20,22,28,30,33,43-45];
- композиции программ [39] для корректного объединения нескольких фрагментов кода, реализующих разные функциональности, в одну программу;
- проверки безопасности потоков данных [7,36], чтобы удостовериться в отсутствии уязвимостей;
- обнаружения вредоносных фрагментов кода (программных «вирусов») [12,42].

Но наиболее важной областью применения алгоритмов проверки эквивалентности является разработка оптимизирующих преобразований программ. Исследования в этом направлении начались на рубеже 60-70 гг XX века (см. [1,6,10,14,19,46,50]) и привели к значительному прогрессу в развитии средств компиляции программ и повышении качества программного кода. Оптимизирующие преобразования программ нуждаются в проверке их

корректности, и это можно осуществить лишь при помощи средств проверки эквивалентности программ. Методы, используемые на практике для этой цели, ограничиваются анализом ациклических фрагментов программ (линейных участков) [5,15,24], конечных разверток циклов [21,31,33], или программ, выполняющих специальные преобразования, для которых удается вычислять инварианты циклов [17,23,32].

Однако алгоритмы проверки эквивалентности программ можно использовать не только как средства контроля корректности оптимизирующих преобразований, но и как средства построения самих оптимизирующих преобразований. Для пояснения этого тезиса обратимся к следующему примеру.

Одна из наиболее простых моделей вычислений – это детерминированные конечные автоматы. Задача оптимизации автоматов состоит в том, чтобы для заданного автомата A построить эквивалентный автомат B (т.е. автомат, распознающий тот же самый язык $L(A) = L(B)$), имеющий наименьшее число состояний. Решение этой задачи можно провести в три этапа.

1. Разбить состояния автомата на классы эквивалентности. Два состояния q' и q'' считаются эквивалентными, если эквивалентны автоматы $A[q']$ и $A[q'']$, имеющие эти состояния в качестве начальных. Именно для выделения классов эквивалентности состояний автомата A используется алгоритм проверки эквивалентности автоматов.
2. Склеить эквивалентные состояния. В каждом классе эквивалентности выбирается произвольное состояние, и в это состояние направляются все переходы автомата, которые ранее вели в другие состояния того же класса эквивалентности.
3. Удалить все бесполезные состояния вместе с входящими в них и исходящими из них переходами. Бесполезным считается всякое состояние, которое недостижимо из начального состояния или из которого недостижимо никакое финальное состояние. Для выявления бесполезных состояний достаточно воспользоваться общеизвестными алгоритмами теории графов.

В результате применения указанных преобразований к произвольному детерминированному конечному автомату A будет получен минимальный (по числу состояний) эквивалентный автомат B, т.е. решена задача оптимизации конечных автоматов.

В этой статье мы покажем, что та же самая стратегия решения задачи оптимизации может быть применена и к более сложным вычислительным системам, моделирующим программы. Мы ограничимся рассмотрением двух таких систем – конечных детерминированных автоматов-преобразователей и алгебраических моделей программ над полугруппами.

Автоматы-преобразователи (машины с конечным числом состояний, трансдьюсеры) возникли как обобщение конечных автоматов Рабина-Скотта. В отличие от конечных автоматов преобразователи вычисляют отношения на множествах конечных слов. Они находят применение в системном программировании для построения простейших компиляторов [2], драйверов, осуществляющих фильтрацию и преобразования строк, изображений, потоков данных [4,37], в системах автоматизированного проектирования управляющих систем для разработки контроллеров [29], в компьютерной лингвистике для создания программ распознавания речи [25] и др. Проблема эквивалентности для конечных детерминированных автоматов-преобразователей над строками была исследована в статьях [9,34,49]. В статье [26] для этого класса преобразователей был предложен алгебраический метод их минимизации.

Преобразователи также могут служить простой моделью реагирующих программ, функция которых состоит в том, чтобы вырабатывать правильные отклики в ответ на внешние воздействия. К числу программ такого рода относятся операционные системы, сетевые протоколы, драйверы, контроллеры. В этих программах разные последовательности действий могут приводить к одному и тому же результату, поэтому элементарные (базовые) действия реагирующей программы можно рассматривать как порождающие элементы некоторой полугруппы. Для конечных детерминированных автоматов-преобразователей над полугруппами в статье [49] был разработан полиномиальный по времени алгоритм проверки эквивалентности. Именно с его помощью для этой разновидности автоматов-преобразователей в данной статье будет предложен алгоритм минимизации.

Алгебраические модели программ были впервые введены в статье [53] как результат синтеза двух моделей вычислений – схем программ Ляпунова-Янова [52] (синтаксическая компонента) и дискретных преобразователей Глушкова-Летичевского [47] (семантическая компонента). Эта простая модель последовательных программ может быть использована для решения широкого спектра задач анализа поведения императивных программ, включая задачи оптимизации. В частности, в статье [54] для одного класса алгебраических моделей программ был предложен метод минимизации схем программ, основанный на построении полной системы эквивалентных преобразований.

В статье [47] было замечено, что задача минимизации схем программ тесно взаимосвязана с задачей проверки эквивалентности. В этой работе была предложена рекурсивная процедура проверки эквивалентности схем программ, в которой преобразования, направленные на минимизацию числа ветвлений переходов в состояниях схемы программы, чередуются с проверкой

эквивалентности схем программ, полученных в результате этих преобразований. В общих чертах этот метод подпадает под описанную выше схему минимизации моделей программ. Однако при отсутствии эффективного алгоритма проверки эквивалентности использовать этот метод для минимизации схем программ невыгодно – его сложность по времени экспоненциально зависит от размера оптимизируемой программы.

В данной статье мы описываем прямой (нерекурсивный) алгоритм минимизации схем программ, семантика операторов которых определяется полугруппами специального вида – левосократимыми полугруппами с неразложимой единицей. В основу этого алгоритма положены идеи, высказанные в работе [47]. Высокая эффективность нашего алгоритма обусловлена явным использованием известных эффективных алгоритмов проверки эквивалентности схем программ в полугрупповых моделях программ [40,48,55].

Статья организована следующим образом. В разделе 2 приведено описание модели вычислений конечных автоматов-преобразователей над полугруппами. В разделе 3 описан алгоритм минимизации для преобразователей, работающих над разрешимыми группами, доказана его корректность. В разделе 4 приведено описание синтаксиса и семантики схем программ в алгебраической модели программ. В разделе 5 описан алгоритм минимизации схем программ над упорядоченными полугруппами и доказана его корректность. В заключении проводится обсуждение используемого в статье подхода к решению задачи оптимизации моделей программ и оцениваются перспективы развития этого подхода и его применения для оптимизации других моделей программ.

2. Автоматы-преобразователи: основные понятия

Пусть задан конечный алфавит Σ. Записью Σ^* обозначим множество всех конечных слов над алфавитом Σ, включая пустое слово λ. Буквы алфавита Σ могут быть истолкованы как элементарные события, происходящие во внешней среде и оказывающие влияние на информационную систему, взаимодействующую с этой средой. В этом случае слова можно понимать как возможный сценарий поведения внешней среды по отношению к информационной системе.

Рассмотрим полугруппу S с бинарной операцией $\cdot$, порожденную множеством элементов B и имеющую единичный (нейтральный) элемент e. Порождающие элементы полугруппы можно рассматривать как элементарные операторы (действия), которые выполняет информационная система в ответ на внешние воздействия. Бинарная операция полугруппы в этом случае соответствует последовательной композиции этих операторов. Если две композиции операторов отвечают одному и тому же элементу полугруппы S, то это означает, что эти последовательности операторов вычисляют одинаковый результат. При этом любой элемент полугруппы может быть представлен в

виде *полугруппового выражения* – последовательной композиции базовых элементов множества B.

Конечным автоматом-преобразователем (далее просто преобразователем) над полугруппой S называется система $\pi = \langle \Sigma, S, Q, q_0, F, T \rangle$, в которой

- Q – конечное множество состояний;
- q_0 – начальное состояние, $q_0 \in Q$;
- F – подмножество финальных состояний, $F \subseteq Q$;
- $T \subseteq Q \times \Sigma \times S \times Q$ – конечное отношение переходов.

Четверки (q, a, s, q') из множества T называются *переходами*; для их обозначения используем запись $q \xrightarrow{a/s} q'$. Для каждого состояния $q, q \in Q$, запись $\pi[q]$ будет обозначать преобразователь $\langle \Sigma, S, Q, q, F, T \rangle$, в котором состояние q играет роль начального состояния. *Размером* преобразователя π называется число, равное сумме числа состояний $|Q|$ и длин полугрупповых выражений, задающих элементы s, которые приписаны переходам преобразователя.

Вычислением преобразователя π на входном слове $w = a_1 a_2 \dots a_n$ называется всякая последовательность переходов

$$run = q_i \xrightarrow{a_1/s_1} q_{i+1} \xrightarrow{a_2/s_2} \cdots \xrightarrow{a_{n-1}/s_{n-1}} q_{i+n-1} \xrightarrow{a_n/s_n} q_{i+n} .$$

Конечный преобразователь может служить моделью последовательной реагирующей программы. Каждый переход $q \xrightarrow{a/s} q'$ означает, что при получении внешнего воздействия a в состоянии управления q эта программа совершает переход в состояние управления q' и выполняет последовательность операторов s. Последовательность таких переходов образует вычисление реагирующей программы. Элемент $s = s_1 \cdot s_2 \cdots s_n$ полугруппы S называется *образом* входного слова w, а пара (w, s) называется *меткой* вычисления run. Запись $q_i \xrightarrow{w/s} q_{i+n}$,будет использоваться для сокращенного обозначения вычисления преобразователя на входном слове w. Если $q_i = q_0$, то вычисление run называется *начальным вычислением*. Если $q_{i+n} \in F$, то вычисление run называется *финальным вычислением*. *Полным вычислением* называется всякое вычисление преобразователя, которое является как начальным, так и финальным. Записью $Lab(\pi)$ обозначим множество меток (w, s) всех полных вычислений преобразователя π; это множество является отношением, которое реализует преобразователь π. Запись $Lang(\pi)$ будем использовать для обозначения множества всех слов, на которых преобразователь π имеет полное вычисление, т.е. $Lang(\pi) = \{w : (w, \alpha) \in Lab(\pi)\}$. Очевидно, для любого конечного автомата-преобразователя π множество $Lang(\pi)$ является регулярным языком.

Преобразователи π' и π'' считаются *эквивалентными* (и обозначаются записью $\pi' \sim \pi''$), если $Lab(\pi') = Lab(\pi'')$. Образ входного слова представляет собой результат вычисления реагирующей программы в ответ на последовательность воздействий среды. Внешний наблюдатель регистрирует результаты лишь тех вычислений, которые достигают финальных состояний управления. Поэтому наблюдаемое поведение реагирующей программы π полностью определяется множеством меток $Lab(\pi)$. Таким образом, эквивалентность преобразователей означает, что соответствующие реагирующие программы имеют одинаковое наблюдаемое поведение. Преобразователь считается *минимальным* (по числу состояний), если не существует эквивалентных ему преобразователей с меньшим числом состояний.

Состояние q преобразователя π считается *полезным*, если хотя бы одно полное вычисление проходит через это состояние. В противном случае состояние будем называть *бесполезным*. Преобразователь считается *детерминированным*, если для любой буквы a и для любого состояния q множество T содержит не более одного перехода вида $q \xrightarrow{a/s} q'$. Для детерминированного преобразователя π введем обозначение

$$\pi(w) = \begin{cases} s, \text{если } (w,s) \in Lab(\pi) \\ \text{неопределено иначе} \end{cases}.$$

Таким образом, два детерминированных преобразователя π' и π'' эквивалентны в том и только том случае, если равенство $\pi'(w) = \pi''(w)$ выполняется для любого слова w.

В статье [49] предложен общий метод построения разрешающих процедур для задач проверки эквивалентности детерминированных преобразователей на полугруппами S , вложимыми в конечно порожденные разрешимые группы. Полугруппа S называется *вложимой* в группу G, если эта группа содержит подполугруппу S' , изоморфную полугруппе S. Полугруппа называется разрешимой, если в ней разрешима *проблема равенства слов*, т.е. существует алгоритм, который для любой пары выражений, составленных из порождающих элементов полугруппы, может определить, представляют ли эти выражения один и тот же элемент полугруппы. В статье [49] также показано, что если проблема равенства слов в группе G разрешима за полиномиальное время, то и проблема эквивалентности конечных детерминированных преобразователей над полугруппой S разрешима за полиномиальное время. Опираясь на этот результат, мы покажем, как можно эффективно решить задачу минимизации конечных детерминированных преобразователей, применяя ту же самую стратегию, какую используют для минимизации детерминированных конечных автоматов. В этой статье при решении этой задачи мы будем полагать, что полугруппа S является группой,

т.е. $S = G$. Для всякого элемента s группы G запись s^- будет обозначать элемент группы G, обратный элементу s.

3. Минимизация детерминированных автоматов-преобразователей над группами

В предложенном методе минимизации автоматов преобразователей используется процедура проверки эквивалентности, однако, конкретный вид этой процедуры не имеет существенного значения. Фактически, эта процедура используется как «черный ящик» или оракул.

Пусть задан конечный детерминированный автомат-преобразователь $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$. Два состояния q' и q'' назовем *подобными*, если существует такой элемент s группы G, что для любого слова w выполняется равенство $s \cdot \pi[q'](w) = \pi[q''](w)$. Этот элемент s будем называть *балансиром* для пары подобных состояний q' и q''. Состояния q' и q'' назовем *сбалансированными*, если они подобны с балансиром e. Алгоритм минимизации конечных детерминированных автоматов-преобразователей над группой определяется на основании следующих утверждений.

Утверждение 1. Отношение подобия и отношение сбалансированности на множестве состояний автомата-преобразователя π являются отношениями эквивалентности.

Доказательство. Рефлексивность отношений подобия и сбалансированности очевидна. Симметричность отношения подобия имеет место ввиду того, что $s \cdot \pi[q'](w) = \pi[q''](w) \Rightarrow \pi[q'](w) = s^- \cdot \pi[q''](w)$. Поскольку $e^- = e$, отсюда следует симметричность отношения сбалансированности. Транзитивность отношения подобия следует из соотношения

$$s_1 \cdot \pi[q'](w) = \pi[q''](w) \ \wedge \ s_2 \cdot \pi[q''](w) = \pi[q'''](w) \Rightarrow s_2 s_1 \cdot \pi[q'](w) = \pi[q'''](w).$$

Поскольку $e \cdot e = e$, отсюда следует транзитивность отношения сбалансированности. QED

Утверждение 2. Если q' и q'' – пара подобных состояний с балансиром s, и при этом состояние q' является финальным, то состояние q'' также является финальным, а s – нейтральный элемент e группы G.

Доказательство. Следует из того факта, что всякое состояние q является финальным тогда и только тогда, когда значение $\pi[q](\lambda)$ определено и при этом равно e. QED

Утверждение 3. Если группа G является разрешимой, то отношение подобия состояний автомата-преобразователя также разрешимо. Более того, если проблема равенства слов в группе G разрешима за полиномиальное время, то

и подобие состояний конечного детерминированного автомата-преобразователя можно проверить за полиномиальное время.

Доказательство. Пусть q' и q'' – пара состояний конечного детерминированного автомата-преобразователя $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$. Если $Lang(\pi[q']) \neq Lang(\pi[q''])$, то состояния q' и q'', очевидно, не являются подобными. Если $Lang(\pi[q']) = Lang(\pi[q'']) = \emptyset$, то состояния q' и q'', очевидно, являются подобными, и любой элемент группы G (в т.ч. нейтральный) может служить балансиром пары q', q''. В заключение рассмотрим случай, когда $Lang(\pi[q']) = Lang(\pi[q'']) = L \neq \emptyset$. Возьмем произвольное слово $w \in L$. Пусть $\pi[q'](w) = \alpha, \pi[q''](w) = \beta$. Тогда состояния q' и q'' могут быть подобными в том и только том случае, когда равенство $\beta \cdot \alpha^{-} \cdot \pi[q'](u) = \pi[q''](u)$ выполняется для любого слова u. Чтобы проверить это равенство, обратимся к следующей паре преобразователей $\pi' = \langle \Sigma, S, Q \cup \{q\}, q, F, T' \rangle$ и $\pi'' = \langle \Sigma, S, Q \cup \{q\}, q, F, T'' \rangle$, где $q \notin Q$, $T' = T \cup \{(q, a, \beta \cdot \alpha^{-}, q')\}$, $T'' = T \cup \{(q, a, e, q'')\}$, и a – произвольная буква алфавита Σ. Из определения этих преобразователей видно, что $Lang(\pi') = Lang(\pi'') = aL$, и для любого слова u справедливы равенства $\pi'(au) = \beta \cdot \alpha^{-} \cdot \pi[q'](u)$ и $\pi''(au) = \pi[q''](u)$. Таким образом, для проверки подобия состояний q' и q'' достаточно проверить эквивалентность преобразователей π' и π''. Проверку эквивалентности преобразователей, работающих над разрешимой группой G, можно провести, используя метод, описанный в статье [49].

QED

Пусть имеются преобразователь π, состояние q этого преобразователя и элемент s группы G. Обозначим записью $Tr_1(\pi, q, s)$ преобразователь, получающийся из π следующим образом:

- каждый переход вида $p \xrightarrow{x/g} q$, где $p \neq q$, заменяется на переход $p \xrightarrow{x/g \cdot s^{-}} q$;
- каждый переход вида $q \xrightarrow{y/h} p$, где $p \neq q$, заменяется на переход $q \xrightarrow{y/s \cdot h} p$;
- каждый переход вида $q \xrightarrow{x/h} q$ заменяется на переход $q \xrightarrow{y/s \cdot h \cdot s^{-}} q$.

Утверждение 4. Каковы бы ни были преобразователь $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$, состояние q, не являющееся ни начальным, ни финальным (т.е. $q \notin F \cup \{q_0\}$), и элемент $s, s \in G$, для преобразователя $\pi_1 = Tr_1(\pi, q, s)$ справедливы соотношения

$Lab(\pi) = Lab(\pi_1)$, т.е. Tr_1 – эквивалентное преобразование трансдьюсеров;

$\pi_1[q](w) = s \cdot \pi[q](w)$ для любого слова w;

$\pi_1[p](w) = \pi[p](w)$ для любого слова w и состояния p, отличного от q.

Доказательство. Поскольку состояние q не является ни начальным, ни финальным, каждое финальное вычисление преобразователя π_1, проходящее через состояние q, имеет вид $p \xrightarrow{a_1/s_1} \cdots \xrightarrow{x/g\cdot s^-} q \xrightarrow{y/s\cdot h} \cdots$. Поэтому для любого слова w и состояния p, отличного от q, добавленные элементы s^- и s на переходах, инцидентных состоянию q, взаимно поглощаются, и верно равенство $\pi_1[p](w) = \pi[p](w)$. Поскольку состояние q не является начальным, отсюда следует равенство $Lab(\pi) = Lab(\pi_1)$. Однако для финального вычисления, начинающегося из состояния q, добавленный элемент s в первом переходе не будет компенсирован, и поэтому имеет место равенство $\pi_1[q](w) = s \cdot \pi[q](w)$. QED

Следствие. Предположим, что q' и q'' – пара подобных состояний с балансиром s в преобразователе π, и при этом состояние q' не является ни начальным, ни финальным. Тогда q' и q'' сбалансированы в преобразователе $Tr_1(\pi, q', s)$.

Пусть имеются преобразователь π и пара состояний q', q'' этого преобразователя. Обозначим записью $Tr_2(\pi, q', q'')$ преобразователь, получающийся из π заменой каждого перехода $p \xrightarrow{x/g} q'$, ведущего в состояние q', на переход $p \xrightarrow{x/g} q''$.

Утверждение 5. Если состояния q' и q'' сбалансированы в преобразователе π, то $\pi \sim Tr_2(\pi, q', q'')$.

Доказательство. Пусть $\pi_1 = Tr_2(\pi, q', q'')$. Рассмотрим произвольное состояние q и покажем, что для любого слова w выполняется равенство $\pi[q](w) = \pi_1[q](w)$. Для этого воспользуемся индукцией по длине слова w. Базис индукции (случай $w = \lambda$) очевидным образом следует из утверждения 2. Рассмотрим случай $w = xu$. Если в преобразователе π имеется переход $q \xrightarrow{x/g} p$, где $p \neq q'$, то этот переход сохраняется в преобразователе π_1. Поэтому $\pi[q](xu) = g \cdot \pi[p](u) = g \cdot \pi_1[p](u) = \pi_1[q](xu)$. Если в преобразователе π имеется переход $q \xrightarrow{x/g} q'$, то в преобразователе π_2 вместо этого перехода используется переход $q \xrightarrow{x/g} q''$. Учитывая, что $\pi[q'](u) = \pi[q''](u)$, можем получить следующую цепочку равенств $\pi[q](xu) = g \cdot \pi[q'](u) = g \cdot \pi[q''](u) = g \cdot \pi_1[q''](u) = \pi_1[q](xu)$. QED

Пусть имеются преобразователь $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$ и состояние q, $q \in Q \setminus \{q_0\}$. Обозначим записью $Tr_3(\pi, q)$ преобразователь, получающийся из π удалением состояния q и всех переходов, входящих в это состояние и исходящих из него. Очевидно следующее

Утверждение 6. Если преобразователь π имеет бесполезное состояние q, отличное от начального, то $\pi \sim Tr_3(\pi, q)$.

Назовем автомат-преобразователь *приведенным*, если он не содержит бесполезных состояний, отличных от начального, а также различных подобных состояний.

Утверждение 7. Автомат-преобразователь минимален по числу состояний тогда и только тогда, когда он приведен.

Доказательство. Необходимость. Если преобразователь π содержит бесполезное состояние q, отличное от начального, то согласно утверждению 6 эквивалентный ему преобразователь $Tr_3(\pi, q)$ имеет меньшее число состояний. Предположим, что преобразователь π содержит различные подобные состояния q' и q'' с балансиром s. Тогда хотя бы одно из этих состояний (например, q') не является начальным. Если хотя бы одно из состояний q' и q'' является финальным, то согласно утверждению 2 оба состояния являются финальными и при этом сбалансированными (т.е. $s = e$). Тогда согласно утверждениям 5 и 6 преобразователь $Tr_3(Tr_2(\pi, q', q''), q')$ эквивалентен преобразователю π и имеет меньшее число состояний. Если оба состояния q' и q'' не являются финальными, то согласно утверждениям 4-6 преобразователь $Tr_3(Tr_2(Tr_1(\pi, q', s), q', q''), q')$ эквивалентен преобразователю π и имеет меньшее число состояний.

Достаточность. Предположим противное: существуют два таких эквивалентных преобразователя $\pi = \langle \Sigma, G, Q, q_0, F, T\rangle$ и $\pi' = \langle \Sigma, G, Q', {q'}_0, F', T'\rangle$, что π является приведенным, π' является минимальным и при этом $|Q'| < |Q|$. Коль скоро преобразователь π' минимальный, у него нет бесполезных состояний. Так как $|Q'| < |Q|$, существует такая пара слов w_1, w_2 и три состояния $q' \in Q'$ и $q_1, q_2 \in Q$, для которых имеют место начальные вычисления

$$q_0' \xrightarrow{w_1/g_1} q', q_0' \xrightarrow{w_2/g_2} q', q_0 \xrightarrow{w_1/h_1} q_1, q_0 \xrightarrow{w_2/h_2} q_2.$$

Поскольку $\pi \sim \pi'$, для любого слова w верны равенства $g_1\pi'[q'](w) = h_1\pi[q_1](w)$ и $g_2\pi'[q'](w) = h_2\pi[q_2](w)$. Значит, $g_1^- h_1\pi[q_1](w) = g_2^- h_2\pi[q_2](w)$ и, следовательно, состояния q_1, q_2 преобразователя π подобны вопреки условию утверждения. Полученное противоречие свидетельствует о справедливости утверждения. QED

Итеративный алгоритм минимизации конечных детерминированных автоматов-преобразователей работает следующим образом. Для заданного преобразователя $\pi = \langle \Sigma, G, Q, q_0, F, T\rangle$ до тех пор, пока это возможно, в произвольном порядке выполняются следующие действия.

1. Выделить бесполезное состояние q, $q \neq q_0$, и применить преобразование $Tr_3(\pi, q)$.

2. Выделить пару различных сбалансированных состояний q', q'', где $q' \neq q_0$, и применить преобразование $Tr_3(Tr_2(\pi, q', q''), q')$.
3. Выделить пару различных подобных состояний q', q'' с балансиром s, где $q' \neq q_0$ и $s \neq e$, и применить преобразование $Tr_3(Tr_2(Tr_1(\pi, q', s), q', q''), q')$.

Каждое из применяемых преобразований сокращает число состояний в преобразователе π. Поэтому алгоритм обязательно завершает работу спустя не более $|Q|$ шагов. Утверждения 2, 4-6 гарантируют, что преобразователь, полученный в результате работы алгоритма, будет эквивалентен исходному, а утверждение 1 – приведенность полученного преобразователя. Поэтому, согласно утверждению 7, по окончании работы описанного алгоритма будет построен минимальный по числу состояний автомат-преобразователь, эквивалентный исходному. Для сведения проверки подобия состояний преобразователя к проверке равенства слов в группе G, а также для вычисления балансира подобных состояний достаточно использовать утверждение 3. Таким образом, верна

Теорема 1. Если проблема равенства слов в группе G разрешима за полиномиальное время, то существует полиномиальный по времени алгоритм минимизации конечных детерминированных преобразователей над группой G.

4. Схемы последовательных программ: основные понятия

Схемы последовательных программ, являющиеся основным объектом исследований теории схем программ, допускают разные формы представления; для их описания можно использовать алгебраические формулы [52], размеченные графы или системы переходов специального вида [47,51,53]. Схемы программ также можно рассматривать как детерминированные автоматы-преобразователи с конечным числом состояний, имеющие особую семантику. Как раз в таком виде схемы программ были представлены в работах [46,47]. Для поддержания синтаксического единообразия анализируемых моделей вычислений мы воспользуемся в данной статье именно этой формой представления схем программ.

Детерминированный автомат-преобразователь $\pi = \langle \Sigma, S, Q, q_0, F, T \rangle$ с входным алфавитом Σ, работающий над полугруппой S, может расцениваться как *схема программ* (далее просто схема), если полагать, что

- множество состояний Q – это множество *точек ветвления* (или просто точек) программы, в которых проводится проверка логических условий;

- состояние q_0 – это *точка входа* в программу, или просто *вход* программы;
- в схеме имеется не более одного финального состояния f – это *точка выхода* из программы, или просто *выход* программы; из этой точки не исходит ни одного перехода, и в ней завершаются все результативные вычисления программы;
- алфавит Σ – это множество *значений логических условий*, в зависимости от которых осуществляется передача управления в ту или иную точку программы;
- полугруппа S – это множество *состояний данных*, с которыми работает программа;
- образующие полугруппы S – это *элементарные операторы* программы, осуществляющие преобразования данных: если $s \in S$ и a – это образующий элемент полугруппы S, то элемент $s' = s \cdot a$ нужно рассматривать как результат применения элементарного оператора a к состоянию данных s;
- полугрупповые выражения $g = a_1 \cdot a_2 \cdot \cdots \cdot a_k$ – это последовательные композиции элементарных операторов программы, которые мы будем называть *операторными цепочками*;
- каждый переход $p \xrightarrow{x/g} q$ преобразователя помечен непустой операторной цепочкой g и описывает *линейный участок* программы, в котором цепочка g выполняется в том случае, когда в точке ветвления p логические условия принимают значение x, и после выполнения этой цепочки операторов управление передается в точку ветвления q.

Вычисление схемы – это чередующаяся последовательность проверок логических условий в точках ветвления, сопровождающихся выбором соответствующих линейных участков, и выполнений операторных цепочек, приписанных этим участкам. Успешные вычисления схемы начинаются в ее входе и завершаются в ее выходе. Состояние данных, которое достигается по завершении успешного вычисления, считается результатом вычисления.

Приведенные соображения, определяющие, по сути дела, основные принципы устройства и функционирования императивных программ, могут быть более строго формализованы следующим образом. Пусть заданы алфавит (множество значений логических условий) Σ и полугруппа (множество состояний данных) S. *Интерпретацией* называется всякая функция $I: S \to \Sigma$, которая ставит в соответствие каждому состоянию данных некоторое значение логических условий. *Вычислением* схемы π в интерпретации I называется последовательность переходов этой схемы

$$run(\pi, I) = q_0 \xrightarrow{a_1/g_1} q_1 \xrightarrow{a_2/g_2} \cdots \xrightarrow{a_{n-1}/g_{n-1}} q_{n-1} \xrightarrow{a_n/g_n} \cdots,$$

удовлетворяющая двум условиям:

1. q_0 – вход схемы π;
2. для любого $i, i \geq 1$, выполняется равенство $a_i = I(e \cdot g_1 \cdot g_2 \cdot \cdots \cdot g_{i-1})$.

Вычисление, оканчивающееся в выходе $q_n = f$, называется *терминальным вычислением*; состояние данных $s_n = g_1 \cdot g_2 \cdot \cdots \cdot g_n$, с которым завершается терминальное вычисление $run(\pi, I)$, считается *результатом вычисления схемы π в интерпретации I* и обозначается записью $\pi(I)$. Если схема π не имеет терминальных вычислений в интерпретации I, то результат вычисления $\pi(I)$ считается неопределенным. Схемы π_1 и π_2 считаются *эквивалентными* (и обозначаются записью $\pi_1 \approx \pi_2$), если равенство $\pi_1(I) = \pi_2(I)$ выполняется для любой интерпретации I. Схема считается *минимальной* (по числу точек ветвления), если не существует эквивалентной ей схемы с меньшим числом точек ветвления.

Некоторые понятия, введенные для автоматов-преобразователей, сохраняют тот же смысл и для схем программ. Особо отметим лишь понятия полезности и бесполезности точки ветвления, которые для схем программ имеют несколько иное значение. Точка ветвления схемы π является *полезной*, если через нее проходит вычисление этой схемы хотя бы в одной интерпретации I, и *бесполезной* иначе. Полезность состояния автомата-преобразователя равносильна тому, что оно лежит на каком-либо пути из начального состояния в финальное. Для схем программ это в общем случае неверно.

Полугруппа S называется *упорядоченной*, если для любой пары ее элементов h, g выполнено соотношение $g \cdot h = g \Rightarrow h = e$, и *левосократимой* (или обладающей свойством левого сокращения), если для любой тройки g, h_1, h_2 ее элементов выполнено соотношение $g \cdot h_1 = g \cdot h_2 \Rightarrow h_1 = h_2$. В следующем разделе приводятся описание и анализ алгоритма минимизации схем программ над разрешимыми упорядоченными левосократимыми полугруппами. Некоторые конструкции и технические приемы этого алгоритма были предложены в работе [47], однако авторы этой работы осуществляли минимизацию схем программ по другому параметру –

быстродействию схемы. Кроме того, предложенный ими перекрестно-рекурсивный метод проводит совместное решение двух задач – минимизации и проверки эквивалентности схем программ. Процедура минимизации обращается к процедуре проверки эквивалентности при выполнении упрощающих преобразований, а для проверки эквивалентности нужно, в свою очередь, осуществить минимизацию схем программ. Такой рекурсивный алгоритм имеет экспоненциальную сложность и не представляет практической ценности. Вместе с тем, в целом ряде статей (см., например [40-42,48,55]) были предложены полиномиальные по времени алгоритмы проверки эквивалентности схем программ, работающих над упорядоченными полугруппами. Опираясь на них, мы предлагаем более простое и эффективное решение задачи минимизации схем программ. Фактически, для этого нам пришлось вычленить процедуру минимизации из решения, описанного в статье [47], совместить ее с классической процедурой минимизации конечных автоматов-распознавателей и адаптировать к явному использованию алгоритмов проверки эквивалентности схем программ.

5. Минимизация схем программ над упорядоченными левосократимыми полугруппами

В этом разделе мы рассматриваем схемы программ, работающие над разрешимой упорядоченной левосократимой полугруппой S. Как и в случае автоматов-преобразователей, процедура проверки эквивалентности схем программ используется в виде «черного ящика». Ранее мы отметили, что принадлежность точки ветвления какому-либо пути из входа в выход, вообще говоря, еще не свидетельствует о ее полезности. Однако следующее утверждение показывает, что для рассматриваемых нами полугрупп эти понятия оказываются равносильными.

Утверждение 8 [38]. Пусть S – упорядоченная левосократимая полугруппа, и в схеме $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$ имеется последовательность переходов $q_0 \xrightarrow{a_1/g_1} q_1 \xrightarrow{a_2/g_2} \cdots \xrightarrow{a_{n-1}/g_{n-1}} q_{n-1} \xrightarrow{a_n/g_n} \cdots$. Тогда существует интерпретация, в которой эта последовательность переходов является вычислением.

Следствие. Точка q схемы π полезна тогда и только тогда, когда она лежит на каком-либо пути из входа в выход.

Пусть имеются схема π и точка q этой схемы. Обозначим записью $Mod_1(\pi, q)$ схему, получающуюся из π удалением точки q и всех переходов, начинающихся или оканчивающихся в этой точке. Тогда очевидно

Утверждение 9. Если q – бесполезная точка схемы π, отличная от входа, то $\pi \approx Mod_1(\pi, q)$.

Точка q называется *существенной* в схеме π, если существуют такие интерпретации I_1, I_2, отличающиеся только значением $I_1(e) \neq I_2(e)$ в нейтральном элементе полугруппы S (начальном состоянии данных), что

результаты вычислений схемы $\pi[q]$ в этих интерпретациях различны: $\pi[q](I_1) \neq \pi[q](I_2)$. В противном случае точка q называется *фиктивной*. Пусть имеются схема π, точка q этой схемы и логическое условие a. Обозначим записью $Mod_2(\pi, q, a)$ схему, получающуюся из π следующим образом:

- если в схеме π содержится переход вида $q \xrightarrow{a/h} q'$, то каждый переход $q \xrightarrow{b/g} q''$, начинающийся в точке q, заменяется на переход $q \xrightarrow{b/h} q'$;
- в противном случае из схемы удаляются все переходы, начинающиеся в точке q.

Утверждение 10 [45]. Пусть a – произвольное логическое условие. Тогда точка q схемы π фиктивна в том и только том случае, если $\pi \approx Mod_2(\pi, q, a)$.

Пусть имеются схема π, состояние q этой схемы и логическое условие a. Обозначим записью $Mod_3(\pi, q, a)$ схему, получающуюся из π следующим образом:

- если в схеме π содержится переход вида $q \xrightarrow{a/h} q'$, то каждый переход $p \xrightarrow{b/g} q$, оканчивающийся в точке q, заменяется на переход $p \xrightarrow{b/gh} q'$;
- в противном случае из схемы удаляются все переходы, оканчивающиеся в точке q.

Утверждение 11. Пусть q – фиктивная точка схемы π, отличная от выхода, а a – логическое условие. Тогда $\pi \approx Mod_3(\pi, q, a)$.

Доказательство. Согласно утверждению 10, достаточно показать эквивалентность схем $\pi' = Mod_2(\pi, q, a)$ и $\pi'' = Mod_3(\pi, q, a)$. Рассмотрим произвольную интерпретацию I. Если схема π не содержит перехода, начинающегося в точке q и помеченного буквой a, то ни одно из вычислений $run(\pi', I)$, $run(\pi'', I)$ не достигает выхода. В противном случае вычисление $run(\pi'', I)$ получается из вычисления $run(\pi', I)$ заменой каждой пары соседних переходов $q' \xrightarrow{b/h} q \xrightarrow{c/g} q''$, проходящих через точку q, на переход $q' \xrightarrow{b/hg} q''$. Легко видеть, что в обоих случаях $\pi'(I) = \pi''(I)$. QED

Точки q_1, q_2 схем π_1, π_2 назовем *эквивалентными* (и обозначим записью $q_1 \approx_{(\pi_1,\pi_2)} q_2$), если эквивалентны схемы $\pi_1[q_1]$ и $\pi_2[q_2]$. Если $\pi_1 = \pi_2 = \pi$, то для краткости вместо $\approx_{(\pi_1,\pi_2)}$ будем писать $\approx_\pi$.

Утверждение 12. Пусть точки q, q' эквивалентны в схеме π. Тогда точка q существенна в том и только в том случае, если существенна точка q'.

Доказательство. Допустим, что точка q существенна, а точка q' - нет. Тогда существуют интерпретации I_1, I_2, отличающиеся только значением в начальном состоянии данных ($I_1(e) \neq I_2(e)$), для которых верно соотношение $\pi[q](I_1) \neq \pi[q](I_2)$. При этом $\pi[q'](I_1) = \pi[q'](I_2)$ вопреки условию эквивалентности точек q, q'. QED

Схему назовем *безызбыточной*, если:

- она не содержит бесполезных точек, кроме, быть может, входа;
- она не содержит фиктивных точек, кроме, быть может, входа и выхода;
- если ее вход фиктивен, то в него не ведет ни одного перехода.

Для безызбыточной схемы удается построить разбиение точек на классы эквивалентности, настолько же удобное для анализа, как и разбиение состояний обычного конечного автомата-распознавателя.

Утверждение 13. Пусть π – безызбыточная схема, $q \xrightarrow{a/h} p$ – переход этой схемы, и $q \approx_\pi q'$. Тогда в схеме π имеется также переход $q' \xrightarrow{a/h} p'$, причем $p \approx_\pi p'$.

Доказательство. Если вход схемы π бесполезен, то схема не содержит других точек, и справедливость утверждения очевидна. Далее полагаем, что вход схемы полезен. Из полезности точки q и утверждения 8 следует, что существует такая интерпретация I, что вычисление $run(\pi[q], I)$ результативно и начинается с перехода $q \xrightarrow{a/h} p$. Так как $q \approx_\pi q'$, вычисление $run(\pi[q'], I)$ также результативно. Пусть вычисление $run(\pi[q'], I)$ начинается с перехода $q' \xrightarrow{b/g} p'$. Тогда $b = I(e) = a$. Осталось обосновать два соотношения: $h = g$ и $p \approx_\pi p'$.

Предположим, что $h \neq g$. Возможен один из трех случаев: 1) существует такое состояние данных u, что $hu = g$; 2) существует такое состояние данных u, что $h = gu$; 3) для любого состояния данных u верны соотношения $hu \neq g$ и $h \neq gu$. Рассмотрим подробно только первый случай: $h \neq g$, $hu = g$; те же самые рассуждения можно провести и в двух других случаях. Если p – выход, то в силу упорядоченности полугруппы S вычисления $run(\pi[q], I)$, $run(\pi[q'], I)$ не могут иметь одинаковый результат. Значит, точка p не является ни входом, ни выходом и существенна. Тогда существуют две такие интерпретации I_1, I_2, различающиеся только значением в начальном состоянии данных ($I_1(e) \neq I_2(e)$), что $\pi[p](I_1) \neq \pi[p](I_2)$. Рассмотрим интерпретации I_1', I_2' со следующими свойствами: для любого состояния данных s верно $I_i'(hs) = I_i(s)$; $I_1'(e) = I_2'(e) = a$. Такие интерпретации можно корректно определить для любой упорядоченной левосократимой полугруппы S. Тогда $\pi[q](I_1') = h \cdot \pi[p](I_1) \neq h \cdot \pi[p](I_2) = \pi[q](I_2')$. Заметим, что

вычисления $run(\pi[q'], I_1')$ и $run(\pi[q'], I_2')$ начинаются с перехода $q' \xrightarrow{a/g} p'$, и для рассматриваемого случая верно равенство $run(\pi[q'], I_1') = run(\pi[q'], I_2')$. Поэтому $\pi[q'](I_1') = \pi[q'](I_2')$. Но коль скоро $q \approx_\pi q'$, справедливы также равенства $\pi[q](I_1') = \pi[q'](I_1') = \pi[q'](I_2') = \pi[q](I_2')$ вопреки полученному выше соотношению $\pi[q](I_1') \neq \pi[q](I_2')$. Этим противоречием обосновывается равенство $h = g$.

Теперь обоснуем соотношение $p \approx_\pi p'$. Предположим противное: точки p, p' не эквивалентны в схеме π. Тогда существует такая интерпретация I, что $\pi[p](I) \neq \pi[p'](I)$. Основываясь на этой интерпретации, построим интерпретацию I', обладающую следующими свойствами: для любого состояния данных s верно равенство $I'(hs) = I(s)$; $I'(e) = a$. Из эквивалентности точек q, q' следует цепочка равенств $h \cdot \pi[p](I) = \pi[q](I') = \pi[q'](I') = g \cdot \pi[p'](I)$. Ввиду того что $h = g$ и полугруппа S является левосократимой, должно быть справедливо равенство $\pi[p](I) = \pi[p'](I)$, которое противоречит установленному ранее соотношению $\pi[p](I) \neq \pi[p'](I)$. QED

Пусть имеется схема π, содержащая точки q и q'. Обозначим записью $Mod_4(\pi, q, q')$ схему, получаемую из π заменой каждого перехода $p \xrightarrow{a/h} q$, оканчивающегося в точке q, на переход $p \xrightarrow{a/h} q'$.

Утверждение 14. Пусть π – безызбыточная схема, и q, q' – ее различные эквивалентные существенные точки. Тогда $\pi \approx Mod_4(\pi, q, q')$.

Доказательство. Пусть $\pi' = Mod_4(\pi, q, q')$. Рассмотрим произвольную интерпретацию I и вычисления

$$run(\pi, I) = q_0 \xrightarrow{a_1/h_1} q_1 \xrightarrow{a_2/h_2} q_2 \xrightarrow{a_3/h_3} \dots,$$

$$run(\pi', I) = p_0 \xrightarrow{b_1/g_1} p_1 \xrightarrow{b_2/g_2} p_2 \xrightarrow{b_3/g_3} \dots.$$

Для обоснования утверждения достаточно показать, что эти вычисления имеют одинаковую длину и при этом для любого $i, i \geq 1$, справедливы соотношения $h_i = g_i$ и $q_i \approx_\pi p_i$. Воспользуемся индукцией по индексу i, предварительно заметив, что ввиду того, что схема π безызбыточна, точки $q_1, q_2, \dots, p_1, p_2, \dots$ существенны в этой схеме.

Базис индукции. Если в схеме π не содержится переход, начинающийся в точке $q_0 = p_0$ и помеченный буквой $I(e)$, то оба вычисления $run(\pi, I)$, $run(\pi', I)$ имеют длину 0. Иначе оба этих вычисления непусты, $a_1 = I(e) = b_1$, и согласно определению преобразования Mod_4 верно равенство $h_1 = g_1$. Если переход $q_0 \xrightarrow{a_1/h_1} q_1$ содержится в схеме π', то $q_1 = p_1$. Иначе $q_1 = q$ и $p_1 = q'$. В любом случае справедливо соотношение $q_1 \approx_\pi p_1$.

Индуктивный переход. Предположим, что $q_i \approx_\pi p_i$, $h_1 = g_1$, …, $h_i = g_i$, и длина хотя бы одного из вычислений $run(\pi, I)$, $run(\pi', I)$ превышает i. Для

определенности будем считать, что таковым является вычисление $run(\pi, I)$, содержащее переход $q_i \xrightarrow{a_{i+1}/h_{i+1}} q_{i+1}$ (аналогичные рассуждения применимы, если предполагать, что длина вычисления $run(\pi', I)$ превышает i). Согласно утверждению 13 схема π содержит переход $p_i \xrightarrow{a_{i+1}/h_{i+1}} p$, где $p \approx_\pi q_{i+1}$. Из равенств $a_{i+1} = h_1 \cdot \ldots \cdot h_i = g_1 \cdot \ldots \cdot g_i$ следует, что если $p \neq q$, то $p_i \xrightarrow{a_{i+1}/h_{i+1}} p$ – $(i+1)$-й по порядку переход вычисления $run(\pi', I)$, а иначе таковым является переход $p_i \xrightarrow{a_{i+1}/h_{i+1}} q'$. В любом случае длина вычисления $run(\pi', I)$ превосходит i и справедливы соотношения $g_{i+1} = h_{i+1}$, $q_{i+1} \approx_\pi p_{i+1}$. QED

Безызбыточную схему назовем *приведенной*, если она не содержит различных эквивалентных точек.

Утверждение 15. Схема программ минимальна тогда и только тогда, когда она является приведенной.

Доказательство. Необходимость. Если схема π содержит бесполезную точку q, отличную от входа, то согласно утверждению 9 схема $Mod_1(\pi, q)$ эквивалентна π и содержит меньшее число точек. Если схема π содержит фиктивную полезную точку q, отличную от входа и выхода, то согласно утверждению 8 в схеме π содержится переход $q \xrightarrow{a/h} q'$, где $q \neq q'$. Тогда согласно утверждениям 8, 9, 11 схема $Mod_1(Mod_3(\pi, q, a), q)$ эквивалентна π и содержит меньшее число точек. Если схема π безызбыточна и содержит различные эквивалентные точки q', q'', то по утверждению 12 обе эти точки существенны, и хотя бы одна из них (скажем, q') отлична от входа. Тогда согласно утверждениям 8, 9, 14 схема $Mod_1(Mod_4(\pi, q', q''), q')$ эквивалентна π и содержит меньшее число точек. Осталось рассмотреть следующий случай. Схема π: не содержит бесполезных точек, отличных от входа q_0; не содержит фиктивных точек, отличных от входа и выхода; содержит различные эквивалентные точки q', q''; не является безызбыточной. Тогда по утверждению 12 схема π содержит переход $q_0 \xrightarrow{a/h} q$, где $q \neq q_0$, и используя утверждение 11, получим безызбыточную схему $Mod_3(\pi, q_0, a)$, эквивалентную π, имеющую столько же точек и содержащую различные эквивалентные точки q', q''. Способ получения из нее эквивалентной схемы с меньшим числом состояний описан ранее.

Достаточность. Пусть $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$ – произвольная приведенная схема. Рассмотрим произвольную эквивалентную ей минимальную схему $\pi' = \langle \Sigma, G, Q', q_0', F', T' \rangle$. Как было показано выше, схема π' является приведенной. Так как $\pi \approx \pi'$, справедливо соотношение $q_0 \approx_{(\pi,\pi')} q_0'$. Тогда, используя утверждение 13, нетрудно показать, что для каждой точки схемы π существует эквивалентная ей точка схемы π', и для каждой точки схемы π' существует эквивалентная ей точка схемы π. Если схема π' содержит точку q', эквивалентную двум различным точкам q_1, q_2 схемы π, то $q_1 \approx_\pi q_2$, что

противоречит приведенности схемы π. Значит, отношение эквивалентности между точками схемы π и точками схемы π' является взаимно-однозначным соответствием. Таким образом, $|Q| = |Q'|$. QED

Итеративный алгоритм минимизации схем программ работает в два этапа: вначале для исходной схемы строится эквивалентная ей безызбыточная схема, а затем построенная безызбыточная схема преобразуется к приведенному виду.

На первом этапе, пока это возможно, для заданной схемы $\pi = \langle \Sigma, S, Q, q_0, F, T \rangle$ выполняются следующие действия.

1. Выделить бесполезную точку q, $q \neq q_0$, и применить преобразование $Mod_1(\pi, q)$.
2. Выделить полезную фиктивную точку q, $q \notin F \cup \{q_0\}$, выбрать произвольный переход $q \xrightarrow{a/h} q'$, где $q' \neq q$, и применить преобразование $Mod_1(Mod_3(\pi, q, a), q)$.
3. Если вход схемы полезен и фиктивен, то выбрать переход $q_0 \xrightarrow{a/h} q$, где $q \neq q_0$, и применить преобразование $Mod_3(\pi, q_0, a)$.
4. Если вход схемы бесполезен, то удалить все переходы, начинающиеся во входе.

Действия 3, 4 выполняются не более одного раза. Каждое из действий 1, 2 сокращает число точек схемы. Поэтому первый этап минимизации завершается спустя не более $|Q|$ шагов. Согласно утверждениям 8, 9 и 11 в результате выполнения действий первого этапа будет построена безызбыточная схема, эквивалентная исходной схеме. На основании утверждения 8 проверка полезности точки сводится к проверке достижимости вершин в графе переходов схемы. Также утверждением 8 гарантируется наличие переходов, требуемых в действиях 2 и 3. Из утверждения 10 следует, что проверка фиктивности точки q в схеме π сводится к проверке эквивалентности схем π и $Mod_2(\pi, q, a)$.

На втором этапе для безызбыточной схемы $\pi = \langle \Sigma, S, Q, q_0, F, T \rangle$ выполняется одно и то же действие, пока оно может быть выполнено: выделить пару различных эквивалентных точек q', q'', где $q' \neq q_0$, и применить преобразование $Mod_1(Mod_4(\pi, q', q''), q')$. Согласно определению безызбыточности и утверждению 12 обе точки q', q'' отличны от входа и выхода и существенны. Утверждения 9 и 14 гарантируют эквивалентность исходной и преобразованной схем. Применяемое преобразование сокращает число состояний в схеме. Поэтому второй этап минимизации схемы завершается спустя не более $|Q|$ шагов. По завершении второго этапа

получается приведенная схема. Ее минимальность гарантируется утверждением 15. Таким образом, справедлива

Теорема 2. Если проблема эквивалентности схем программ над упорядоченной левосократимой полугруппой S разрешима за полиномиальное время, то существует полиномиальный по времени алгоритм минимизации схем программ над полугруппой S.

6. Заключение

Мы показали, что для некоторых моделей программ эффективные алгоритмы решения задачи минимизации размеров программ могут быть легко построены на основе алгоритмов проверки эквивалентности программ. Это свидетельствует о важной роли, которую могут сыграть алгоритмы проверки эквивалентности программ в решении задач глобальной оптимизации программ. Полученные результаты открывают новое направление исследований проблемы эквивалентности в теории моделей программ – разработка методов применения процедур проверки эквивалентности схем программ для построения оптимизирующих преобразований программ. На следующем этапе исследований можно предпринять попытку усилить утверждения теорем 1 и 2 настоящей статьи. В разделе 3 описан алгоритм минимизации конечных детерминированных автоматов-преобразователей, работающих над разрешимыми группами. Вероятно, этот алгоритм можно распространить и на класс автоматов-преобразователей, работающих над полугруппами, вложимыми в разрешимые группы. Именно для этого класса преобразователей в статье [49] была доказана разрешимость проблемы эквивалентности, и поэтому здесь для решения задачи минимизации также можно воспользоваться процедурами проверки эквивалентности. В разделе 5 предложен алгоритм минимизации схем программ, работающие над разрешимой упорядоченными левосократимыми полугруппами. Однако в статьях [40,42,48] были предложены полиномиальные по времени алгоритмы проверки эквивалентности схем программ, работающих над другими классами полугрупп; эти алгоритмы также можно было бы использовать в качестве вспомогательного средства для разработки эффективных процедур минимизации схем программ.

Список литературы

[1]. Abel N.E., Bell J.R. Global optimization in compilers. Proceedings of the First USA-Japan Computer Conference. 1972.

[2]. Aho A.V., Sethi R., Ullman J.D. Compilers: principles, techniques, and tools. Addison-Wesley, Reading, MA, 1986.

[3]. Alias, C., Barthou, D. On the recognition of algorithm templates. Proceedings of the 2-nd International Workshop on Compiler Optimization Meets Compiler Verification, Electronic Notes in Theoretical Computer Science, 2004, v. 82, N 2, p. 395–409.

[4]. Alur R., Cerny P. Streaming transducers for algorithmic verification of single-pass list-processing programs. Proceedings of 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2011, p. 599-610.

[5]. Arons T., Elster E., Fix L., Mador-Haim S., Mishaeli M., Shalev J., Singerman E., Tiemeyer A., Vardi M. Y., Zuck L. D. Formal verification of backward compatibility of microcode. CAV, 2005, p. 185–198.

[6]. Allen F.E., Cocke J.L. A catalogue of optimizing transformations. Design and Optimization of Compilers, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[7]. Barthe, G., D'Argenio, P.R., Rezk, T. Secure information flow by self-composition. Proceedings of the 17th IEEE Workshop on Computer Security Foundations, 2004, p. 100.

[8]. Benton, N. Simple relational correctness proofs for static analyses and program transformations. Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2004, ACM, New York.

[9]. Blattner M, Head T. The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences, 1979, v. 19, p. 45-49.

[10]. Busam V.A., England D.E. Optimization of expressions in Fortran. Communications of the Association for Computing Machinary. 1969, v. 12, N 12, p. 666-674.

[11]. Chaki, S., Gurfinkel, A., Strichman, O. Regression verification for multi-threaded programs. Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation, 2012, p. 119-135.

[12]. Christodorescu M., Jha S., Seshia S. A., Song D., Bryant R. E. Semantic-aware malware detection. Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, 2005.

[13]. Dissegna, S., Logozzo, F., Ranzato, F. Tracing compilation by abstract interpretation. Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2014, p. 47-59.

[14]. 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.

[15]. Feng X., Hu A. J. Cutpoints for formal equivalence verification of embedded software. Proceedings of the 5-th ACM International Conference on Embedded Software, 2005, p. 307–316.

[16]. Godlin, B., Strichman, O. Regression verification. Proceedings of the 46th Annual Design Automation Conference, 2009, IEEE Computer Society, Washington.

[17]. Goldberg, B., Zuck, L., Barrett, C. Into the loops: practical issues in translation validation for optimizing compilers. Theoretical Computer Science, 2005, v. 132, N 1, p. 53–71.

[18]. Guo, S.-Y., Palsberg, J. The essence of compiling with traces. Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2011, p. 563-574.

[19]. Kildall G. A unified approach to global program optimization. Conference Record of the Annual ACM SIGPLAN-SIGACT Simposium on Principles of Programming Languages, 1973, p. 194-206.

[20]. Kundu, S., Tatlock, Z., Lerner, S. Proving optimizations correct using parameterized program equivalence. Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2009, p. 327-337.

[21]. Lahiri S. K., Hawblitzel C., Kawaguchi M., Rebelo H. SYMDIFF: A language-agnostic semantic diff tool for imperative programs. Proceedings of the 24th International Conference on Computer Aided Verification, 2012, p. 712–717.

[22]. Le V., Afshari, M., Su, Z. Compiler validation via equivalence modulo inputs. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2014, ACM, New York.
[23]. Lopes N. P., · Monteiro J. Automatic equivalence checking of programs with uninterpreted functions and integer arithmetic. International Journal on Software Tools for Technology Transfer, 2015, N 1, p. 1-16.
[24]. Matsumoto, T., Saito, H., Fujita, M. Equivalence checking of C programs by locally performing symbolic simulation on dependence graphs. Proceedings of the 7-th International Symposiumon Quality Electronic Design, 2006, p. 370-375.
[25]. Mohri M. Finite state transducers in language and speech processing. Computer Linguistics, 1997, v. 23, N 2.
[26]. Mohri M. Minimization algorithms for sequential transducers. Theoretical Computer Science, 2000, v. 234, p. 177-201.
[27]. Namjoshi, K.S., Zuck, L.D. Witnessing program transformations. Proceedings of the 20th International Conference on Static Analysis, 2013, Springer, Berlin, Heidelberg.
[28]. Necula, G.C. Translation validation for an optimizing compiler. Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation. 2000, p. 83-94.
[29]. Nerode A., Kohn W. Models for hybrid systems: automata, topology, controllability, observability. Cornell University, Technical Report 93-28, 1993, MIT Press, Cambridge.
[30]. Pnueli, A., Siegel, M., Singerman, E. Translation validation. Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, 1998, p. 151-166.
[31]. Ramos D. A., Engler D. R. Practical, low-effort equivalence verification of real code. CAV, 2011, p. 669–685.
[32]. Sharma R., Schkufza E., B/ Churchill, A. Aiken. Data-driven equivalence checking. Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications, 2013, p. 391-406.
[33]. Shashidhar, K.C., Bruynooghe,M., Catthoor, F., Janssens, G. Verification of source code transformations by program equivalence checking. Proceedings of the 14th International Conference on Compiler Construction. Springer, 2005, Berlin, Heidelberg.
[34]. de Souza R. On the decidability of the equivalence for k-valued transducers. Proceedings of 12th International Conference on Developments in Language Theory, 2008, p. 252-263.
[35]. Stepp, M., Tate, R., Lerner, S. Equality-based translation validator for LLVM. Proceedings of the 23rd International Conference on Computer Aided Verification, 2011, p. 737-742.
[36]. Terauchi, T.: Aiken, A. Secure information flow as a safety problem. Proceedings of the 12th International Conference on Static Analysis. 2005, p. 352-367.
[37]. Veanes M., Hooimeijer P., Livshits B., et al. Symbolic finite state transducers: algorithms and applications. Proceedings of the 39th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2012.
[38]. Verdoolaege, S., Janssens, G., Bruynooghe, M. Equivalence checking of static affine programs using widening to handle recurrences. Proceedings of the 21st International Conference on Computer Aided Verification, 2009, Springer, Berlin, Heidelberg.
[39]. Yang W., Horowitz S., Reps T. A program integration algorithm that accommodates semantics-preserving transformations. ACM Transactions of Software Engineering and Methodology, 1992, v. 1, N 3, p. 310–354.

[40]. 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.

[41]. Zakharov V.A.. The equivalence problem for computational models: decidable and undecidable cases. Lecture Notes in Computer Science, 2001, v. 2055, p. 133-153.

[42]. Zakharov V.A., Kuzurin N.N., Podlovchenko R.I., Shcherbina V.V. Using algebraic models of programs for detecting metamorphic malwares. Труды Института Системного программирования, Том 12, 2007. с. 77-94.

[43]. Zaks, A., Pnueli, A. CoVaC: Compiler validation by program analysis of the cross-product. In: Proceedings of the 15th International Symposium on Formal Methods. Springer, 2008, Berlin, Heidelberg.

[44]. Zhao, J., Nagarakatte, S., Martin, M.M.K., Zdancewic, S. Formal verification of SSA-based optimizations for LLVM. Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2013, ACM, New York.

[45]. Zuck, L., Pnueli, A., Goldberg, B., Barrett, C., Fang, Y., Hu, Y. Translation and run-time validation of loop transformations. Formal Methods for System Design, 2005, v. 27, N 3, p. 335–360.

[46]. Глушков В.М. К вопросу о минимизации микропрограмм и схем алгоритмов. Кибернетика. 1966, N 5, с. 1-3.

[47]. Глушков В.М., Летичевский А.А. Теория дискретных преобразователей. Избранные вопросы алгебры и логики: сб. статей. Новосибирск:Наука, 1973, с. 5-39.

[48]. Захаров В.А. Быстрые алгоритмы разрешения эквивалентности пропозициональных операторных программ на упорядоченных полугрупповых шкалах. Вестник Московского университета, сер. 15, Вычислительная математика и кибернетика. - 1999, N 3. с. 29-35.

[49]. Захаров В.А. Моделирование и анализ последовательных реагирующих программ. Труды Института системного программирования РАН, том 27, выпуск 2, 2015, стр. 221-250. DOI: 10.15514/ISPRAS-2015-27(2)-13

[50]. Касьянов В.Н. Оптимизирующие преобразования программ. М.: Наука, 1988. 336 с.

[51]. Котов В.Е., Сабельфельд В.К. Теория схем программ. М.:Наука, 1991. 348 с.

[52]. Ляпунов А.А. О логических схемах программ. Проблемы кибернетики, вып. 1, М.:Физматгиз, 1958, с. 46-74.

[53]. Подловченко Р.И. О минимизации схем программ с перестановочными блоками. Программирование, 2008, N 4, с. 72-77.

[54]. Подловченко Р.И. Полугрупповые модели программ. Программирование. 1981, N 4, с.3-13.

[55]. Подымов В.В., Захаров В.А. Полиномиальный алгоритм проверки эквивалентности в модели программ с перестановочными и подавляемыми операторами. Труды ИСП РАН, 2014, вып. 3, с. 145-166. DOI: 10.15514/ISPRAS-2014-26(3)-8

On the Application of Equivalence Checking Algorithms for Program Minimization

[1, 2, 3, 4]*V.A.Zakharov <zakh@cs.msu.su>*
[2, 4]*V.V.Podymov <valdus@yandex.ru>*
Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia
[2]*Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation*
[3]*Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[4]*Higher School of Economics, National Research University, 20 Myasnitskaya Ulitsa, Moscow 101000, Russia*

Annotation. Equivalence checking algorithms found vast applications in system programming; they are used in software refactoring, security checking, malware detection, program integration, regression verification, compiler verification and validation. In this paper we show that equivalence checking procedures can be utilized for the development of global optimization transformation of programs. We consider minimization problem for two formal models of programs: deterministic finite state transducers over finitely generated decidable groups that are used as a model of sequential reactive programs, and deterministic program schemata that are used as a model of sequential imperative programs. Minimization problem for both models of programs can be solved following the same approach that is used for minimization of finite state automata by means of two basic optimizing transformations, namely, removing of useless states and joining equivalent states. The main results of the paper are Theorems 1 and 2.

Theorems 1. If $\boldsymbol{G}$ is a finitely generated group and the word problem in $\boldsymbol{G}$ is decidable in polynomial time then minimization problem for finite state deterministic transducers over $\boldsymbol{G}$ is decidable in polynomial time as well.

Theorem 2. If $\boldsymbol{S}$ is a decidable left-contracted ordered semigroup of basic program statements and the word problem in $\boldsymbol{S}$ is decidable in polynomial time then minimization problem for program schemata operating on the interpretation over $\boldsymbol{S}$ is decidable in polynomial time as well.

Keywords: program equivalence, optimizing transformations, reactive program, program scheme, decision procedure.

DOI: 10.15514/ISPRAS-2015-27(4)-8

For citation: Zakharov V.A., Podymov V.V. On the Application of Equivalence Checking Algorithms for Program Minimization. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 145-174 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-8.

References

[1]. Abel N.E., Bell J.R. Global optimization in compilers. Proceedings of the First USA-Japan Computer Conference. 1972.

[2]. Aho A.V., Sethi R., Ullman J.D. Compilers: principles, techniques, and tools. Addison-Wesley, Reading, MA, 1986.

[3]. Alias, C., Barthou, D. On the recognition of algorithm templates. Proceedings of the 2-nd International Workshop on Compiler Optimization Meets Compiler Verification, Electronic Notes in Theoretical Computer Science, 2004, v. 82, N 2, p. 395–409.

[4]. Alur R., Cerny P. Streaming transducers for algorithmic verification of single-pass list-processing programs. Proceedings of 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2011, p. 599-610.

[5]. Arons T., Elster E., Fix L., Mador-Haim S., Mishaeli M., Shalev J., Singerman E., Tiemeyer A., Vardi M. Y., Zuck L. D. Formal verification of backward compatibility of microcode. CAV, 2005, p. 185–198.

[6]. Allen F.E., Cocke J.L. A catalogue of optimizing transformations. Design and Optimization of Compilers, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[7]. Barthe, G., D'Argenio, P.R., Rezk, T. Secure information flow by self-composition. Proceedings of the 17th IEEE Workshop on Computer Security Foundations, 2004, p. 100.

[8]. Benton, N. Simple relational correctness proofs for static analyses and program transformations. Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2004, ACM, New York.

[9]. Blattner M, Head T. The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences, 1979, v. 19, p. 45-49.

[10]. Busam V.A., England D.E. Optimization of expressions in Fortran. Communications of the Association for Computing Machinary. 1969, v. 12, N 12, p. 666-674.

[11]. Chaki, S., Gurfinkel, A., Strichman, O. Regression verification for multi-threaded programs. Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation, 2012, p. 119-135.

[12]. Christodorescu M., Jha S., Seshia S. A., Song D., Bryant R. E. Semantic-aware malware detection. Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, 2005.

[13]. Dissegna, S., Logozzo, F., Ranzato, F. Tracing compilation by abstract interpretation. Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2014, p. 47-59.

[14]. 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.

[15]. Feng X., Hu A. J. Cutpoints for formal equivalence verification of embedded software. Proceedings of the 5-th ACM International Conference on Embedded Software, 2005, p. 307–316.

[16]. Godlin, B., Strichman, O. Regression verification. Proceedings of the 46th Annual Design Automation Conference, 2009, IEEE Computer Society, Washington.

[17]. Goldberg, B., Zuck, L., Barrett, C. Into the loops: practical issues in translation validation for optimizing compilers. Theoretical Computer Science, 2005, v. 132, N 1, p. 53–71.

[18]. Guo, S.-Y., Palsberg, J. The essence of compiling with traces. Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2011, p. 563-574.

[19]. Kildall G. A unified approach to global program optimization. Conference Record of the Annual ACM SIGPLAN-SIGACT Simposium on Principles of Programming Languages, 1973, p. 194-206.
[20]. Kundu, S., Tatlock, Z., Lerner, S. Proving optimizations correct using parameterized program equivalence. Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2009, p. 327-337.
[21]. Lahiri S. K., Hawblitzel C., Kawaguchi M., Rebelo H. SYMDIFF: A language-agnostic semantic diff tool for imperative programs. Proceedings of the 24th International Conference on Computer Aided Verification, 2012, p. 712–717.
[22]. Le V., Afshari, M., Su, Z. Compiler validation via equivalence modulo inputs. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2014, ACM, New York.
[23]. Lopes N. P., · Monteiro J. Automatic equivalence checking of programs with uninterpreted functions and integer arithmetic. International Journal on Software Tools for Technology Transfer, 2015, N 1, p. 1-16.
[24]. Matsumoto, T., Saito, H., Fujita, M. Equivalence checking of C programs by locally performing symbolic simulation on dependence graphs. Proceedings of the 7-th International Symposiumon Quality Electronic Design, 2006, p. 370-375.
[25]. Mohri M. Finite state transducers in language and speech processing. Computer Linguistics, 1997, v. 23, N 2.
[26]. Mohri M. Minimization algorithms for sequential transducers. Theoretical Computer Science, 2000, v. 234, p. 177-201.
[27]. Namjoshi, K.S., Zuck, L.D. Witnessing program transformations. Proceedings of the 20th International Conference on Static Analysis, 2013, Springer, Berlin, Heidelberg.
[28]. Necula, G.C. Translation validation for an optimizing compiler. Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation. 2000, p. 83-94.
[29]. Nerode A., Kohn W. Models for hybrid systems: automata, topology, controllability, observability. Cornell University, Technical Report 93-28, 1993, MIT Press, Cambridge.
[30]. Pnueli, A., Siegel, M., Singerman, E. Translation validation. Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, 1998, p. 151-166.
[31]. Ramos D. A., Engler D. R. Practical, low-effort equivalence verification of real code. CAV, 2011, p. 669–685.
[32]. Sharma R., Schkufza E., B/ Churchill, A. Aiken. Data-driven equivalence checking. Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications, 2013, p. 391-406.
[33]. Shashidhar, K.C., Bruynooghe,M., Catthoor, F., Janssens, G. Verification of source code transformations by program equivalence checking. Proceedings of the 14th International Conference on Compiler Construction. Springer, 2005, Berlin, Heidelberg.
[34]. de Souza R. On the decidability of the equivalence for k-valued transducers. Proceedings of 12th International Conference on Developments in Language Theory, 2008, p. 252-263.
[35]. Stepp, M., Tate, R., Lerner, S. Equality-based translation validator for LLVM. Proceedings of the 23rd International Conference on Computer Aided Verification, 2011, p. 737-742.
[36]. Terauchi, T.: Aiken, A. Secure information flow as a safety problem. Proceedings of the 12th International Conference on Static Analysis. 2005, p. 352-367.

[37]. Veanes M., Hooimeijer P., Livshits B., et al. Symbolic finite state transducers: algorithms and applications. Proceedings of the 39th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2012.

[38]. Verdoolaege, S., Janssens, G., Bruynooghe, M. Equivalence checking of static affine programs using widening to handle recurrences. Proceedings of the 21st International Conference on Computer Aided Verification, 2009, Springer, Berlin, Heidelberg.

[39]. Yang W., Horowitz S., Reps T. A program integration algorithm that accommodates semantics-preserving transformations. ACM Transactions of Software Engineering and Methodology, 1992, v. 1, N 3, p. 310–354.

[40]. 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.

[41]. Zakharov V.A.. The equivalence problem for computational models: decidable and undecidable cases. Lecture Notes in Computer Science, 2001, v. 2055, p. 133-153.

[42]. Zakharov V.A., Kuzurin N.N., Podlovchenko R.I., Shcherbina V.V. Using algebraic models of programs for detecting metamorphic malwares. Труды Института Системного программирования, Том 12, 2007. с. 77-94.

[43]. Zaks, A., Pnueli, A. CoVaC: Compiler validation by program analysis of the cross-product. In: Proceedings of the 15th International Symposium on Formal Methods. Springer, 2008, Berlin, Heidelberg.

[44]. Zhao, J., Nagarakatte, S., Martin, M.M.K., Zdancewic, S. Formal verification of SSA-based optimizations for LLVM. Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2013, ACM, New York.

[45]. Zuck, L., Pnueli, A., Goldberg, B., Barrett, C., Fang, Y., Hu, Y. Translation and run-time validation of loop transformations. Formal Methods for System Design, 2005, v. 27, N 3, p. 335–360.

[46]. Glushkov V.M. K voprosu o minimizatcii microprogram i skhem algoritmov [On the minimization of microprograms and algorithm schemata]. Kibernetika [Cybernetics]. 1966, N 5, p. 1-3.

[47]. Glushkov V.M., Letichevskii A.A. Teoriya diskretnyh preobrazovateley [Theory of discrete transducers]. Izbranniye voprosy algebry i logiki [Selected issues in algebra and logics]. Nauka Publ., 1973, p. 5-39.

[48]. Zakharov V.A. Bystriye algoritmy razresheniya ekvivalentnosti propozicionalnyh operatornyh program na uporyadochennyh polugruppovyh shkalah [Fast equivalence checking algorithms for propositional sequential programs on the ordered semigroup frames]. Vestnik Moskovskogo Universiteta. Vychislitelnaya matematika i kibernetika [Moscow University Computational Mathematics and Cybernetics] . 1999, N 3. p. 29-35.

[49]. Zakharov V.A. Modelirovanije i analiz posledovatelnyh i reagirujuschih program [Modelling and analysis of sequential reactive programs]. Trudy ISP RAN [The Proceedings of ISP RAS], v. 27, issue 2, 2015, pp. 221-250. DOI: 10.15514/ISPRAS-2015-27(2)-13

[50]. Kasyanov V.N. Optimizirujuschije preobrazovanija program [Optimizing transformations of programs]. Nauka Publ., 1988. 336 p.

[51]. Kotov V.E., Sabelfeld V.K. Teorija shem program [Theory of program schemata]. Nauka Publ., 1991. 348 p.

[52]. Lyapunov A.A. O logicheskih shemah program [On logical program schemata]. Problemy kibernetiki [Problems of Cybernetics], 1958, v. 1, p. 46-74.

[53]. Podlovchenko R.I. Minimization problem for schemes of programs with commutative blocks. Programming and computer software, 2008, v. 34, N 4, c. 237-241.

[54]. Podlovchenko R.I. Polygruppovije modeli program [Semigroup models of programs]. Programmirovanije [Programming and computer software]1981, N 4, c.3-13.

[55]. Podymov V.V., Zakharov V.A. Polynomialnij algoritm proverki ekvivalentnosti program s perestanovochnimi I podavlyaemymi operatorami [A polynomial algorithm for checking the equivalence in models of programs with commutation and vast operators]. Trudy ISP RAN [The Proceedings of ISP RAS], v. 26, N 3, c. 145-166. DOI: 10.15514/ISPRAS-2014-26(3)-8