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 28, выпуск 1

Volume 28, issue 1

Москва 2016

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 областях 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

Методы оптимизации программ на языке JavaScript, основанные на статистике выполнения программы

В.Г. Варданян <vahagvardanyan@gmail.com>
Ереванский государственный университет, 0025, Армения,
г. Ереван, ул. А. Манукяна, дом 1

Аннотация. Язык JavaScript является одним из самых популярных языков для разработки веб-приложений. В связи с ростом производительности персональных компьютеров, мобильных и встраиваемых систем использование JavaScript стало возможным также и в масштабных приложениях. Более того, в настоящее время язык JavaScript активно используется в операционных системах в качестве одного из основных языков для создания пользовательских приложений. Примерами таких систем являются Tizen OS и Firefox OS. С ростом популярности языка многие крупные компании выпустили свои реализации JavaScript, в которых для генерации машинного кода в основном используется многоуровневая динамическая компиляция. В данной работе описываются разработанные методы оптимизации динамических многоуровневых компиляторов с учетом информации о профиле выполнения программы. Метод был реализован в динамическом компиляторе языка JavaScript V8, разработанном компанией Google. Использование профиля выполнения программы позволяет оптимизировать программу для конкретных входных данных. Это особенно актуально в связи с использованием JavaScript в операционных системах. Сценарий использования оптимизации на основе профиля программы в операционных системах следующий: на этапе тестирования программного обеспечения можно организовать сбор информации о профиле программы и использовать его для оптимизации приложений под конкретные случаи выполнения. Одним из новых применений использования информации о профиле программы может быть обеспечение немедленного переключения выполнения часто исполняющихся участков кода на уровень оптимизирующего компилятора. Другое применение – удаление обратных переходов на неоптимизирующие уровни выполнения.

Ключевые слова: JavaScript, V8, оптимизация программ, динамическая компиляция

DOI: 10.15514/ISPRAS-2016-28(1)-1

Для цитирования: Варданян В.Г. Методы оптимизации программ на языке JavaScript, основанные на статистике выполнения программы. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 5-20. DOI: 10.15514/ISPRAS-2016-28(1)-1

1. Введение

В настоящее время широкое распространение получили программы на нетипизированных сценарных языках. Одним из повсеместно используемых языков является JavaScript. С использованием JavaScript написаны многие крупные многофункциональные приложения, такие как Gmail, Google docs и другие. JavaScript также используется в платформе Node.js [1] для разработки веб-приложений на стороне сервера. Более того, уже имеются разработки операционных систем для телефонов, планшетов и ноутбуков, которые подразумевают использование JavaScript как одного из основных языков для создания приложений. Примерами таких систем могут быть Tizen [2] и FirefoxOS [3]. Тем самым, все больше возрастают требования к производительности программ на языке JavaScript, а также к паузам при интерактивном взаимодействии. Многие современные реализации JavaScript используют технологию динамической компиляции (JIT-компиляция), что позволяет применять широкий класс оптимизаций и за счет этого достичь лучшей производительности. При динамической компиляции время, затраченное на компиляцию, добавляется к общему времени выполнения. Поэтому важно соблюдать баланс между сложностью выполняемых оптимизаций и временем задержки запуска программы.

Чтобы достичь такого баланса, используется технология многоуровневой JIT-компиляции. Такое решение обеспечивает быстрый запуск программы, начиная выполнение на неоптимизирующих уровнях компиляции. Далее, наиболее часто исполняющиеся участки кода выполняются на оптимизирующих уровнях компиляции для генерации более качественного машинного кода.

Целью данной работы является разработка и реализация методов оптимизации многоуровневых динамических компиляторов, основанных на профиле выполнения.

Дальнейшее изложение построено следующим образом. В разд. 2 приводится описание архитектуры компилятора V8 и примененные технологии (замена на стеке, спекулятивная компиляция и т.д.) для построения оптимизирующих многоуровневых JIT-компиляторов. В разд. 3 дается обзор работ в предметной области. В разд. 4 описывается схема функционирования предлагаемого решения. В разд. 5 приведены основные результаты.

2. Архитектура V8

Для генерации машинного кода в V8 [4] используются два разных компилятора (рис. 1). Единицей компиляции является функция (метод). Первыми этапами работы V8 являются лексический и синтаксический анализ.

Исходный код разбивается на лексемы, методом рекурсивного спуска строится синтаксическое дерево. После этого начинает работать компилятор первого уровня Full-Codegen. На первом уровне функция переводится в машинный код с выполнением минимального набора оптимизаций, что позволяет быстрее приступить к выполнению кода. При генерации машинного кода для каждой инструкции учитываются все возможные случаи выполнения для данной операции. На этом уровне собирается профиль программы – информация о типах полей объектов. Кроме того, базовый компилятор V8 расставляет счетчики для определения часто выполняемых участков кода (функций и циклов). Когда такой участок кода обнаруживается, компиляция переходит на второй, оптимизирующий уровень – Crankshaft. На этом уровне из абстрактного синтаксического дерева строится граф потока управления в SSA-представлении – Hydrogen. Это внутреннее представление позволяет выполнить ряд машинно-независимых оптимизаций, таких как встраивание функций, удаление мертвого кода, удаление общих подвыражений, оптимизации циклов и т.д.

Crankshaft использует собранную в предыдущем уровне информацию о типах для эффективного хранения целочисленных переменных и операций над ними. На 64-битных архитектурах для целых чисел используется следующее кодирование: старшие 32 бита хранят число, младшие биты нулевые. На 32-битных архитектурах для хранения целых чисел используется 31 бита, младший бит нулевой. Целые числа в таком представлении называются Smi (от английского small integer). В JavaScript отсутствует строгая типизация. Это значит, что в общем случае переменные ссылаются на объекты. Все указатели на объекты в V8 имеют младший бит (определяющий четность числа) установленным в единицу, что позволяет отличить Smi от других объектов. Кодирование использует тот факт, что все адреса выровнены, т.е. объектов с нечетными адресами не существует.

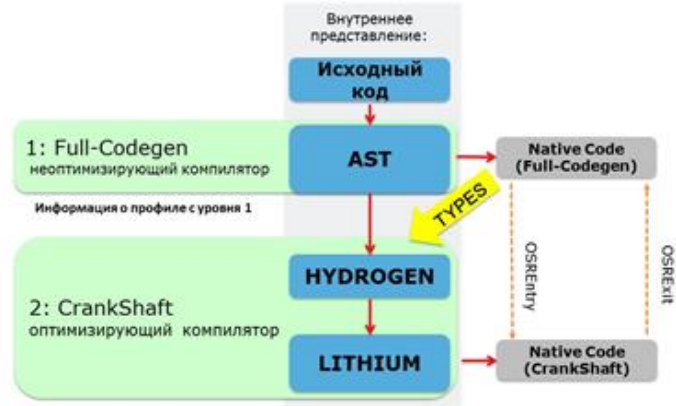


Рис. 1. Многоуровневая архитектура компилятора V8
Fig.1. Multilevel architecture of the V8 compiler

Crankshaft распространяет полученную из уровня Full-Codegen информацию о профиле по всему графу. Это позволяет генерировать машинный код спекулятивным образом, основываясь на предположении, что определенные свойства переменных (тип, значение и т.д.) останутся неизменными при следующих вызовах функции. В код вставляются необходимые проверки. Когда одна из таких проверок не выполняется, происходит переход на первый, неоптимизированный уровень. Этот процесс называется деоптимизацией. Для переключения между разными уровнями компиляции используется технология замены на стеке (OSR) [5]: выполнение функции приостанавливается и текущий стек функции заменяется новым. После выполнения “замены на стеке”, во всех местах вызова этой функции производится перенаправление на новую версию функции. При этом, переключение может произойти как с первого уровня на второй (OSR entry), так и наоборот (OSR exit).

После выполнения всех оптимизаций представление Hydrogen переводится в машинно-зависимое представление – Lithium. В отличие от представления Hydrogen, Lithium использует близкое к машинному коду трехадресное представление. Это представление используется для эффективной организации распределения регистров, а также для кодогенерации.

3. Обзор работ

Существуют два подхода для сбора информации о профиле программы – динамический и статический. Статический метод основан на алгоритмическом анализе и часто используется во время компиляции программы для эффективной реализации той или иной оптимизации. Например, оценка приблизительного количества выполнения функций или циклов позволяет более эффективно реализовать распределение регистров.

При использовании динамического метода, информация собирается во время выполнения программы. По сравнению с статическом профилированием, динамический метод позволяет собирать более точную информацию. Существует два способа получения статистики во время исполнения программы: инструментировать ее, - то есть вставлять счетчики в код при генерации или еще в промежуточное представление программы, - либо собирать выборочным профилировщиком аппаратных прерываний. В первом случае код увеличивается в размерах и сильно замедляется работа программы [6]. Во-втором же случае влияние на время выполнения гораздо меньше.

Профилирование поддерживается во многих индустриальных компиляторах. Например, в компиляторной инфраструктуре LLVM [7] реализованы три метода профилирования: на уровне функций, базовых блоков или ребер в промежуточном представлении. Все методы могут быть использованы независимо друг от друга. Компилятор GCC [8] также поддерживает несколько разных методов профилирования.

Много новых работ посвящены к улучшению производительности программ, написанных на языках с динамическими типами. В работах [9] [10] [11] описывается методы предварительной оптимизации программ на языке JavaScript. В работе [12] описываются методы компиляции программ с динамическими типами в статическое внутренне представление LLVM. Данный метод позволяет применить реализованные в LLVM оптимизации к программам, написанным на языке JavaScript. В работах [13] [14] [15] [16] [17] приведены разные методы оптимизации динамических многоуровневых компиляторов.

В данной работе описываются особенности оптимизации многоуровневых динамических компиляторов с использованием информации о статистке выполнения программы на примере компилятора V8.

4. Особенности использования профиля выполнения программы в многоуровневых динамических компиляторах

В целях улучшения производительности многоуровневых JIT-компиляторов важно обеспечить, чтобы часто выполняемые участки программы как можно больше времени выполнялась на оптимизирующих уровнях компиляции. Для обоснования этого утверждения приведем анализ сравнения времени выполнения разных уровней компилятора V8 на нескольких тестовых наборах языка JavaScript (рис 2а и 2б).

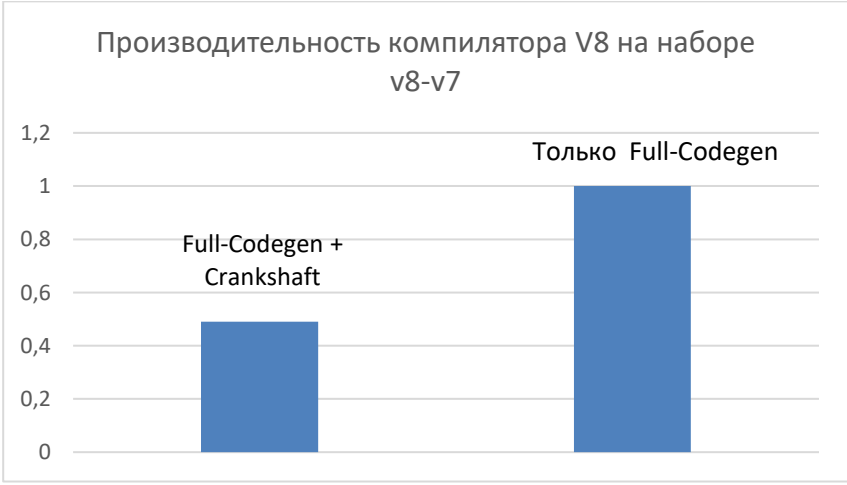


Рис 2а. Сравнение производительности уровней V8 на наборе v8-v7. Единицей времени выступает время выполнения компилятора Full-Codegen
Fig 2a. Comparing performance of the V8 levels on the benchmark v8-v7. The unit of time is duration of run-time of the Full-Codegen compiler

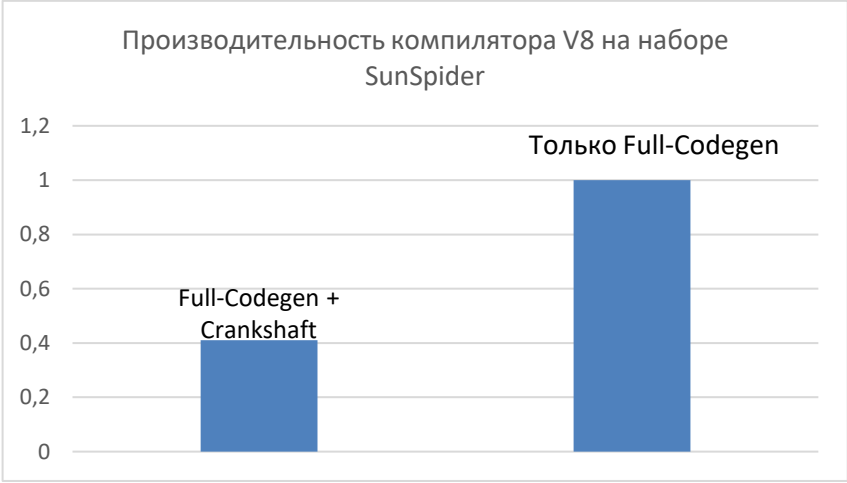


Рис 2б. Сравнение производительности уровней V8 на наборе SunSpider. Единицей времени выступает время выполнения компилятора Full-Codegen.
Fig 2b. Comparing performance of the V8 levels on the benchmark of SunSpider. The unit of time is duration of run-time of the Full-Codegen compiler

Из анализа сравнения видно, что производительность оптимизирующих уровней компиляции в среднем 2-2.5 больше, нежели производительность неоптимизирующих уровней. Поэтому, в целях улучшения производительности компилятора важно добиться, чтобы горячие участки кода как можно больше времени выполнялись именно на оптимизирующих уровнях компиляции. Одним способом достижения этой цели является как можно быстрое переключение выполнения таких участков на оптимизирующие уровни компиляции.

Другим способом является устранение обратных переходов на неоптимизирующие уровни выполнения и последующих перекомпиляций функций.

4.1 Использование профиля выполнения программы для обеспечения более быстрого перехода на оптимизирующие уровни компиляции

Более быстрого перехода на оптимизирующие уровни выполнения можно достичь путем внедрения в компилятор механизмов, позволяющих собирать и сохранять информацию о горячих участках кода. Многоуровневая архитектура компилятора V8 уже содержит механизмы для нахождения часто выполняемых участков кода, следовательно, сбор информации не добавляет никаких дополнительных накладных расходов. В компилятор V8 был добавлен модуль для сохранения этой информации. Далее, при последующих запусках программы сохраненная информация о горячих участках кода

позволяет сразу перевести выполнение таких участков на оптимизирующие уровни компиляции. При этом необходимо учитывать тот факт, что на первых уровнях выполнения собирается необходимая для спекулятивной компиляции информация о типах и объектах программы, что ограничивает немедленный переход на оптимизирующие уровни выполнения.

Была реализована новая эвристика для переключения выполнения горячих функций на оптимизирующие уровни компиляции с учетом сохраненной информации. Переход осуществляется, как только будет собран необходимый процент информации о типах (25% по умолчанию) для функции. Этот метод особенно эффективен для программ с небольшим временем выполнения. Для таких программ часто характерна следующая ситуация: функция помечается как кандидат для выполнения на оптимизирующем уровне. Компиляция этой функции на оптимизирующем уровне производится параллельно с выполнением программы на нижних уровнях, и программа заканчивается раньше, чем начинается выполнение оптимизированной версии функции. Реализованный метод позволяет переключать выполнение на оптимизирующий уровень намного раньше, что приводит к значительному росту производительности для программ с небольшим временем выполнения.

4.2 Использование профиля программы для устранения обратных переходов на неоптимизирующие уровни выполнения.

Из-за динамических свойств языка JavaScript в реальных приложениях деоптимизации могут встречается часто. В компиляторе V8 при выполнении деоптимизации для функции собирается новый профиль для организации реоптимизации. Для исключения больших временных затрат на постоянную реоптимизацию кода, при множественных деоптимизаций оптимизация для данной функции запрещается. В компилятор V8 был добавлен модуль для сохранения информации о количестве и причинах деоптимизаций. При последующих запусках программы эта информация используется следующим образом:

- Если при первом запуске программы для какой-либо функции оптимизация была запрещена из-за множественных деоптимизаций, при последующих запусках не будут предприниматься попытки для ее оптимизации.
- Если при первом запуске программы в какой-либо функции происходит деоптимизация из-за недостаточной информации о типах, переключение выполнения этой функции на оптимизирующие уровни откладывается, что позволяет собрать более полную информацию о типах (50% по умолчанию).

4.3 Использование профиля программы для выбора оптимального набора оптимизаций.

На разных приложениях эффект конкретных оптимизаций на сгенерированный машинный код может быть незначительным. При динамической компиляции такие оптимизации могут стать причиной ухудшения производительности для конкретных приложений. Приведем анализ эффективности реализованных в компиляторе V8 оптимизаций на тестовом наборе v8-v7 и на реальном примере использования сайта Gmail. Более подробный анализ и тестирования на других приложениях (Facebook, Wordpress) можно найти в [18]. Для имитации реального использования сайта Gmail была использована платформа Sikuli UI [19], которая позволяет автоматизировать использование графического интерфейса. Сценарий использования сайта Gmail следующий: программа авторизуется в Gmail используя данные тестового пользователя. Затем загружаются все входящие сообщения и производится поиск нескольких сообщений. В конце программа выходит из системы. Запуск тестового набора v8-v7 также был произведен из браузера с помощью Sikuli UI.Для оценки влияния каждой оптимизации на производительность в целом были проведены тестирования с выключением около одиннадцати оптимизаций: нумерация глобальных значений, вынос инвариантного кода за цикл, анализ диапазонов, удаление лишних Ф-функций, встраивание вызовов функций, встраивание полиморфных вызовов функций и с последовательным включением каждой из этих оптимизаций. Результаты тестирований на наборе v8-v7 и сайте Gmail приведены на Рис. 3а и 3б.

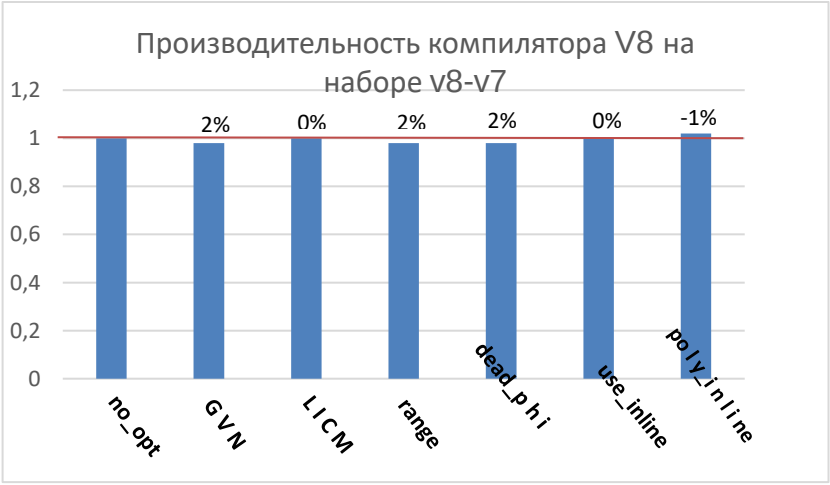


Рис 3а. Относительное время выполнения V8 на наборе v8-v7. Единицей времени выступает время выполнения компилятора с выключенными оптимизациями
Fig 3a. Relative duration of the V8 run-time on the benchmark v8-v7. The unit of time is duration of the compiler run-time with turned-off optimizations

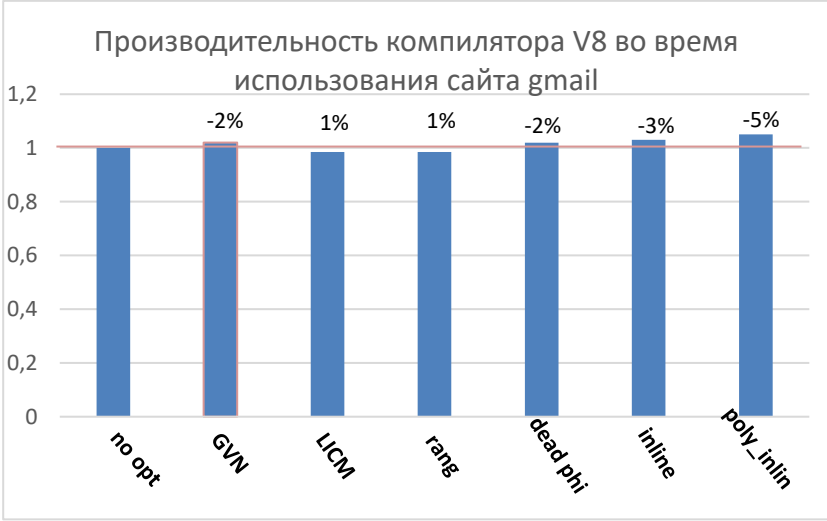


Рис 3б. Относительное время выполнения V8 при использовании сайта gmail. Единицей времени выступает время выполнения компилятора с выключенными оптимизациями
Fig 3b. Relative duration of the V8 run-time with the gmail site. The unit of time is duration of the compiler run-time with turned-off optimizations

Как видно из результатов, на наборе v8-v7 большинство оптимизаций имеет положительный эффект на время выполнения, однако оптимизация “встраивание полиморфных вызовов” в целом ухудшает производительность на 1%. На реальном примере использования сайта Gmail, наоборот, большинство протестированных оптимизаций имеет негативный эффект на производительность. Оптимизация глобальной нумерации значений для удаления общих подвыражений ухудшает производительность на 2%, а встраивание полиморфных вызовов на 5%.

Сохраненная информация о эффективности каждой оптимизации позволяет выбрать оптимальный набор оптимизаций для каждого приложения. В компиляторе V8 были добавлены соответствующие счетчики для оценки эффективности каждой оптимизации (вычисляется количество удаленных/измененных оптимизацией инструкций, а также количество итераций циклов функций). Кроме того, был реализован модуль для сохранения собранной информации. Была добавлена поддержка для следующих оптимизаций:

- Нумерация глобальных значений, для удаления общих подвыражений
- Вынос инвариантного кода за цикл
- Удаление мертвого кода
- Анализ диапазонов

- Удаление избыточных проверок
- Удаление избыточных Ф-функций
- Удаление избыточных обращений к памяти (load/store elimination)
- Escape анализ, для оптимизации хранения объектов в куче

Для оценки необходимости использования той или иной оптимизации была реализована метрика, которая зависит от нескольких параметров. Эффективность оптимизации вычисляется следующей формулой:

$$\mathbf{E = \sum_i (FC * LCi),\ i = 0, 1, \dots n}$$

Где n количество удаленных (измененных) оптимизацией инструкций, FC — количество выполнения функции, LCi — количество итераций цикла, в котором находилась удаленная (измененная) инструкция. Вычисляется также примерное количество шагов, необходимых для выполнения каждой из этих оптимизаций (OptSteps). Если для данной оптимизации E < OptSteps, она отключается при последующих запусках программы. Например, оптимизация “удаление мертвого кода”, выполняется за два прохода по всем инструкциям графа управления. При первом проходе отмечаются все живые переменные. При этом для каждой инструкции проверяются несколько условий. При первом проходе выполняются приблизительно 2*m шагов, где m количество инструкций в графе. Во время второго прохода непомеченные инструкции удаляются (не больше чем за m шагов). Сама оптимизация вызывается два раза для каждой функции. Значение OptSteps для этой оптимизации равно m*2*(2+1) = 6*m, где m количество инструкций в графе потока управления. Такая оценка была реализована для каждой из рассматриваемых оптимизаций.

Важно также отметить, что выполнение некоторых оптимизаций может повлиять на эффективность других. Например, анализ диапазонов используется при удалении избыточных проверок переполнения. В таких случаях, для оценки эффективности также учитывается возможный эффект на другие оптимизации.

Количество выполнений циклов и функций используется также для выбора методов распределения регистров. Так для функций с коротким временем выполнения используется обыкновенный алгоритм линейного сканирования [20], а для больших функций с тяжелыми циклами используется жадный алгоритм линейного сканирования [21].

5. Результаты

Использование информации о часто выполняемых участках кода и количестве деоптимизаций не добавляет дополнительных накладных расходов при сборе информации, так как инструментарии уже были реализованы в компиляторе V8. Внедрение инструментарий для вычисления количества итераций циклов

и эффективности оптимизаций замедляет набор SunSpider всего на 5%. Набор Octane замедляется на 23%, а Kraken на 25%.

Использование собранной информации позволяет достичь существенного роста производительности при последующих запусках программы. Организация более быстрого перехода на оптимизирующие уровни выполнения, а также реализация метода выбора оптимального набора оптимизаций для конкретных приложений позволило ускорить множество тестов из наборов SunSpider и Kraken. В среднем тестовый набор SunSpider стал выполняться на 11% быстрее, ускорение конкретных тестов составляет 50% (Таблица 1). Тестовый набор Kraken в среднем ускорился на 4%. На наборе Octane тесты *deltablue* и *crypto* ускорились на 7%, тест *richards* на 5% а *gameboy* на 3%. Пример использования сайта gmail ускорился на 1%.

Таблица 1. Производительности набора SunSpider с использованием оптимизаций на основе профиля программы (меньше - лучше)
Table 1. Performance of the SunSpider benchmark using optimizations based on the profile of the program (less – better)

Тест	Производительность V8, мс	Производительность V8 с реализованными улучшениями, мс	Улучшение, %
3d-cube	48.8	46	5.73
3d-morph	51.7	52.4	-
3d-raytrace	52.4	53	-1.1
access-binary-tree	7.1	7.1	-
access-fannkuch	20.7	20.7	-
access-nbody	13.5	13.1	3
access-nsieve	8.8	7.2	18
bitops-3bit-in-byte	4.0	3.6	10
bitops-bits-in-byte	11.8	11.2	5
bitops-bitwise-and	8.1	8.1	-
bitops-nsieve-bits	11.9	12	-
control-flow-recursive	6.0	6.0	-
crypto-aes	26.2	23.6	9.9
crypto-md5	17.8	17.5	1.7

crypto-sha1	17.8	19.0	-6.7
date-format-tofte	48.7	49.3	-1.2
date-format-xparb	70.1	71.0	-1.2
math-cordic	13.4	13.1	2.2
math-partial-sum	33.0	32.7	-
math-spectral-norm	11	7.2	34
regexp-dna	32.2	32.2	-
string-base64	27.0	26.8	-
string-fasta	70.7	35.4	50
string-tagcloud	101.0	101.0	-
string-unpack-code	91.4	93.1	-1.8
string-validate-input	40.6	30.1	25.8
Суммарное время	680.3	602.2	11.4

6. Заключение

В рамках данной работы был реализован метод оптимизации многоуровневого JIT-компилятора V8, основанный на статистике выполнения программы. При этом по возможности были использованы реализованные в компиляторе инструментарии, что позволило минимизировать накладные расходы при сборе необходимой информации. Собранная статистика выполнения программы позволяет значительно улучшить производительность компилятора V8.

Список литературы

[1]. Страница платфомы Node.js: nodejs.org.
[2]. Страница платформы Tizen: tizen.org.
[3]. Страница платофомы FirefoxOS: www.mozilla.org.
[4]. Страница компилятора V8: https://developers.google.com/v8/.
[5]. S. J. Fink и F. Qian, «Design, Implementation, and Evaluation of Adaptive Recompilation with on-stack replacement» *Proceedings of the IEEE,* pp. 241-252, 2003.
[6]. T. Ball и J. R. Larus, «Thomas Ball and James R. Larus. Optimally profiling and tracing programs» *In POPL* '92: Proceedings of the 19th ACM SIGPLAN-SIGACT

symposium on Principles, pp. 59-70, 1992.
[7]. Страница инфраструктуры LLVM: LLVM.org.
[8]. Страница компилятора GCC: https://gcc.gnu.org/.
[9]. R. Zhuykov, V. Vardanyan, D. Melnik, R. Buchatskiy и E. Sharygin, «Augmenting JavaScript JIT with Ahead-of-Time Compilation,» In Proceedings of IEEE,Computer Science and Information Technologies, pp. 116-120, 2015.
[10]. R. Zhuykov, V. Vardanyan, D. Melnik, R. Buchatskiy и E. Sharygin, «Augmenting JavaScript JIT with Ahead-of-Time Compilation,» 10th International Conference on Computer Science and Information Technologies, pp. 236-240, 2015.
[11]. S. Jeon и J. Choi, «Reuse of JIT compiled code based on binary code patching in JavaScript engine,» J. Web Eng, pp. 337-349, 2012.
[12]. В. Варданян, В. Иванишин, С. Асрян, А. Хачатрян и Д. Акопян, «Динамическая компиляция программ на языке JavaScript в статически типизированное внутреннее представление LLVM». Труды Института системного программирования РАН, том 27 (выпуск 6), 2015 г., стр. 33-48. DOI: 10.15514/ISPRAS-2015-27(6)-3
[13]. V. Vardanyan, «Optimizations of JavaScript programs,» GSPI's scientific journal 2014, pp. 122-128.
[14]. Р. Жуйков, Д. Мельник, Р. Бучацкий, В. Варданян, В. Иванишин и Е. Шарыгин, «Методы динамической и предварительной оптимизации программ на языке JavaScript,» Труды Института системного программирования РАН, том 26 (выпуск 1), 2014 г., стр. 297-314. DOI: 10.15514/ISPRAS-2014-26(1)-10
[15]. S.-W. Lee и S.-M. Moon, «Selective just-in-time compilation for client-side mobile javascript engine,» Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems, pp. 5-14, 2011.
[16]. S.-W. Lee, S.-M. Moon, W.-K. Jung, J.-S. Oh и H.-S. Oh, «Code size and performance optimization for mobile JavaScript just-in-time compiler,» Proceedings of the 2010 Workshop on Interaction between Compilers and Computer Architecture, 2010.
[17]. V. Vardanyan, S. Asryan и R. Buchatskiy, «Integrated register rematerialization in JavaScript V8 JIT compiler,» 10th International Conference on Computer Science and Information Technologies, pp. 240-244, 2015.
[18]. Анализ эффективности оптимизаций в компиляторе V8: http://www.cs.cmu.edu/~ishafer/compilers/.
[19]. Страница платформы Sikuli UI: http://www.sikuli.org/.
[20]. M. Poletto и V. Sarkar, «Linear scan register allocaton» ACM Transactions on Programming Languages and Systems, pp. 895-913 , 1999.
[21]. Описаные жадного алгоритма линейного сканирования: http://blog.llvm.org/2011/09/greedy-register-allocation-in-llvm-30.html.

Profile-based optimizations for JavaScript programs

V. Vardanyan <vaag@ispras.ru>,
Yerevan State University
Alex Manoogian, 1, 0025, Yerevan, Republic of Armenia

Abstract. In recent years, JavaScript has become one of the most popular programming languages on the web. Many massive applications are written using JavaScript, such as Gmail, Google docs, etc. JavaScript is also used in the Node.js — server-side web application developing platform. Moreover, JavaScript is the main language for developing applications on some operating systems for mobile and media devices. Examples of such systems are Tizen and FirefoxOS. Due to increasing popularity of JavaScript many big companies produced and continue to develop their own dynamic compilers for this language. JIT compilation makes it possible to implement many well-known classic optimizations to improve programs performance. To maintain a trade-off between quick startup and doing sophisticated optimizations, JavaScript engines usually use multiple tiers for compiling hot functions: lower tier JITs generate less efficient code, but can start almost immediately (e.g., even with interpretation), while higher tier JITs aim at generating very effective code for hot places, but at the cost of long compilation time. So even highly optimized JavaScript execution engines require some time to "warm-up" before reaching their peak performance. This work is dedicated to the performance improvement of modern dynamic multitier JIT compilers by designing and implementing profile-based optimizations in JavaScript V8 JIT compiler.

Keywords: JavaScript, V8, program optimization, dynamic compilation.

DOI: 10.15514/ISPRAS-2016-28(1)-1

For citation: V. Vardanyan. Profile-based optimizations for JavaScript programs. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 1, 2016, pp. 5-20 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-1

References

[1]. Node.js website: nodejs.org.
[2]. Tizen website: tizen.org.
[3]. FirefoxOS website: www.mozilla.org.
[4]. V8 compiler website: https://developers.google.com/v8/.
[5]. S. J. Fink и F. Qian, «Design, Implementation, and Evaluation of Adaptive Recompilation with on-stack replacement» Proceedings of the IEEE, pp. 241-252, 2003.
[6]. T. Ball и J. R. Larus, «Thomas Ball and James R. Larus. Optimally profiling and tracing programs» In POPL ’92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles, pp. 59-70, 1992.
[7]. LLVM website: http://LLVM.org.
[8]. GCC website: https://gcc.gnu.org/
[9]. R. Zhuykov, V. Vardanyan, D. Melnik, R. Buchatskiy и E. Sharygin, «Augmenting JavaScript JIT with Ahead-of-Time Compilation,» In Proceedings of IEEE,Computer Science and Information Technologies, pp. 116-120, 2015.
[10]. R. Zhuykov, V. Vardanyan, D. Melnik, R. Buchatskiy и E. Sharygin, «Augmenting JavaScript JIT with Ahead-of-Time Compilation,» 10th International Conference on

Computer Science and Information Technologies, pp. 236-240, 2015.
[11]. S. Jeon и J. Choi, «Reuse of JIT compiled code based on binary code patching in JavaScript engine,» J. Web Eng, pp. 337-349, 2012.
[12]. V. Vardanyan, V. Ivanishin, S. Asryan, A. Khachatryan, J. Hakobyan, «Dinamicheskaja kompiljacija programm na jazyke JavaScript v staticheski tipizirovannoe vnutrennee predstavlenie LLVM» («Dynamic Compilation of JavaScript Programs to the Statically Typed LLVM Intermediate Representation»). Trudy ISP RAN [Proceedings of ISP RAS], Volume 27 (Issue 6), 2015. pp. 33-48 (in Russian). DOI: 10.15514/ISPRAS-2015-27(6)-3
[13]. V. Vardanyan, «Optimizations of JavaScript programs,» GSPI's scientific journal 2014, pp. 122-128.
[14]. R. Zhuykov, D. Melnik, R. Buchatskiy, V. Vardanyan, V. Ivanishin и E. Sharygin, «Metody dinamicheskoj i predvaritel'"noj optimizacii programm na jazyke JavaScript» («Dynamic and ahead of time optimization for JavaScript programs») Trudy ISP RAN [Proceedings of ISP RAS], Volume 26, (Issue 1), 2014. pp. 297-314 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-10
[15]. S.-W. Lee и S.-M. Moon, «Selective just-in-time compilation for client-side mobile javascript engine,» Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems, pp. 5-14, 2011.
[16]. S.-W. Lee, S.-M. Moon, W.-K. Jung, J.-S. Oh и H.-S. Oh, «Code size and performance optimization for mobile JavaScript just-in-time compiler,» Proceedings of the 2010 Workshop on Interaction between Compilers and Computer Architecture, 2010.
[17]. V. Vardanyan, S. Asryan и R. Buchatskiy, «Integrated register rematerialization in JavaScript V8 JIT compiler,» 10th International Conference on Computer Science and Information Technologies, pp. 240-244, 2015.
[18]. Quantifying Optimization Efficacy in V8 JIT compiler: http://www.cs.cmu.edu/~ishafer/compilers/
[19]. Sikuli UI website: http://www.sikuli.org/.
[20]. M. Poletto и V. Sarkar, «Linear scan register allocaton» ACM Transactions on Programming Languages and Systems, pp. 895-913 , 1999.
[21]. Description of greedy linear scan algorithm: http://blog.llvm.org/2011/09/greedy-register-allocation-in-llvm-30.html

Инфраструктура статического анализа программ на языке C#

В. К. Кошелев <vedun@ispras.ru>
В. Н. Игнатьев <valery.ignatyev@ispras.ru>
А. И. Борзилов <helendile@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25.

Аннотация. В работе рассмотрены различные аспекты статического анализа программ на языке C# с целью обнаружения максимального количества ошибок за минимально приемлемое время. Описан полный цикл статического анализа программного обеспечения, при этом основное внимание уделяется особенностям, возникающим при анализе языка C#. Рассмотрены методы, позволяющие учитывать популярные возможности языка на всех уровнях анализа: построения графа вызовов и графа потока управления, проведение анализа потоков данных и чувствительного к контексту и путям межпроцедурного анализа. Предлагается метод символьного исполнения, основанный на таких работах, как Bounded Model Checking и Saturn Software Analysis Project. В статье описана организация модели памяти, позволяющая как проводить точный внутрипроцедурный анализ, так и создавать компактные представления для привязанных к функциям условий, используемые при межпроцедурном анализе. Особое внимание уделяется теме оптимизации возникающих на этапе чувствительного к путям анализа условий. Условия необходимо оптимизировать как по размеру, поскольку при межпроцедурном чувствительном к путям анализе необходимо сохранять большое количество условий для каждой проанализированной функции, так и по сложности, поскольку время анализа ограничено. Решение условий производится при помощи современных SMT-решателей, таких как Microsoft Z3 Prover. В статье также рассмотрены различные подходы к моделированию поведения библиотечных функций: при помощи резюме в виде набора признаком или в виде упрощенных реализаций на языке C#. Все приведённые решения реализованы в инструменте статического анализа SharpChecker и протестированы на наборе проектов различного объема (от 1.5 тыс. до 1.35 млн. строк кода) с открытым исходным кодом.

Ключевые слова: Статический анализ; использование нулевого указателя; чувствительность к путям; чувствительность к контексту; резюме функции; поиск дефектов; Roslyn; C#

DOI: 10.15514/ISPRAS-2016-28(1)-2

Для цитирования: Кошелев В. К., Игнатьев В. Н., Борзилов А. И. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40. DOI: 10.15514/ISPRAS-2016-28(1)-2

1. Введение

Статический анализ программ в последние годы занимает важную роль при разработке программного обеспечения, позволяя обнаруживать ошибки в ПО без фактического исполнения программы. За счёт использования все более сложных методов анализа появляется возможность повысить качество результатов, обнаруживая новые ошибки и снижая относительное количество ложных срабатываний.

В настоящее время язык программирования C# достиг высокой популярности и, согласно индексу TIOBE [1], занимает четвертую позицию в рейтинге распространённости. Благодаря хорошей архитектуре, богатому выбору доступных библиотек и хорошо поддерживаемой инфраструктуре программирования все больше промышленного и открытого ПО разрабатывается с использованием этого языка. Это означает, что к программам предъявляются высокие требования по качеству как результирующей программы, так и исходного текста. Для соответствия таким стандартам качества существуют различные методы, одним из которых является использование инструментов статического анализа исходного текста. Учитывая тот факт, что C# не используется при разработке низкоуровневого ПО, а большая часть существующих проектов на C# имеет существенно больший размер, чем драйверы операционной системы или встраиваемое в аппаратуру ПО, задача формальной верификации с помощью статического анализа не является актуальной для C#. В то же время, задача обнаружения максимального количества ошибок за минимально возможное время является очень востребованной.

Специфика разрабатываемых на рассматриваемом языке проектов определяет список наиболее важных типов ошибок, которые могут приводить к отказам ПО или даже позволят злоумышленнику эксплуатировать найденные уязвимости.

Основные проблемы могут возникать из-за:

- использования null;
- утечки ресурсов (памяти или, например, файловых дескрипторов), полученных при взаимодействии с unmanaged кодом;
- использования уже освобождённых unmanaged объектов;
- попадания пользовательских данных в базы данных или пользовательский интерфейс без корректной проверки (SQL Injection, XSS и т.д.)
- ошибок, связанных с явным приведением типов.

Как несложно заметить, список важных проблем отличается от соответствующего списка для языков C/C++. Например, очень важной проблемой для C++ является возможность доступа к данным за пределами отведённого буфера, которая не является актуальной для C# по двум причинам. Во-первых, при доступе за границы массива возникнет

соответствующее исключения, и эксплуатировать такую уязвимость с целью манипуляции чужими данными невозможно. Во-вторых, наличие встроенных в язык средств работы с коллекциями привело к тому, что доступ по индексу практически не используется в реальном ПО, а реализуется с помощью итераторов и Linq [2].

Большинство разработанных на данный момент инструментов статического анализа создавались для языков C/C++ и Java и не могут без серьёзных изменений хорошо работать с программами на C#. Постоянное использование исключений, делегатов, Linq, вызовов через интерфейсы, свойств классов, которые автоматически вызывают геттеры и сеттеры, концепция Disposable объектов для взаимодействия с unmanaged окружением, встроенные синтаксические конструкции и одновременно библиотечные методы для организации блокировок не позволяют без существенных доработок применять существующие инструменты анализа. Поэтому в данной работе подробно рассмотрены практические приёмы, позволяющие учитывать особенности языка на всех уровнях анализа: начиная с построения графа потока управления и графа вызовов, заканчивая анализом условий для достижения чувствительности к контексту вызова и путям выполнения.

Существующие инструменты статического анализа C# промышленного уровня можно разделить на две группы: поддерживающие контекстно- и потоково-чувствительный анализ (разрабатываемый в ИСП РАН инструмент Svace [3], инструменты от Coverity [4], Klocwork [5]), и основанные на абстрактном синтаксическом дереве (АСД) (инструменты от SonarLint [6], СиПроВер [7]). Продукты из первой группы имеют очень высокую стоимость и применяются в основном в крупных компаниях, а легковесные анализаторы из второй группы гораздо более доступны, однако принципиально не могут обнаружить большой класс ошибок.

Приведённые аргументы свидетельствуют о необходимости адаптации алгоритмов статического анализа, созданных для C++ и Java, для использования в анализе C#. Кроме того, задача разработки контексто- и потоково- чувствительного инструмента статического анализа является очень актуальной, поскольку на данный момент практически отсутствуют доступные аналогичные инструменты.

Статья организована следующим образом. В части 2 показана общая схема проведения анализа, которая в соответствии с этапами анализа уточняется в последующих частях. Так, в 3 части рассмотрены вопросы построения графа вызовов, в 4 – особенности графа потока управления. Части 5-7 описывают схему организации чувствительного к путям и контексту межпроцедурного анализа. В восьмом разделе приведена оценка результатов тестирования инструмента SharpChecker [8], реализующего все рассмотренные подходы. В заключении подведены итоги работы.

2. Схема работы анализатора

Разработанный инструмент статического анализа SharpChecker, описанный в данной работе, способен находить 30 различных типов ошибок, среди которых есть как использующие исключительно синтаксический анализ, так и анализ потоков данных, а также наиболее мощный чувствительный к контексту и путям выполнения межпроцедурный анализ.

Для понимания особенностей обработки специфичных возможностей языка C# предварительно рассмотрим общую схему работы анализатора, представленную на рисунке 1.

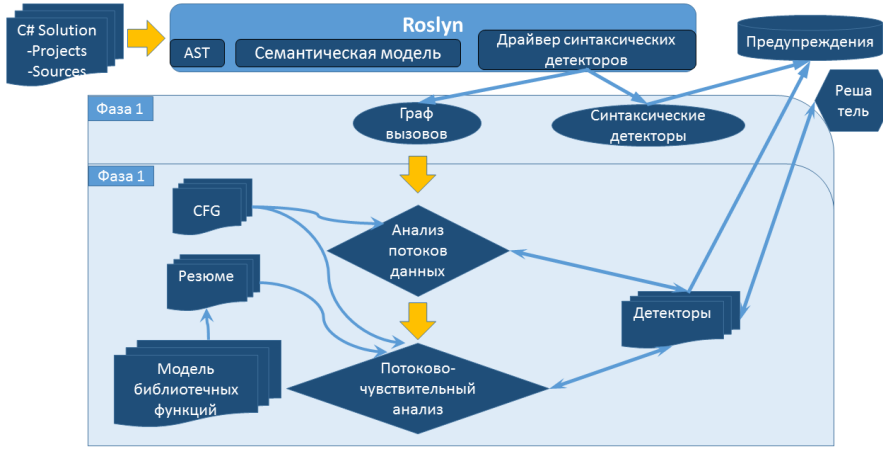


Рис. 1. Схема работы анализатора SharpChecker
Fig. 1. Operational scheme of the SharpChecker analyzer

Большинство проектов на языке C# используют для организации сборки механизмы, основанные на файлах проектов и решений, предоставляемые Microsoft Visual Studio [9]. Разработанный анализатор использует инфраструктуру Roslyn [10] для работы с файлами системы сборки, а также для компиляции исходного текста программы. Roslyn – это открытая компиляторная платформа, поддерживающая набор механизмов для разработки статических анализаторов. Инструмент SharpChecker использует только часть Roslyn, отвечающую за разбор файлов проектов, решений, в результате которой определяется множество файлов, используемое для сборки заданной программы или библиотеки, а также необходимое окружение. Кроме этого, Roslyn использован для построения абстрактного синтаксического дерева и таблицы символов каждого модуля компиляции. Для достижения хорошего качества результатов статический анализатор обязан учитывать правила сборки, определяющие, в первую очередь, правила для компоновщика, или, другими словами, граф вызовов. Таким образом, на

первом этапе с помощью Roslyn выполняется построение абстрактного синтаксического дерева для всех файлов, участвующих в сборке, производится анализ всех возможных вызовов, в том числе неявных, строится иерархия наследования классов и собирается другая информация, необходимая для построения статического графа вызовов. Одновременно, на этом же этапе выполняется поиск синтаксических ошибок, для обнаружения которых достаточно АСД. Второй этап реализует построение графа вызовов программы на основе собранных данных, а затем уточняет его на основе результатов анализа виртуальных функций и интерфейсов. Особенности построения графа вызовов подробнее описаны в части 3.

Третий этап состоит в обходе графа вызовов в обратном топологическом порядке от вызываемой функции к вызывающей. При этом возможные циклы в графе вызовов разрываются в произвольном месте. В течение этой стадии анализа выполняются как анализ потоков данных и использующие его детекторы, так и потоково-чувствительные детекторы. Кроме того, часть собранной информации сохраняется в резюме функции и используется при анализе функций, вызывающих данную. Обход функций производится параллельно, что позволяет существенно сократить время работы на многоядерных машинах.

Следующий этап реализует проверку большинства правил, а также накопление информации о результатах анализа функций в резюме для последующего использования в вызывающей функции.

Таким образом, в процессе обхода графа вызовов по предоставленным Roslyn АСД и таблице символов осуществляется построение графа потока управления (ГПУ) для каждой функции. При этом анализатор использует два представления ГПУ для каждой функции, различающиеся обработкой лямбда-функций. Особенности построения ГПУ для специфичных конструкций языка C# рассмотрены в части 4.

Следующий этап реализует классический анализ потоков данных, на основе которого работают детекторы, например, поиск неиспользованных значений, включающий как неиспользуемые переменные, так и результаты вычислений.

При этом происходит обход базовых блоков ГПУ, во время которого детекторы вычисляют свои множества, соответствующие GEN и KILL [11] в терминах классического анализа, а затем по правилам, заданным в тех же детекторах, обходится граф потока управления для построения IN и OUT. Наконец, вызывается обработчик завершения анализа, который использует собранную информацию для выдачи предупреждений.

После этапа анализа потоков данных работает чувствительный к путям анализ.

Здесь выполняется накопление информации в резюме каждой функции и анализ всех возможных путей выполнения программы с учётом уже построенных резюме.

Описанная схема работы позволяет эффективно сочетать поиск различных типов ошибок в программе, постепенно накапливая недостающую информацию и освобождая ненужные данные.

3. Построение графа вызовов

В рассматриваемом инструменте граф вызовов необходим для решения двух основных задач.

1. Для каждого вызова в программе (с учётом делегатов, интерфейсов, геттеров и сеттеров свойств, лямбда-выражений) определить вызываемую функцию или множество функций, которые могут быть вызваны.
2. Построить правильный порядок обхода функций, чтобы вызываемые функции были проанализированы к моменту анализа вызывающих. Это позволяет достичь контекстной чувствительности за счёт использования резюме вызываемых функций.

Построение графа вызовов состоит из двух этапов. Сначала осуществляется обход АСД, во время которого происходит:

1. Анализ вершин, соответствующих объявлениям класса, для построения иерархии классов. Эта информация используется на втором этапе для уточнения графа в контексте виртуальных вызовов.
2. Анализ вершин АСД, в которых возможен вызов:
    a. непосредственный вызов метода, статического метода, конструктора;
    b. доступ к свойству объекта, когда возможен вызов геттера или сеттера;
    c. создание лямбда-выражений и анонимных функций;
    d. неявные вызовы переопределённых операторов;
    e. неявные вызовы операторов преобразования;
    f. вызовы финализаторов (деструкторов).

    Для каждого узла АСД, в котором происходит вызов, осуществляется связывание по строковому идентификатору функции с существующим резюме или создаётся новое резюме.

После завершения компиляции всего решения выполняется построение графа вызовов при помощи адаптированного для C# алгоритма Class Hierarchy Analysis (CHA) [12]. Полученный в результате граф топологически сортируется, после чего производится выбор функций, допускающих параллельный анализ.

4. Построение графа потока управления

Граф потока управления лежит в основе анализа и должен отражать все возможные пути выполнения программы. В рассматриваемом анализаторе

базовые блоки состоят из вершин АСД. Это позволяет сохранить связь между сложными анализами путей исполнения и соответствующими оригинальными конструкциями языка. Использование языково-независимого представления, например, трехадресного кода, позволило бы упростить анализ за счет избавления от синтаксического сахара. Однако такой подход приводит к потере информации об исходных конструкциях, что влияет на качество последующего анализа и, особенно, содержания предупреждений. В примере 1 показан единственный содержательный базовый блок ГПУ для метода `foo`. Каждая инструкция базового блока – это вершина АСД, тип которой указан в правом столбце. Конструкция **[число]** означает номер инструкции в базовом блоке, результат которой используется в качестве аргумента в данной инструкции.

```
public void foo()
{
   int p = 1;
   bar(p,
     new MyClass());
}
```

```
0: 1                        LiteralExpression
1: int p = [0]          VariableDeclaration
2: MyClass                  IdentifierName
3: new [2]()       ObjectCreationExpression
4: p                        IdentifierName
5: bar                      IdentifierName
6: [5]([4], [3])       InvocationExpression
```

Пример 1. Базовый блок графа потока управления
Example 1. Basic block of control flow graph

Ребра, соединяющие базовые блоки, хранят разные атрибуты, указывающие на то, является ли ребро обратным для циклов, соответствует положительной или отрицательной ветви условного оператора, является ли входом, выходом или замыканием лямбда-выражения, соответствует ли ребро переходу по пользовательскому или системному, явному или неявному исключению и некоторые другие. Кроме того, на рёбрах хранятся условия перехода для чувствительного к путям анализа.

Для упрощения работы с циклами на фазе чувствительного к путям анализа, условие входа в цикл дублируется в его конце, чтобы позволить совершить выход из цикла без прохода по обратному ребру.

Существенное влияние на результаты анализа оказывает способ представления исключений в ГПУ. В языке C# практически каждая инструкция в исходном коде может вызвать исключение – например, NullReferenceException. Практика программирования также поощряет использование исключений. В итоге вызов практически любой функции может закончиться выбросом исключения. Практические эксперименты показали, что необходимо разделять пользовательские и системные исключения, выброшенные явно оператором throw и пришедшие из вызовов библиотечных функций. Такое разделение используется в детекторе утечки unmanaged памяти в случае, например, отсутствия свободного места на диске

или ошибки передачи данных по сети. Кроме того, для корректной обработки явных перехватов исключений, требуется знать, какие исключения могут возникнуть во время выполнения функции.

Таким образом, при построении ГПУ для каждого вызова функции необходим список всех возможных в ней исключений. Для пользовательских функций этот список строится во время анализа, а информация о возможных исключениях при работе библиотечных функций загружается из базы данных резюме, более подробное описание которой содержится в части 5.

Разбиение базового блока и создание ребра для исключения после каждого вызова существенно увеличивает как сложность самого графа, так и время его последующей обработки. Поэтому на данный момент дробление базового блока осуществляется только для явных пользовательских исключений. Остальные добавляются как возможные выходы в конце базового блока. Однако это решение имеет недостатки. Рассмотрим пример сравнительно частой ситуации.

```
public StreamReader GetStreamReaderOrThrow(bool b)
{
     if (b)
         return new StreamReader("stream.txt");
     throw new NotImplementedException();
 }

 public void Test(bool b)
 {
     StreamReader reader = null;
     try
     {
         reader = GetStreamReaderOrThrow(b);
     }
     finally
     {
         reader.Dispose();
     }
 }
```

Пример 2. Код на C#, содержащий возможное использование null
Example 2. C #-code with possible use of null

Если в функции `Test` не добавлять ребро между точкой вызова и присваиванием, то ошибку возможного использования *null* в `reader.Dispose()` найти не получится.

Широкое распространение в коде на C# имеют также лямбда-выражения, лямбда-функции и анонимные функции. Для упрощения будем называть все

эти сущности одним словом лямбды. Без точного межпроцедурного чувствительного к путям анализа определить место их вызова невозможно. Поэтому для нужд различных детекторов используются различные эвристики. На этапе анализа потоков данных работают сравнительно простые детекторы, поэтому для них разумным компромиссом является непосредственное встраивание тела лямбды в ГПУ. Поскольку количество исполнений лямбды также неизвестно, добавляются ребро, минующее тело лямбды, и ребро, ведущее из ее выхода во вход. Такое встраивание позволяет повысить качество анализа по аналогии с компиляторной оптимизацией inlining, давая анализу информацию о том, что будет происходить с переменными, использованными в лямбдах. Например, это позволяет существенно повысить качество анализа неиспользуемых значений без более слабых эвристик, считающих, что лямбды используют все переменные. По окончанию анализа потоков данных ребра, соединяющие тело лямбды с остальной функцией, разрываются. Таким образом, в дальнейшем лямбды не учитываются.

Язык C# содержит достаточное количество удобного синтаксического сахара, который также требуется представлять в ГПУ. К таким конструкциям относятся операторы “??”, “?.”, yield break, yield return, switch по строкам и goto по вычисляемым выражениям, интерполированные строки и т.д. Основная сложность их поддержки в графе потока управления заключается в аккуратном создании базовых блоков и рёбер между ними, а также построении правильных условий переходов по этим рёбрам на этапе анализа путей.

Таким образом, построение ГПУ C# программы для последующего использования в статическом анализаторе – это поиск компромиссов между сложностью предшествующего построению анализа и последующего. Набор предложенных выше методов построения сохраняет большую часть информации в удобном для последующего обхода и анализа виде. Обходчик путей выполнения программы по ГПУ выдаёт набор предопределённых событий, обработчики которых реализованы в детекторах.

5. Чувствительный к путям анализ

В отличие от большинства доступных инструментов статического анализа, SharpChecker поддерживает чувствительный к путям и контексту анализ. Такой анализ позволяет обнаруживать наиболее интересные типы ошибок, такие как утечка ресурсов, использование *null*, ошибки приведения типов.

Для проведения внутрипроцедурного анализа используется подход, схожий с подходом BMC[13]. Подход BMC рассматривает пути на графе потока управления, которые начинаются в точке входа в программу, и длина которых не превышает некоторую константу. При подсчёте длины допускается наличие разных весов у разных рёбер графа. В данной работе для выполнения развёртки циклов считается, что прямое ребро ГПУ имеет вес ноль, а обратное – один. Таким образом в ходе анализа рассматривается лишь конечный набор

путей в ГПУ. В отличие от оригинального BMC, рассматриваемый подход при обработке вызовов функций использует резюме функций вместе вставки, а также использует объединение состояний в точках слияния.

Для представления множества путей ГПУ, которые будут проанализированы, удобно ввести понятия графа развёртки. Графом развёртки будем называть граф, обладающий следующими свойствами:

- граф развертки является ациклическим;
- каждая вершина графа развертки сопоставлена какой-либо вершине ГПУ;
- в графе развертки выделены две вершины: входная и выходная, которые сопоставлены входной и выходной вершине графа ГПУ соответственно;
- между двумя вершинами графа развертки есть ребро только в том случае, если в ГПУ есть ребро между соответствующими вершинами;
- все вершины графа развертки достижимы из входной вершины;
- выходная вершина достижима из всех вершин графа развертки;

Каждому пути в графе развертки по построению соответствует путь в ГПУ, следовательно, граф развертки задает множество путей в ГПУ. В том случае, если какая-либо вершина в графе развертки имеет более одного входного ребра, будем считать, что для последующего анализа инструкций данной вершины необходимо вначале произвести объединение входных состояний. Таким образом граф развертки задает не только множество рассматриваемых путей, но и набор вершин, в которых будет производиться объединение состояний.

Далее будем считать, что граф развертки уже построен. В данной работе используется алгоритм построения графа развертки, описанный в статье [14]. Весь дальнейший анализ проводится не для ГПУ, а для графа развертки, что корректно ввиду соответствия их путей.

В данном подходе точкой входа анализа является точка входа в анализируемую функцию. Начальное состояние в точке входа, включающее аргументы функции и состояние памяти, параметризуется набором типизированных символьных переменных. Для параметризации аргументов функции достаточно рассмотреть отображение $Params : P \to S$, где P это множество аргументов функции, а S – множество символьных переменных. Будем различать три типа символьных переменных: целочисленную, булевую и ссылочную – соответственно, S_I, S_B и S_R. Все целочисленные символьные переменные являются знаковыми с фиксированным размером в битах. Целочисленные и булевые символьные переменные могут принимать любое допустимое их типом значение. Значения ссылочных переменных удовлетворяют следующему ограничению: две ссылочные переменные равны в том и только том случае, если они обе примут значение null. Из этого следует, что никакие ссылочные переменные не являются алиасами.

Кроме явно заданных параметров, начальное состояние также задает состояние памяти. Для описания параметризованного состояния памяти введем частичное отображение $Heap : S_R \times F \rightharpoonup S$, которое паре «символьная переменная и поле» ставит в соответствие символьную переменную, полученную при соответствующем чтении из поля. Параметризованным начальным состоянием будем называть пару отображений $State_{entry} = \langle Params, Heap \rangle$.

Для задания $Params$ достаточно каждому параметру функции сопоставить уникальную символьную переменную. Задание *Heap* осуществляется лениво. В том случае, когда к переменной, содержащей в качестве значения символьную переменную s, применяется операция чтения поля f, и пара $\langle s, f \rangle$ не содержится в Heap, Heap доопределяется $\langle s, f \rangle \rightarrow s_{new}$, где s_{new} – новая символьная переменная. В силу того, что анализу подвергается только конечный набор путей, число операций с памятью будет тоже конечным, следовательно, отображение Heap также будет определено на конечном наборе пар. Здесь и далее под состоянием памяти понимается состояние лишь той части памяти, которая доступна с помощью операций работы с памятью на рассматриваемых путях выполнения.

Параметризированное начальное состояние позволяет представлять результаты выполнения операторов функции в виде выражений над символьными переменными. Каждое символьное выражение также имеет один из трех типов: целочисленный, булевый или ссылочный. Правила построения символьных выражений следующие:

- константы являются символьными выражениями соответствующих типов;
- символьные переменные являются символьными выражениями соответствующих типов;
- для двух символьных выражений одинаковых типов определены операции сравнения, которые являются булевыми символьными выражениями;
- для двух целочисленных символьных выражений определены все арифметические и битовые операции, которые также являются целочисленными символьными выражениями;
- для двух булевых символьных выражений определены все стандартные булевы операции, которые также являются булевыми символьными выражениями;
- для символьных выражений определены стандартные унарные операции, которые также являются символьными выражениями соответствующих типов;
- для двух символьных выражений одинаковых типов и булевого символьного выражения определена тернарная условная операция того же типа, результат которой совпадает с первых выражением,

если булевое выражение верно, а иначе совпадает со вторым выражением.

Множество построенных таким образом символьных выражений обозначим как SE. Отдельно будем выделять SE_B.и SE_R, булевое символьное выражение и ссылочное символьное выражение соответственно.

Для задания резюме функции необходимо ввести два дополнительных отображения: $Param_{out}: P \rightarrow SE$ и $Heap_{out}: S_R \times F \rightharpoonup SE$. Отображение $Param_{out}$ задает соответствие между выходными параметрами функции и символьными переменными начального состояния, а $Heap_{out}$ между состоянием памяти после выполнения функции и символьными переменными. Тогда резюме функции задает отображения $State_{entry}$, $Param_{out}$, $Heap_{out}$. Отображения $Param_{out}$ и $Heap_{out}$ могут содержать дополнительные символьные переменные, не являющиеся частью параметризированного входного состояния. Наличие данных переменных обусловлено как наличием вызовов функций, реализация которых отсутствует, так и ограничением на размер резюме. В том случае, если число различных символьных выражений в $Param_{out}$ и $Heap_{out}$ чрезмерно велико, часть из них заменяется на новые неизвестные символьные переменные. Новые символьные переменные, возникшие при применении резюме, также становятся частью параметризации начального состояния.

Пусть при анализе конкретной функции построено её параметризованное начальное состояние, а также резюме всех вызываемых в ней функций. Тогда абстрактным состоянием для заданного ребра графа развертки назовем множество возможных состояний памяти и переменных относительно выбранной параметризации начального состояния. Начальное состояние может быть описано тройкой

$$State = \langle Var: V \rightarrow SE, Heap: S \times F \rightarrow SE,\ Pred \rangle, Pred \in SE_B.$$

Var и $Heap$ задают состояние переменных и памяти через символьные переменные, а $Pred$ является предикатом пути, задающим ограничения на символьные переменные. Тогда каждое решение предиката пути, т.е. такой набор символьных переменных на котором $Pred$ обращается в истину, будет однозначно задавать состояние переменных и памяти.

В работе [14] представлен метод, позволяющий по заданным графу развертки и набору резюме вызываемых функций строить абстрактные состояния $State$ для каждого ребра графа развертки. В данном методе, резюме для анализируемой функции строится из абстрактного состояния $State$, полученного в выходной вершине графа развертки.

Построение детекторов ошибок происходит на базе абстрактного состояния $State$. Каждый детектор может расширить $State$, добавив к нему дополнительную информацию и определив правила объединения этой информации. Типичным примером дополнительной информации является частичное отображение $SE \rightharpoonup SE_B$. Такое отображение задает условие, при

котором символьное выражение обладает заданным свойством. Примерами таких свойств являются «символьное выражение сравнивалось с явным null» или «символьное выражение является ссылкой на освобожденный ресурс». Детектор ошибки может следить за символьным выполнением, изменяя свою часть абстрактного состояния. В случае, если символьное выражение, обладающее заданным свойством при данном условии, используется в потенциально опасной операции, то детектор выдаст предупреждение, если предикат пути и данное условие имеют общее решение.

6. Вычисление условий

Провести вычисления предиката пути можно наивным алгоритмом, пересчитывая предикат с учетом условий на ребрах и объединяя предикаты путей различных абстрактных состояний в точках объединения. Данный алгоритм предложен в предыдущей работе [14]. Однако в силу того, что объединению зачастую подвергаются ветки оператора if с противоположными условиями, итоговые формулы предикатов путей могут быть сильно упрощены. Такое упрощение используется как для уменьшения потребления памяти при сохранении предикатов в резюме, так и для ускорения решения совместности формул.

Для упрощения формул можно использовать два подхода. Первый подход заключается в использовании специального представления, автоматически упрощающего формулу во время её построения. Примером такого представления являются ROBBD [15].

Второй подход заключается в использовании свойства графа развертки: если через две вершины проходят одни и те же пути, то и предикаты путей в этих вершинах будут совпадать. Рассмотрим его подробнее.

Выберем конкретную вершину в графе развертки. Будем считать, что все вершины топологически меньше данной уже обработаны и их предикаты пути посчитаны. Найдем для данной вершины её область пост-доминирования. Рассмотрим граф вершин топологически меньших либо равных данной, назовём его G'. Построим разрез $\langle S, T\rangle$, такой, что к S относятся все вершины G', которые в исходном графе не пост-доминирует данная вершина, а к T – все вершины из области пост-доминирования. Пусть $\{s_i, t_i\}$ – список ребер, лежащих на разрезе. Тогда предикат пути для данной вершины построим следующим образом: $\bigvee_i(Pred(s_i) \wedge Cond(s_i, t_i))$, где $Pred(s_i)$ – предикат пути для вершины s_i, а $Cond(s_i, t_i)$ – условие перехода по ребру.

Стоит отметить, что оба подхода к минимизации размера формул могут применяться одновременно.

Перейдем к вычислению условия объединения. Пусть дана тройка вершин $\langle lhs, rhs, join\rangle$, таких, что есть ребра $\langle lhs, join\rangle$ и $\langle rhs, join\rangle$. Заранее известно, что вершина $join$ была достигнута либо на пути, прошедшем через ребро $\langle lhs, join\rangle$, либо на пути, прошедшем через ребро $\langle rhs, join\rangle$. Для того, чтобы

различать, по какому из этих двух ребер была достигнута вершина $join$, достаточно воспользоваться условием $lhs_{cond} = Pred(lhs) \vee Cond(lhs, join)$, соответствующим тому, что путь прошел по ребру $\langle lhs, join \rangle$. Аналогично, можно рассмотреть обратное условие $rhs_{cond} = Pred(rhs) \vee Cond(rhs, join)$ для ребра $\langle rhs, join \rangle$. В силу детерминированности выбранной параметризации, условия lhs_{cond} и rhs_{cond} несовместны. Однако эти условия зачастую слишком громоздкие, и для хорошей производительности их необходимо упростить.

Задачу упрощения можно сформулировать следующим образом. Необходимо найти такое условие *Interpol*, что $Interpol \rightarrow lhs_{cond}$ и $\neg Interpol \rightarrow rhs_{cond}$. Такое условие можно получить, применив к формулам lhs_{cond} и rhs_{cond} интерполирующий решатель [16]. Применение такого решателя на каждую операцию объединения достаточно затратно, поэтому рассмотрим алгоритм построения условия *Interpol* на основе дерева доминаторов.

Пусть dom это непосредственный доминатор вершин lhs и rhs, тогда рассмотрим множество вершин топологически меньше либо равных lhs и больше либо равных dom, назовём его L. Рассмотрим аналогичное множество для rhs, назовем его R. Рассмотрим тогда $L \cap R$ и $L \setminus R$, без ограничения общности будем считать, что $L \setminus R$ не пусто. Тогда $Interpol = \bigvee_i (Pred_{dom}(u_i) \wedge Cond(u_i, v_i))$, где $\{u_i, v_i\}$ – ребра, лежащие на разрезе $\langle L \cap R, L \setminus R \rangle$, а $Pred_{dom}(u)$ предикат пути из вершины dom до вершины u.

7. Резюме внешних функций

С точки зрения анализатора функции делятся на пользовательские, т.е. те, для которых есть исходный код, и внешние, или библиотечные, – из системных или пользовательских библиотек. В связи с этим, их резюме существенно отличаются.

Рассмотрим сначала, где можно взять информацию о внешних функциях. Во-первых, их можно анализировать в бинарном виде. Поскольку большинство распространённых библиотек для C# скомпилировано в CIL [17], можно использовать, например, Mono.Cecil[18] для дополнительного анализа бинарного представления. Одна из первых версий описываемого инструмента использовала этот подход, однако впоследствии решено было отказаться от анализа CIL, поскольку для получения качественных результатов необходима серьёзная инфраструктура, практически дублирующая функциональность анализатора исходного кода, что требует существенного времени на реализацию.

Поскольку большинство исходных текстов для популярных библиотек доступно для скачивания и анализа, другим способом является предварительный анализ исходного кода библиотеки по аналогии с

пользовательскими функциями или даже анализ на лету. Второй способ требует распространения огромного количества исходного кода библиотек вместе с инструментом, а также существенно увеличивает объем анализируемого кода, поскольку многие функции имеют сложную реализацию, кроме того используют другие функции без исходного текста, включая unmanaged библиотеки, например, WinAPI [19].

Предварительный анализ также не решает проблему с функциями без исходного кода, а также требует разработки механизмов сериализации и постоянного обновления после исправления ошибок или улучшения качества анализатора. Кроме того, для корректного анализа требуется поддержка различных версий одних и тех же библиотек.

Опыт разработки показывает, что для анализа библиотечных функций в большинстве случаев достаточно информации, представимой в виде нескольких десятков текстовых или логических атрибутов. Поэтому одним из используемых на данный момент решений является SQL база данных, содержащая записи для более 60000 популярных функций. База создавалась путём автоматизированного разбора и анализа официальной документации MSDN [20], а также исправлений и дополнений по результатам оценки работы инструмента. Для каждой функции хранится битовый вектор, представляющий логические свойства, например, что она возвращает Disposable объект и различные другие свойства, например, список возможных исключений. Подобным образом хранится информация о параметрах, например, что они не могут быть *null*. Такой подход позволяет быстро, достаточно качественно и предсказуемо предоставлять информацию, достаточную для большинства детекторов анализатора.

Серьёзным недостатком данного подхода является сложность моделирования сторонних эффектов и условий их возникновения. Рассмотрим характерный пример 3.

```
public string Foo()
{
    var list = new List<int>();
    list.Add(5);
    string check = null;
    foreach (var elem in list)
    {
        check = "Not null";
    }
    return check.ToString();
}
```

Пример 3. Код на C#, порождающий ложное срабатывание без дополнительного моделирования библиотечных функций
Example 3. C #-code generating a false alarm without additional modeling library functions

В результате добавления в `list` элемента 5 список `list` перестает быть пустым, поэтому тело цикла `foreach` всегда выполняется, и переменная `check` всегда инициализируется, следовательно, использование *null* невозможно. Однако анализатор предполагает, что `list` может быть пустым, потому что в описанной модели функции `List<T>.Add(T)` нет возможности прямо указывать подобные эффекты. Поэтому переход по ребру в обход `foreach` возможен, что приводит к выдаче ложного предупреждения.

Для поддержки сложных эффектов используется моделирование внешних функций на языке C#. Модель функции в данном случае представляет собой упрощённую реализацию. Такая реализация компилируется и анализируется на лету как пользовательская функция, и для неё строится резюме, как для функции, имеющей исходный код.

8. Результаты работы инструмента

В данном разделе приводятся результаты тестировании инструмента SharpChecker на наборе проектов с открытым исходным кодом. В таблице 1 представлено количество предупреждений, выданных для различных проектов, а также время работы анализатора. Представлены наиболее интересные с точки зрения данной работы группы ошибок, при поиске которых используется чувствительный к путям и контексту анализ:

- Null reference – ошибки, связанные с потенциальным использованием null. В данную группу объединены как использование переменной, которой могло быть явно присвоено значение null, так и использование переменной после или до сравнения с null;
- Resource leak – утечки ресурсов, связанные с отсутствием Dispose или неправильным использованием конструкции using;
- Wrong cast – ошибки приведения типов;
- Unreachable code – недостижимый код.

Таблица 1. Результаты тестирования инструмента SharpChecker на наборе программ с открытым исходным кодом
Table 1. Results of the Sharp Checker tool testing on a set of open source software

Проект	LOC	Null reference	Resource leak	Wrong cast	Unreachable code	Время анализа (мин:сек)
Sake	1K	0	0	0	0	0:12
Polly	5K	0	0	0	2	0:21
BobBuilder	6.5K	1	14	0	3	0:12
Shadowsocks	18K	0	10	4	6	0:42

Perspective	20K	10	1	0	1	0:20
CSParser	21K	13	1	0	7	0:48
NetMQ	30K	11	10	1	7	0:47
Jil	49K	33	1	0	20	8:14
LibGit	51K	15	14	0	1	1:06
OpenBVE	57K	5	18	0	50	14:03
Cassandra	63K	19	53	2	4	1:44
OpenRA	105K	42	31	0	22	8:11
FSpot	110K	179	24	0	30	1:47
ShareX	145K	42	53	0	23	2:45
Banshee	168K	143	20	0	34	3:03
Lucene.Net	528K	63	820	1	87	13:39
Roslyn	1.35M	1399	0	5	139	36:59

Можно заметить, что объем проекта не всегда коррелирует с временем анализа: это объясняется наличием в некоторых проектах сложных участков кода. Большое количество NRE на некоторых проектах объясняется стилем кодирования, при котором оператор *as* используется вместо явного приведения типа. Большое количество утечек ресурсов происходит в тестах.

9. Заключение

Разработка инструментов статического анализа – это поиск компромисса между скоростью работы и качеством результата. В отличие от компилятора, статический анализатор допускает некоторые эвристики, позволяющие ускорить работу за счёт неконсервативной обработки наиболее ресурсоёмких конструкций или процессов. В работе представлен набор методов, в том числе эвристических, комбинация которых позволяет достигать хороших результатов как по качеству результата, так и по времени работы.

На базе компиляторной инфраструктуры Roslyn разработан инструмент статического анализа программ на языке C# SharpChecker [8]. В данном инструменте реализованы предложенные методы организации чувствительного к путям и контексту межпроцедурного анализа. Результаты тестирования инструмента SharpChecker показывают его высокую эффективность на промышленных проектах с открытым исходным кодом.

Список литературы

[1]. Веб-сайт TIOBE Index [HTML] http://www.tiobe.com/tiobe_index
[2]. Описание инструмента LINQ (Language-Integrated Query): Microsoft Developer Network [HTML] https://msdn.microsoft.com/ru-ru/library/bb397926.aspx

[3]. 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: 10.15514/ISPRAS-2014-26(1)-7
[4]. Веб-сайт Coverity: Software Testing and Static Analysis Tools [HTML] http://www.coverity.com/
[5]. Веб-сайт Klocwork: Source Code Analysis Tools for Security & Reliability [HTML] http://www.klocwork.com/
[6]. Веб-сайт SonarLint [HTML] http://www.sonarlint.org/
[7]. Веб-сайт PVS-Studio for C/C++/C# [HTML] http://www.viva64.com/
[8]. Веб-сайт SharpChecker [HTML] http://sharpchecker.ispras.ru/ru/
[9]. Веб-сайт Visual Studio - Microsoft Developer Tools [HTML] https://www.visualstudio.com/ru-ru/visual-studio-homepage-vs.aspx
[10]. Веб-сайт Roslyn .NET Compiler Platform [HTML] https://github.com/dotnet/roslyn
[11]. Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д. Компиляторы. Принципы, технологии и инструментарий: пер. с англ. / под ред. И.В. Красикова. М.: ООО «И.Д. Вильямс», 2008. 1184 с.
[12]. Vijay Sundaresan, Laurie Hendren, Chrislain Razafimahefa, Raja Vallée-Rai, Patrick Lam, Etienne Gagnon, and Charles Godin. Practical virtual method call resolution for Java. 2000. In Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '00). ACM, New York, NY, USA, pp. 264-280. doi: 10.1145/353171.353189
[13]. Falke Stephan, Merz Florian, Sinz Carsten. LLBMC: Improved Bounded Model Checking of C Programs Using LLVM. 2010. Tools and Algorithms for the Construction and Analysis of Systems, pp. 623–626. doi: 10.1007/978-3-642-36742-7_48
[14]. В.К. Кошелев И.А. Дудина В.Н. Игнатьев А.И. Борзилов Чувствительный к путям поиск дефектов в программах на языке C# на примере разыменования нулевого указателя. Труды Института системного программирования РАН, том. 27 (выпуск 5), 2015.г., стр.. 59–86. DOI: 10.15514/ISPRAS-2015-27(5)-5
[15]. H. R. Andersen, An Introduction to Binary Decision Diagrams, 1997, Lecture notes [PDF] http://www.cs.utexas.edu/~isil/cs389L/bdd.pdf
[16]. K. L. McMillan, Interpolants from Z3 proofs, 2011. Formal Methods in Computer-Aided Design (FMCAD), pp. 19–27.
[17]. Описание стандарта ECMA-335 [PDF] http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf
[18]. Веб-сайт библиотеки Mono.Cecil http://www.mono-project.com/docs/tools+libraries/libraries/Mono.Cecil/
[19]. Описание Windows API в Microsoft Developer Network [HTML] https://msdn.microsoft.com/en-us/library/cc433218
[20]. Веб-сайт Библиотека классов .NET Framework в Microsoft Developer Network [HTML] https://msdn.microsoft.com/ru-ru/library/mt472912(v=vs.110).aspx

C# static analysis framework

V.K. Koshelev <vedun@ispras.ru>
V.N. Ignatyev <valery.ignatyev@ispras.ru>
A.I. Borzilov <helendile@ispras.ru>

ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. The paper describes static analysis techniques that are used for defect detection in C# programs. The goal of proposed analysis approaches is to catch more defects within an acceptable amount of time. Although the paper contains a description of full analysis cycle, it mainly focuses on special aspects that distinguish C# analysis approaches from well-known Java and C++ techniques. The paper illustrates methods that allow taking into account C# specialties of all analysis stages: call graph and control flow graph construction, data flow analysis, context- and path-sensitive interprocedural analysis. We propose an adaptation of symbolic execution methods inspired by Bounded Model Checking and Saturn Software Analysis Project. The paper also explains the organization of memory model, which is suitable for both a precise intraprocedural analysis and a creation of compact function-bound conditions used for interprocedural analysis. Special attention is paid to optimization of condition size and simplicity during a path sensitive-analysis. The conditions produced by a path-sensitive analysis are supposed to be solved by modern SMT solvers like Microsoft Z3 Prover. Different approaches to external functions modeling are covered. All proposed approaches are implemented in the SharpChecker static analysis tool and, as evaluated on several open source C# projects of varying size (1K - 1.35M lines of code), display good results and scalability.

Keywords: static analysis; null pointer dereference; path-sensitive analysis; context-sensitive analysis; function summary; bug detection; Roslyn; C#

DOI: 10.15514/ISPRAS-2016-28(1)-2

For citation: Koshelev V.K., Ignatyev V.N., Borzilov A.I. C# static analysis framework. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 1, 2016**,** pp. 21-40 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-2

References

[1]. TIOBE Index [HTML] http://www.tiobe.com/tiobe_index
[2]. LINQ (Language-Integrated Query): Microsoft Developer Network [HTML] https://msdn.microsoft.com/ru-ru/library/bb397926.aspx
[3]. 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 [Proceedings of ISP RAS], volume. 26 (issue 1). 2014. pp. 231-250 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-7
[4]. Coverity: Software Testing and Static Analysis Tools [HTML] http://www.coverity.com/
[5]. Klocwork: Source Code Analysis Tools for Security & Reliability [HTML] http://www.klocwork.com/
[6]. SonarLint [HTML] http://www.sonarlint.org/
[7]. PVS-Studio for C/C++/C# [HTML] http://www.viva64.com/
[8]. SharpChecker [HTML] http://sharpchecker.ispras.ru/ru/
[9]. Visual Studio - Microsoft Developer Tools [HTML] https://www.visualstudio.com/ru-ru/visual-studio-homepage-vs.aspx
[10]. Roslyn .NET Compiler Platform [HTML] https://github.com/dotnet/roslyn

[11]. Alfred V. Aho , Monica S. Lam , Ravi Sethi , Jeffrey D. Ullman, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2006

[12]. Vijay Sundaresan, Laurie Hendren, Chrislain Razafimahefa, Raja Vallée-Rai, Patrick Lam, Etienne Gagnon, and Charles Godin. Practical virtual method call resolution for Java. 2000. In Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '00). ACM, New York, NY, USA, pp. 264-280. doi: 10.1145/353171.353189

[13]. Falke Stephan, Merz Florian, Sinz Carsten. LLBMC: Improved Bounded Model Checking of C Programs Using LLVM. 2010. Tools and Algorithms for the Construction and Analysis of Systems, pp. 623–626. doi: 10.1007/978-3-642-36742-7_48

[14]. V. Koshelev, I. Dudina, V. Ignatyev, A. Borzilov. Chuvstvitel'nyj k putjam poisk defektov v programmah na jazyke C# na primere razymenovanija nulevogo ukazatelja (Path-sensitive bug detection analysis of C# program illustrated by null pointer dereference). Trudy ISP RAN [Proceedings of ISP RAS], volume 27 (issue 5), 2015. pp. 59–86 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-5

[15]. H. R. Andersen, An Introduction to Binary Decision Diagrams, 1997, Lecture notes [PDF] http://www.cs.utexas.edu/~isil/cs389L/bdd.pdf

[16]. K. L. McMillan, Interpolants from Z3 proofs, 2011. Formal Methods in Computer-Aided Design (FMCAD), pp. 19–27.

[17]. ECMA-335 Standard [PDF] http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf

[18]. Mono.Cecil http://www.mono-project.com/docs/tools+libraries/libraries/Mono.Cecil/

[19]. Windows API: Microsoft Developer Network [HTML] https://msdn.microsoft.com/en-us/library/cc433218

[20]. .NET Framework Class Library: Microsoft Developer Network [HTML] https://msdn.microsoft.com/en-us/library/mt472912(v=vs.110).aspx

Основанный на резюме метод реализации произвольных контекстно-чувствительных проверок при анализе исходного кода посредством символьного выполнения

А.В. Дергачёв <dergachev.a@samsung.com>
А.В. Сидорин <a.sidorin@samsung.com>,
Исследовательский центр Samsung,
127018, Россия, г. Москва, ул. Двинцев, дом 12, корпус 1

Аннотация. Описывается методика, позволяющая реализовать поиск дефектов достаточно общего и произвольного вида при использовании межпроцедурного анализа методом резюме при анализе исходного кода программы на высокоуровневых языках программирования, таких как C и C++. Основное внимание уделено трудностям, возникающим при построении и применении резюме в процессе анализа исходного кода (по сравнению с анализом низкоуровневого представления программы), а также достижению гибкости метода анализа, необходимой для поиска дефектов произвольного вида.

Ключевые слова: статический анализ, символьное выполнение, межпроцедурный анализ, контекстно-чувствительный анализ, резюме, C, C++, Clang Static Analyzer

DOI: 10.15514/ISPRAS-2016-28(1)-3

Для цитирования: Дергачёв А.В., Сидорин А.В. Основанный на резюме метод реализации произвольных контекстно-чувствительных проверок при анализе исходного кода посредством символьного выполнения. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 41-62. DOI: 10.15514/ISPRAS-2016-28(1)-3

1. Введение

Автоматизация поиска ошибок в программах и генерации тестов посредством статического анализа исходного кода является одним из методов контроля качества программного продукта, позволяющим выявлять дефекты, трудно обнаруживаемые другими традиционными методами [1, 2]. Символьное исполнение — простая и мощная методика, изначально сформулированная Кингом [3] и заключающаяся в том, что автоматический анализатор пытается

смоделировать поведение программы шаг за шагом вдоль всех возможных путей в графе потока управления, вводя для неизвестных величин символьные обозначения ("символы"), и объявляя возможные комбинации конкретных значений символов эквивалентными, если они ведут программу по одному и тому же пути исполнения. Всякое ветвление в программе, таким образом, разбивает множество "состояний программы" — возможных комбинаций значений символов — на классы эквивалентных состояний, и при последующем обходе эти классы состояний анализируются независимо. Подобное поведение метода символьного выполнения, будучи естественным, представляет собой одновременно и существенную проблему масштабируемости, поскольку каждое ветвление означает удвоение сложности дальнейшего анализа, приводя к экспоненциальному росту сложности. Улучшение масштабируемости анализа является, таким образом, важной практической проблемой, от решения которой зависит возможность производить все более и более исчерпывающий анализ все более и более сложных программ.

Символьное исполнение может применяться при статическом анализе скомпилированного исполняемого файла программы [4], текста программы в упрощенном промежуточном представлении, таком как Java-байткод [5] или LLVM [6], либо непосредственно исходного кода программы [7], а также исполняемого файла при смешанном статическом и динамическом анализе [8]. При этом символьное исполнение исходного кода, написанного на высокоуровневом языке программирования, является наиболее сложным из-за богатства языковых конструкций, поведение которых должно быть смоделировано с учетом того, что значения величин, которыми они оперируют, являются, вообще говоря, символьными; однако результаты подобного анализа являются наиболее полезными, поскольку позволяют получить наглядное и полноценное описание найденных дефектов в терминах исходного языка и исходного кода.

Межпроцедурный анализ означает возможность учитывать контекст вызова функции, благодаря чему корректно находить дефекты, не локализованные ни в одной отдельно взятой функции, но охватывающие несколько вызовов. Например, две проблемы двойного освобождения памяти в простейшем примере кода, приведенном ниже, не могут быть обнаружены без достаточно мощной реализации межпроцедурного анализа:

```
01    void foo(void *p) {
02      free(p);
03    }
04    void bar() {
05      void *q = malloc(1);
06      free(q);
07      foo(q);
```

```
08    }
09    void baz() {
10      void *q = malloc(1);
11      foo(q);
12      free(q);
13    }
```

Листинг 1. Примеры дефектов, для обнаружения которых необходим межпроцедурный анализ
Listing 1. Examples of defects, detection of which requires interprocedural analysis

При этом дефект в функции `bar()` находится в момент анализа функции `foo()` в контексте функции `bar()`, а дефект в функции `baz()` находится во время анализа самой функции `baz()`, однако правильная эмуляция вызова функции `foo()` все равно необходима для успешного обнаружения дефекта. В дальнейшем мы будем неоднократно обращаться к этому примеру.

Контекстная чувствительность, будучи необходимой для поиска более глубоких дефектов, ставит еще острее проблему масштабируемости, поскольку возможные пути выполнения, подвергающиеся анализу, существенно усложняются. Особенно проблематичным является анализ крупных программных комплексов, таких как Android [9], при поиске дефектов, затрагивающих сразу несколько компонентов анализируемой системы. Подобный анализ будет не только межпроцедурным, но и «межмодульным» — охватывающим сразу несколько единиц трансляции. Наивная реализация межпроцедурного анализа путем встраивания [10], при которой всякий вызов функции подразумевает моделирование этой функции заново с учетом контекста, является наиболее простой и в то же время наиболее плохо масштабируемой. Более перспективной выглядит подход, основанный на резюме: каждая функция моделируется единожды, по результатам анализа составляется резюме внешних эффектов функции, а при каждом моделировании вызова пересчету с учетом контекста подвергаются лишь внешние эффекты.

Идея оптимизации межпроцедурного анализа при помощи метода резюме упомянута еще в монографии [11]. Первые эффективные реализации метода появились не столь давно. В частности, поиску дефектов методом символьного выполнения бинарного кода и промежуточного представления посвящены работы [12, 13]. В работах [14, 15] описывается методика поиска отдельно взятых видов дефектов при помощи символьного исполнения исходного кода методом резюме. В настоящей работе описывается методика, позволяющая реализовать поиск дефектов достаточно общего и произвольного вида в условиях реализации межпроцедурного анализа методом резюме, при анализе высокоуровневого исходного кода программы. Основное внимание будет уделено специфике построения и применения

резюме при анализе исходного кода, которое оказывается существенно более сложным, чем в случае анализа низкоуровневого кода, а также достижению гибкости метода анализа, необходимой для поиска дефектов произвольного вида.

В качестве примера готового фреймворка, на который мы будем ориентироваться при описании нашей реализации, мы будем использовать Clang Static Analyzer [16] (далее CSA) — статический анализатор исходного кода, созданный на базе компилятора Clang. В CSA реализован метод символьного выполнения исходного кода программы на языках C, C++ и Objective C, чувствительный к путям выполнения и потоку данных, и реализован межпроцедурный анализ методом встраивания. В рамках настоящей работы на базе CSA был реализован межпроцедурный анализ методом резюме.

2. Терминология и особенности CSA

Как упоминалось выше, символьное выполнение подразумевает исследование всех возможных путей сквозь граф потока управления. В случае CSA это означает, что основным объектом анализа является граф выполнения (реализуемый классом `ExplodedGraph`), состоящий из всех пройденных до настоящего момента путей графа потока управления (CFG). Вершинами графа выполнения являются пары, состоящие из точки выполнения (`ProgramPoint` — элемент блока CFG, конкретный оператор программы) и состояния программы (`ProgramState` — класс эквивалентных с точки зрения символьного выполнения возможных состояний программы). Ребро графа выполнения между вершинами A и B означает, что исполнение оператора, соответствующего точке выполнения вершины A, корректирует состояние точки A до состояния точки B и переводит программу в точку выполнения вершины B. Граф выполнения, вообще говоря, не является деревом, поскольку иногда можно двумя различными путями дойти до одной и той же точки в одном и том же состоянии. Однако для простоты мы будем пользоваться терминологией, которую обычно применяют к деревьям, например, называть пути выполнения ветвями графа, а вершины, не имеющие исходящих ребер — листьями.

С точки зрения CSA анализ представляет собой процедуру построения `ExplodedGraph`'a. CSA имеет каркасную структуру, в которой отдельные проверяющие модули (`Checkers`) реализуют различные виды проверок, а их выполнением управляет ядро системы. Проверяющие модули активно влияют на построение `ExplodedGraph`'a, подписываясь на определенные события, происходящие при его построении (такие как анализ точек программы заданного типа), уточняя состояние программы, отсекая недостижимые ветви выполнения, выдавая отчеты о найденных дефектах.

Состояние программы содержит срез состояния памяти (`Store`), реализующей чувствительность к потоку данных и приписывающей конкретным или символьным регионам памяти конкретные или символьные значения, а также среду выполнения (`Environment`), приписывающую символьные значения текущим выражениям в коде.

Также оно содержит множества возможных значений символов: в соответствии с методикой символьного выполнения, при моделировании оператора ветвления в графе выполнения появляются вершины, соответствующие ветвям исполнения, и состояния в них различаются ограничениями на символьное значение условия: на одной ветви условие предполагается истинным, а на другой — ложным, что и отражается в ограничениях. Различия в значениях символов, не влияющие на выбор ветви исполнения, считаются несущественными. Если накладываемые ограничения противоречат уже имеющимся в состоянии ограничениям на тот же символ (множества значений, описывающие эти ограничения, не пересекаются), то соответствующая ветка недостижима, и анализ по ней не производится.

Третий важный элемент состояния программы — нетипизированное хранилище данных (`GenericDataMap`, далее GDM), принадлежащих проверяющим модулям. Состояния программы, в которых проверяющие модули сохранили различные данные, считаются разными; однако ядро CSA не может точнее судить о содержимом GDM. Одно из основных применений этого механизма заключается в возможности производить анализ графа состояний объектов (“typestate analysis”, [17]). К примеру, проверяющий модуль, ищущий дефекты типа двойного освобождения памяти, подобные приведенным на листинге 1, может хранить в GDM состояние всех указателей на выделенную память (освобождены или нет), и состояния с освобожденным и неосвобожденным указателем будут считаться различными при прочих равных.

3. Составление резюме

3.1 Предварительные замечания

Резюме функции должно содержать достаточно информации для того, чтобы достаточно точно — либо, во всяком случае, достаточно «консервативно» (без добавления недостаточно ненадежных предположений о состоянии программы) — смоделировать влияние вызова функции на дальнейший анализ. Основной смысл резюме — в том, чтобы смоделировать лишь существенные изменения состояния, и не моделировать изменения, не имеющие внешнего эффекта, такие как, например, изменения состояния локальных переменных.

Реализации межпроцедурного анализа методом резюме предполагает однократное символьное выполнение каждой функции вне контекста. В

случаях, когда при составлении резюме встречается вызов другой функции, резюме которой еще составлено не было, анализ текущей функции прерывается на составление резюме вызываемой функции. В случае возникновения рекурсии, когда требуется применить резюме функции, построение которого на момент вызова начато, но не закончено, вызов функции приходится осуществить «консервативно» — без применения межпроцедурного анализа.

3.1 Особенности реализации

В случае CSA к внешним эффектам функции, требующим моделирования, относятся:

1. Наложения ограничений на символы. Если в процессе символьного выполнения функции выясняется, что различные множества значений функции направляют ее выполнение по разным путям и приводят к разным внешним эффектам, то в соответствии с методом символьного выполнения анализ вызывающей функции должен также разделить имеющийся `ProgramState` на соответствующие подклассы. Это означает, что резюме должно содержать информацию о разных ветвях функции, и все ветви должны рассматриваться независимо.
2. Для каждой ветви резюме должна храниться информация о символьных величинах, записываемых функцией в различные регионы памяти, а также о возвращаемом значении.
3. Проверяющие модули должны иметь возможность оставить в резюме пометки произвольного содержания, чтобы иметь возможность смоделировать свое влияние на GDM в процессе выполнения функции.

Хотя и возможно упростить резюме функции до содержимого пометок, позволяющих смоделировать эти три вида внешних эффектов, сама по себе процедура упрощения в значительной степени оказывается излишней. Поэтому, как только функция проанализирована, *мы будем понимать под резюме функции сам законченный* `ExplodedGraph` *этой функции.* Этот тривиальный способ составления резюме позволяет не тратить время на содержательное упрощение результатов анализа, хотя и подразумевает хранение всех графов всех функций, что увеличивает потребление памяти. Более того, вся информация, существенная для моделирования вызова функции, содержится в состоянии программы, соответствующем листу каждой ветви графа: информация, необходимая для наложение ограничений на символы, содержится в виде самих ограничений на символы, информация о записях во внешние регионы памяти содержится в окончательном `Store`, а проверяющие модули могут по ходу составления резюме сохранять в `ProgramState` свои пометки, и эти пометки будут доступны в финальном состоянии.

Есть и еще одно преимущество в хранении полного графа выполнения: полный путь выполнения позволит выдать более подробный отчет о найденном дефекте, как будет показано ниже. Тем не менее, вершины графа, не являющиеся листьями или ветвлениями, ссылки на которые не хранятся нигде в анализаторе или проверяющих модулях, можно удалить. Кроме того, потребление памяти можно заметно сократить, очищая структуры состояния программ в узлах результирующего графа от более ненужных записей. Каждое из этих упрощений на практике приводит к снижению потребления памяти примерно на 10%. Чтобы гарантировать наличие необходимой информации в окончательном состоянии программы, при анализе методом резюме приходится отключить ряд механизмов сборки мусора: так, даже если символ больше нигде не хранится в данном состоянии программы, удалить ограничения на него при построении резюме нельзя, хотя при обычном анализе методом встраивания это допустимо.

4. Применение резюме

Применение резюме происходит в момент вызова функции, для которой имеется построенное резюме, то есть построен готовый `ExplodedGraph`, быть может допустимым образом упрощенный. Процедура применения резюме включает в себя несколько этапов; задача этой процедуры — для каждой ветки резюме (листа графа выполнения вызываемой функции) корректным образом объединить информацию, содержащуюся в резюме, с информацией, содержащейся в текущем состоянии программы, то есть получить своего рода композицию этих двух состояний программы. Слиянию подвергается вся информация, содержащаяся в состоянии программы, хотя часть информации, хранящейся в резюме, может быть в этом процессе признана несущественной и пропущена. Информация об ограничениях, наложенных на символы, и о значениях, записанных в регионы памяти, будет перенесена в новое состояние ядром анализатора. Информация в GDM, имеющая смысл только для проверяющих модулей, будет в процессе применения резюме перенесена в новое состояние самими проверяющими модулями, отвечающими за нее. Это важное и неизбежное соображение означает, что в отличие от анализа методом встраивания, анализ методом резюме в качестве платы за эффективность и масштабируемость подразумевает существенную поддержку со стороны проверяющих модулей, приводящую, вообще говоря, к усложнению логики модулей и дополнительным трудностям при реализации новых проверок.

4.1 Актуализация символьных величин

Одной из основных трудностей при применении резюме является пересчет символов из контекста вызываемой функции ("формальных" символьных величин) — из той формы, в которой они записаны в резюме — в

"актуальные" с учетом контекста вызывающей функции символьные величины. Эту процедуру мы назовем актуализацией символьных величин. Она устанавливает соответствие между различными символами, возникшими при анализе различных функций. Актуализация символьных величин требуется на всех этапах применения резюме.

В CSA реализована подробная иерархия классов символьных величин, отражающая их применение к описанию разнообразных объектов языка высокого уровня. Помимо конкретных величин, имеющих известные численные, строковые или структурные значения, анализатор рассматривает собственно символы, которыми в процессе анализа обозначаются неизвестные численные значения, и регионы памяти, являющиеся единицами модели памяти CSA [18] и используемые как для хранения текущего состояния памяти в `Store`, так и для изображения неизвестных значений указателей.

Так, например, объявлению параметра функции в каждом контексте вызова соответствует регион памяти типа `VarRegion`, построенный по ссылке на объявление переменной в синтаксическом дереве программы. Здесь «контекст вызова» (`StackFrameContext`) может означать либо начало анализа, либо контекст на момент вызова подфункции. В начале анализа содержимое региона, соответствующего параметру, неизвестно, и обозначается специальным символом типа `SymbolRegionValue`, содержащим ссылку на данный `VarRegion`. Далее, если, например, параметр является указателем, то регион памяти, на который он указывает, будет «символьным регионом» (`SymbolicRegion`), хранящим ссылку на символ типа `SymbolRegionValue`, обозначающем значение этого указателя. А если с данным регионом в дальнейшем обращаются как с массивом, то регион элемента этого массива будет иметь класс `ElementRegion`, и хранить ссылку на объемлющий `SymbolicRegion` и на величину индекса элемента, причем последняя также может быть не только конкретной, но и символьной. Подобная иерархия отражает разнообразие конструкций высокоуровневого языка программирования и является существенно более сложной, чем требуется для анализа низкоуровневого представления программы, для которого обычно достаточно оперировать простыми адресами и сдвигами в памяти, как в [4].

Процедура актуализации принимает на вход два объекта — состояние программы в контексте вызывающей функции и символьную величину, которую требуется актуализировать. Если актуализируемая величина содержит в своем определении ссылки на другие величины, они рекурсивно подвергаются актуализации. Таким образом, актуализация представляет собой альфа-переименование символьной величины по некоторому конкретному шаблону. Затем процедура актуализации пытается восстановить символьную величину, являющуюся аналогом исходной, но состоящую из актуализированных подвеличин. Требуемые шаги при этом сильно различаются в зависимости от класса исходной величины. Как будет показано

ниже, тип актуализированной величины не обязан совпадать с классом исходной. Однако некоторые инварианты сохраняются: так, величина типа lvalue всегда актуализируется в lvalue, и аналогично для rvalue.

Другими словами, альфа-переименование подразумевает замену некоторых символьных величин на другие и последующий пересчет всех зависящих от них символьных величин. Говоря упрощенно, под актуализацией мы понимаем частный случай альфа-переименования, в котором *замене подвергаются регионы памяти в пространстве стековых аргументов функции* (сюда включаются `VarRegion`'ы от переменных-параметров и особый регион `CXXThisRegion`, содержащий значение указателя `this` в C++-коде), *и заменяются они на такие же регионы с откорректированным контекстом вызова* (`StackFrameContext`). Все остальные символьные величины, зависящие от этих регионов (такие как `SymbolRegionValue` от этих регионов), будут рекурсивно переименованы, а не зависящие — оставлены без изменения.

Продемонстрируем смысл и метод актуализации символьного значения на примере.

```
01    void foo(int x) {
02      if (x == 0) {}
03    }
04    void bar() {
05      foo(1);
06      foo(y);
07    }
```

Листинг 2. Пример вызова функции, для моделирования которого необходима актуализация символьных величин

Listing 2. Example of a function call, for which modeling an actualization of symbolic values is necessary

Резюме функции `foo()`, описанной в листинге 2, содержит две ветви. Их финальные состояния различаются ограничениями на символ, обозначающийся в терминах CSA как `reg_$0<x>`. Это символ класса `SymbolRegionValue`, обозначающий неизвестное значение, содержащееся в регионе памяти, соответствующем переменной-параметру `x`, в момент начала анализа, то есть начала построения резюме. В одной из ветвей на `reg_$0<x>` наложены ограничения `[0, 0]`, а в другой — `[INT_MIN, -1] U [1, INT_MAX]`. Регион памяти, соответствующий переменной `x`, является в терминах CSA регионом типа `VarRegion`.

На строке 05 производится вызов функции `foo()` с аргументом `1`. При моделировании этого вызова необходимо придать смысл ограничению на `reg_$0<x>`, различающему состояния двух ветвей резюме. Однако сам символ ни разу не встречается в контексте вызывающей функции. Регион

переменной `x` с точки зрения вызывающей функции есть ячейка стека, в которой передан параметр `1` — в этот момент и происходит собственно переименование одного региона параметра на другой: момент начала анализа, заложенный в определении `reg_$0<x>`, оказывается с точки зрения вызывающей функции моментом вызова `foo()`; на момент вызова функции `foo()` в регионе переменной `x` записано значение `1`; следовательно, правильным результатом актуализации символа `reg_$0<x>` при вызове функции `foo()` на строке `05` является конкретное значение `1`. Из этого примера ясно и общее правило: чтобы актуализировать символ типа `SymbolRegionValue`, нужно актуализировать соответствующий ему регион, и взять значение, записанное в этот регион. Отметим, что результатом актуализации того же самого символа `reg_$0<x>` при вызове функции `foo()` на строке `06` является символ `reg_$0<y>`. Более того, результатом актуализации региона `x` на строках 05 и 06 являются разные регионы, соответствующие одному и тому же (с точки зрения абстрактного синтаксического дерева программы) параметру `x` в разных пространствах (`MemorySpace`) аргументов разных вызовов функции `foo()`.

Приведем другой пример, демонстрирующий рекурсивный обход символьной величины в процессе актуализации.

```
01    void foo(const char *x) {
02      if (x[2] == 'a') {}
03    }
04    void bar(const char *y) {
05      foo(y);
06      foo("aaa");
07    }
```

Листинг 3. Пример вызова функции, для моделирования которого необходимо рекурсивное альфа-переименование символьных вечилин

Listing 3. Example of a function call, for modelling of which a recursive alpha-renaming of character values is necessary

При анализе кода из листинга 3 при построении резюме функции `foo()` возникает указатель `reg_$0<x>`, указывающий на заренее неизвестную строку символов. Функция затем разветвляется в зависимости от значения, записанного в один из регионов элементов этой строки. Символ, относительно которого производится ветвление, представляет собой `SymbolRegionValue` от `ElementRegion` от `SymbolicRegion` от `SymbolRegionValue` от параметра `x`. В процессе альфа-переименования при моделировании вызова на строке 05 параметр `x` будет принимать значение `reg_$0<y>`, по нему будет построен другой символьный регион, в котором, в свою очередь, в соответствующем элементе будет записано другое значение. На строке 06 результатом актуализации величины `reg_$0<x>` будет неизвестное значение

адреса строкового литерала `"aaa"` в памяти, результатом актуализации символьного региона — сам регион строкового литерала, а значения его элемента — конкретная (не символьная) величина `'a'`. На рисунках 1 и 2 изображена соответствующая этим двум процедурам иерархия регионов памяти. На рисунке 2 видно, что актуализированный регион может занять другой `MemorySpace` и перестать быть символьным.

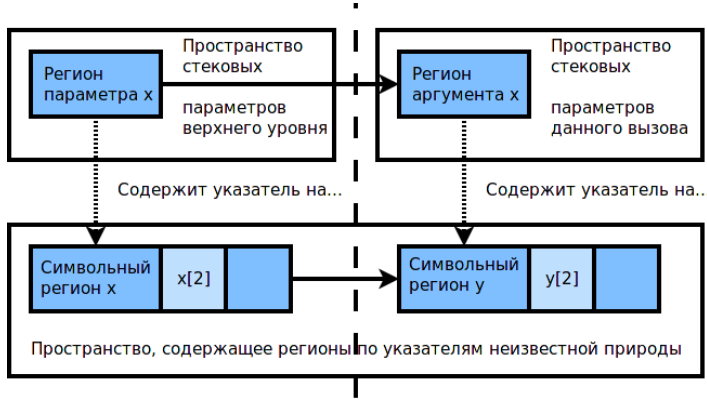


Рис. 1. Процесс рекурсивной актуализации символьного значения `x[2]` *в листинге 3 на строке 05*

Fig.1. The process of recursive actualization of character values x [2] on line 05 of Listing 3

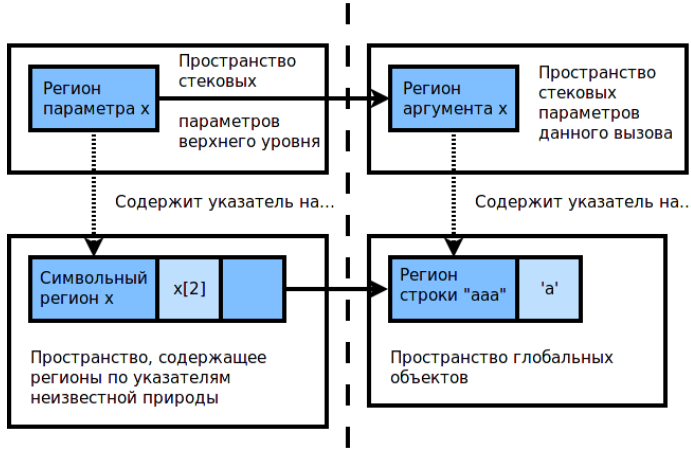


Рис. 2. Процесс рекурсивной актуализации символьного значения `x[2]` *в листинге 3 на строке 06*

Fig.2. The process of recursive actualization of character values x [2] on line 06 of Listing 3

Из приведенных примеров ясно, как описывается процедура актуализации в общем виде. Регионы памяти глобальных и статических переменных не актуализируются — остаются неизменными. Для регионов памяти, соответствующим параметрам функций, актуализация означает перемещение их в соответствующий `MemorySpace`. Подрегионы, такие как регион поля структуры или элемента массива, актуализируются посредством актуализации их базового региона и вычисления в нем аналогичного подрегиона. Регионы, соответствующие символьным указателям, такие как регионы на куче или регионы вокруг указателей неизвестного в контексте вызываемой функции происхождения, актуализируются посредством актуализации символа, лежащего в их основе; при этом они в результате актуализации могут перестать соответствовать символам, то есть стать конкретными — скажем, если происхождение символьного указателя было неизвестным в контексте вызываемой функции, но в контексте вызывающей функции этот указатель актуализировался в адрес известной переменной.

Конкретные значения не подвергаются актуализации. Символы подвергаются актуализации в соответствии с их назначением: для актуализации любого символа требуется получить величину, смысл которой передается назначением символа. Так, символ, которым обозначена неизвестная протяженность региона памяти (`SymbolExtent`), должен актуализироваться в протяженность актуализированного региона памяти, известную или неизвестную. Символ, обозначающий результат арифметической операции над другими символами (такой как `SymSymExpr`), после актуализации представляет собой результат этой же арифметической операции над другими символами — актуализированными левой и правой частью. Наконец, актуализация символов, смысл которых определяется проверяющими модулями (`SymbolMetadata`), неизбежно возлагается на сами проверяющие модули.

Особое внимание следует уделить актуализации символов, созданых для обозначения прочих неизвестных величин, таких как возвращаемое значение функции, тело которой недоступно для анализа (`SymbolConjured`). Эти символы нумеруются целыми числами, позволяющими их различать между собой, но по существу не несут никакой другой полезной информации о своем происхождении; при их актуализации требуется перенумеровать эти символы, чтобы они не конфликтовали с другими уже занятыми номерами.

Важно же здесь то, что при межпроцедурном анализе методом встраивания смысл (класс) символа, в отличие от смысла региона, несет сравнительно мало полезной информации. Однако для метода резюме иерархия символов жизненно необходима именно в связи с процедурой актуализации: так, в примере из листинга 2 мы не можем позволить себе потерять информацию о том, что `reg_$0<x>` является значением параметра `x`, а не просто неизвестной величиной. Таким образом, применение символов типа `SymbolConjured`, не хранящих информацию о своем происхождении, в

случае анализа методом резюме сильно ограничено и должно сводиться к случаям, когда содержательной актуализации заведомо не требуется. Другими словами, *требование возможности осуществить процедуру актуализации накладывает жесткие требования на иерархию символов*. К счастью, иерархия символов CSA претерпела лишь одно изменение: был введен символ, обозначающий адрес региона памяти, в который будет актуализирован символьный указатель, если в контексте вызываемой функции он указывает на известный регион. Без этого изменения иерархия символов CSA была недостаточно гибкой для отражения результата актуализации этого символа. В остальном можно утверждать, что иерархия символов CSA допускает актуализацию, при помощи техники, описанной выше. Описание в общем виде условий на иерархию символьных величин, необходимых и достаточных для проведения актуализации, выходит за рамки настоящей работы.

4.2 Применение резюме на стороне ядра анализатора

С точки зрения ядра анализатора применение ветки резюме выполняется следующим образом. Прежде всего анализу подвергаются ограничения на символы, которые однозначно определяют данную ветку. Всякое ограничение представляет собой пару из символа и множества принимаемых им конкретных значений. Символы, соответствующие ограничениям, хранящимся в резюме, подвергаются актуализации. Полученные в результате актуализации символы могут быть ограничены в контексте вызываемой функции; в противном случае будем считать их множеством их значений весь диапазон возможных значений их типа. Если множество значений символа в контексте вызываемой функции не пересекается с множеством значений актуализированного символа в контексте вызывающей функции, то ветка считается недостижимой и анализ этой ветки не производится; в противном случае новое состояние будет содержать ограничение актуализированного символа до пересечения этих двух множеств.

Затем вычисляется возвращаемое значение вызываемой функции. Это вычисление сводится к актуализации сохраненного в резюме возвращаемого значения.

Далее производится обращение к проверяющим модулям с целью слияния GDM резюме и GDM текущего состояния. Эта процедура скрыта от ядра анализатора. Проверяющие модули часто хранят в GDM символьные величины; на этом этапе они должны будут актуализировать эти величины и перенести в GDM нового состояния. Также в этот момент проверяющие модули могут произвести проверки, обнаружить дефекты в программе и выдать отчет о дефекте. Проверяющим модулям доступно возвращаемое значение функции, вычисленное на предыдущем шаге, а также множества значений всех возможно интересующих их символов, таким образом

запустить проверяющие модули раньше не представляется возможным. Поскольку проверяющие модули могут разветвить состояние на этом шаге, дальнейшие шаги выполняются отдельно для каждой вновь полученной ветки.

На последнем этапе происходит моделирование записей в глобальные переменные. Анализатор перебирает хранящийся в резюме `Store`, анализируя каждую пару, состоящую из региона памяти и связанной с ним символьной величины. Если регион является локальным для вызываемой функции, то он пропускается как несущественный. В противном случае он актуализируется, связанное с ним символьное значение также актуализируется, и связывается с актуализированным регионом в новом состоянии. Актуализация всех величин на этом шаге производится относительно одного и того же состояния; благодаря этому актуализация каждой связи в `Store` не влияет на актуализацию других связей. Это также объясняет, почему слияние `Store` выполняется последним. Так, при актуализации `SymbolRegionValue` необходимо иметь в текущем `Store` связанное с актуализированным регионом значение, записанное там до вызова вызываемой функции, а не после вызова.

4.3 Применение резюме на стороне проверяющих модулей

С точки зрения проверяющего модуля применение резюме является лишь еще одной функцей обратного вызова, позволяющей влиять на текущий анализ. В процессе применения резюме модулям доступна текущая вершина графа текущего анализа и конечная вершина графа резюме, соответствующая применяемой в этой вершине ветке. Как и в случае других функций обратного вызова, в результате применения резюме проверяющий модуль может никак не изменить текущее состояние, либо продолжить текущий анализ, добавив в него 0 (прервав тем самым анализ), 1 или более новых вершин, соотвествующих откорректированным им состояниям программы.

Проверяющий модуль также имеет возможность исследовать GDM состояния, сохранённого в конечной вершине ветви резюме. Это означает, что модуль имеет все необходимое для записи и воспроизведения всех внешних эффектов функции, проявляющихся во влиянии этой функции на содержимое раздела GDM, принадлежащего этому модулю: он может сохранить в состоянии программы подробный журнал внешних эффектов, а затем при применении резюме получить доступ к такому журналу и воссоздать все необходимые эффекты. Это позволит проверяющему модулю корректно продолжить анализ после вызова функции.

Проверяющий модуль также может выдать срабатывания непосредственно во время моделирования вызова функции, то есть во время применения резюме. Это связано с тем, что в отсутствие контекста проверяющий модуль обладает меньшим количеством информации, и не может сделать вывода о наличии дефекта, но в контексте вызывающей функции необходимая дополнительная информация может появиться.

В качестве мотивирующих примеров рассмотрим применение резюме функции `foo()` при анализе функций `bar()` и `baz()` в листинге 1 в присутствии проверяющего модуля, реализующего поиск двойных освобождений памяти. Функция `foo()` сама по себе не содержит дефекта. Однако, во время моделирования вызова функции `foo()` при анализе функции `bar()` должен быть выдан отчет о дефекте: функция `foo()` совершает повторное освобождение памяти, уже освобожденной ранее. Освобожденность памяти ранее в данном случае как раз и является дополнительной информацией, позволяющей судить о наличии дефекта. При анализе функции `baz()`, напротив, ни сама функция `foo()`, ни вызов функции `foo()` в контексте не содержат дефекта, однако функция `foo()` содержит внешний эффект — освобождение памяти, которое должно быть корректно промоделировано, иначе обнаружение дефекта при дальнейшем анализе оказывается невозможным.

Таким образом, проверяющий модуль, применяя резюме `foo()`, должен сделать две вещи: воссоздать внешние эффекты и произвести отложенные проверки. Проверяющий модуль может реализовать это, сохраняя в состоянии программы журнал всех операций выделения и освобождения памяти с символьными значениями указателей на эту память. В момент проведения каждой такой операции модуль будет запоминать эту операцию в журнале. При применении резюме модуль будет исполнять операции из журнала, извлекаемого из состояния, сохраненного в резюме, в том же порядке. Под исполнением операции будет пониматься, во-первых, проверка корректности операции и выдача сообщения о дефекте в случае некорректности, во-вторых учет эффекта операции на текущее состояние программы, то есть проставление отметок об освобожденности или неосвобожденности тех или иных указателей, и в-третьих заполнение журнала операций вызывающей функции, чтобы потом не потерять эту информацию при применении резюме вызывающей функции в контексте какой-либо третьей функции, в котором она сама станет вызываемой. Таким образом, решение о срабатывании может откладываться многократно. Заметим также, что журнал нельзя существенно упростить. Так, его нельзя сделать неупорядоченным, поскольку, например, разница между (`free()`, `free()`, `malloc()`) и (`free()`, `malloc()`, `free()`) слишком существенна.

Поддержка резюме была реализована для нескольких различных проверяющих модулей, отвечающих за нахождение таких дефектов, как целочисленные переполнения, запись в константную память, выход за границы массива, проблемы многопоточности, ошибки при работе с исключениями и корректность работы с вводом-выводом в файловые потоки. Во всех случаях она свелась к последовательному применению вышеописанной методики.

5. Составление отчета о дефекте

Одной из полезных функций CSA является составление подробного отчета о дефекте, показывающего не только место в коде, в котором происходит срабатывание проверяющего модуля, но и путь выполнения программы, предшествовавший срабатыванию. Проверяющие модули могут также активно участвовать в составлении отчета о дефекте, отмечая интересующие их события вдоль пути выполнения. Данный механизм существенно увеличивает полезность отчетов.

При анализе методом встраивания составление отчета о дефекте сводится к подъему от вершины графа выполнения, в которой произошло срабатывание, к корню графа. При проходе через вершину, в которой происходит интересное событие, такое как ветвление, вызов функции, или событие, важное с точки зрения проверяющего модуля, на позицию в коде, которая соответствует данной вершине, ставится дополнительное диагностическое сообщение.

Анализ методом резюме усложняет эту процедуру, и требует дополнительной, на этот раз сравнительно простой, поддержки на стороне проверяющих модулей. Вместо подъема по одному графу выполнения происходит склеивание пути из ветвей нескольких графов выполнения. Имеют место несколько возможных случаев.

Так, если путь выполнения обрывается в процессе применения резюме, как при анализе функции `bar()` в листинге 1, то отрезок пути от вершины, в которой производится отложенное срабатывание, до корня графа резюме добавляется в отчет о срабатывании в качестве конечного отрезка пути. При этом вершина, в которой производится отложенное срабатывание (в нашем примере это момент вызова `free()` в функции `foo()`), известна лишь проверяющему модулю; поэтому, выдавая отчет о дефекте в момент применения резюме, проверяющий модуль должен указать, в какой именно вершине резюме произошло срабатывание. Если отложенное срабатывание является многократно отложенным, то проверяющий модуль вынужден предоставить список конечных вершин во всем стеке функций.

Если же требуется построить отчет о дефекте сквозь промежуточную функцию, резюме которой было полностью применено в течение анализа, подобно тому, как отчет о дефекте в функции `baz()` в листинге 1 должен быть проложен сквозь функцию `foo()`, то вся ветвь резюме вызываемой функции целиком встраивается в отчет. Само встраивание отрезка пути не требует дополнительной поддержки на стороне проверяющих модулей. Однако, разумеется, дополнительный механизм, отвечающий за выдачу дополнительных диагностик, должен быть модифицирован с учетом особенностей анализа методом резюме. В нашем примере это требуется, поскольку вызов функции `free()` в `foo()` интересен. В этом случае вершины, содержащие интересные события, запоминаются в резюме, и при построении отчета при прохождении через эти вершины выдается

дополнительная диагностика. Запоминать весь стек вершин в этом случае не требуется.

6. Оценка масштабируемости метода.

Предварительный анализ практической эффективности и масштабируемости метода резюме по сравнению с методом встраивания производился посредством прогона анализа всей кодовой базы платформы Android [9].

Практическим показателем эффективности анализа является количество найденных дефектов. В данном случае исследовалось количество срабатываний проверяющих модулей, для которых был реализована поддержка метода резюме; доля реальных дефектов («истинных срабатываний») в обоих случаях составляла около 80%, независимо от метода анализа. Дефекты считаются одинаковыми, если выдается сообщение такого же вида в той же строчке кода; если один и тот же дефект найден в процессе анализа несколько раз, то он считается за один.

Табл. 1. Результаты измерения производительности метода резюме
Table 1. The results of performance measurement of the summary method

Ограничение max-nodes	2000	4000	8000	16000	32000	64000	128000
Время анализа	0:02	0:05	0:12	0:25	0:45	1:31	2:53
Число срабатываний	1457	1591	1696	1773	1834	1886	1911
Срабатываний в секунду	12.14	5.30	2.35	1.18	0.68	0.34	0.18
Корректных срабатываний	82%	83%	82%	83%	83%	82%	81%

Табл. 2. Результаты измерения производительности метода встраивания
Table 2. The results of performance measurement of the incorporation method

Ограничение max-nodes	8000	16000	32000	64000	128000	256000	512000
Время анализа	0:04	0:10	0:21	0:43	1:23	2:44	5:20
Число срабатываний	1506	1616	1703	1771	1828	1895	1933
Срабатываний в секунду	6.28	2.69	1.35	0.68	0.36	0.19	0.10
Корректных срабатываний	82%	82%	82%	81%	81%	81%	81%

В таблицах 1 и 2 приведены результаты измерения эффективности, производительности и масштабируемости анализа по этим двум показателям. Различные столбцы соответствуют изменениям основного параметра, контролирующего в CSA соотношение между скоростью анализа и покрытием путей выполнения — ограничением “max-nodes” на максимальный размер одного графа выполнения; это ограничение в случае метода резюме ограничивает размер графа резюме, а в при анализе методом встраивания — размер одного графа с учетом встроенных подграфов. Это означает, что при одинаковых значениях max-nodes анализ методом резюме будет происходить медленнее и покрывать больше путей выполнения программы; однако если уменьшить значение max-nodes для метода резюме (на практике — примерно в 4 раза), то удается получить сравнимое покрытие путей выполнения программы за существенно меньшее время, что и подтверждается экспериментальными данными.

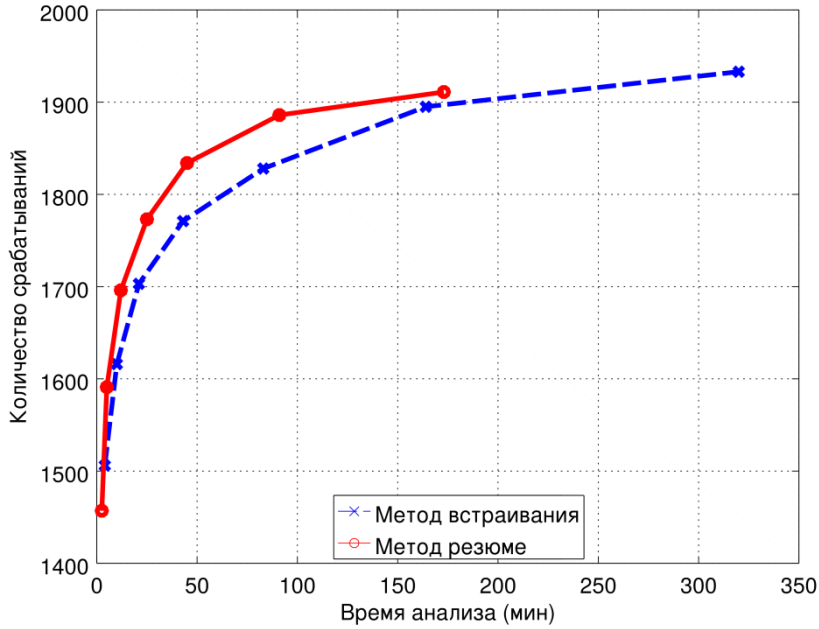


Рис. 3. Зависимость количества найденных дефектов от времени анализа при реализации межпроцедурного анализа методом встраивания и методом резюме
Fig. 3. The dependence of the number of defects found on the analysis run-time duration time in the implementations of interprocedural analysis by the incorporation and summary methods

На рисунке 3 изображены составленные по таблицам 1 и 2 графики зависимости количества дефектов, найденных различными методами, от продолжительности анализа. На графиках видно, метод резюме находит больше дефектов, чем метод встраивания, за то же время, либо позволяет найти аналогичное количество дефектов за существенно меньшее время.

7. Заключение.

В результате проведенного исследования был разработан метод межпроцедурного анализа, основанный на составлении и применении резюме функций, и обладающий существенным превосходством в производительности при анализе крупных программных проектов по сравнению с имеющимися методиками. Предлагаемый метод был реализован на базе статического анализатора Clang Static Analyzer, оперирующего исходным кодом на языках C и C++ и осуществляющим в процессе анализа символьное выполнение кода. Для данного анализатора ранее был реализован межпроцедурный анализ методом встраивания, что позволило провести сравнительный анализ двух подходов. Авторами статьи было продемонстрировано, что разработанный метод является достаточно мощным для анализа исходного кода на высокоуровневых языках программирования, и достаточно расширяемым для реализации различных видов проверок. Был разработан и реализован метод построения информативного отчета о найденных дефектах, учитывающий специфику метода резюме. Проведена оценка производительности и масштабируемости предложенного метода.

Список литературы

[1]. David A. Wheeler. How to Prevent the next Heartbleed. 2015. http://www.dwheeler.com/essays/heartbleed.html
[2]. John Carmack. Static Code Analysis. 2011. http://www.viva64.com/en/a/0087/
[3]. King, James C. Symbolic Execution and Program Testing. Commun. ACM, 19(7), 1976. P. 385–394.
[4]. Godefroid Patrice, Klarlund Nils, Sen Koushik. DART: Directed Automated Random Testing. Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI ’05. New York, NY, USA: ACM, 2005. P. 213–223.
[5]. Saswat Anand, Păsăreanu Corina S, Willem Visser. JPF-SE: A Symbolic Execution Extension to Java Pathfinder. International conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2007.
[6]. В.П. Иванников, А.А. Белеванцев, А.Е. Бородин и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды Института системного программирования РАН. 2014, 26 (1). С. 231–250.
[7]. Matsumoto Hiroo. Applying Clang Static Analyzer to Linux Kernel. 2012 LinuxCon Japan. Yokohama: 2012. 6.
[8]. Cadar Cristian, Dunbar Daniel, Engler Dawson. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. Proceedings of the

8th USENIX Conference on Operating Systems Design and Implementation. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008. P. 209–224.

[9]. Android Open Source Project. http://source.android.com/

[10]. Reps Thomas, Horwitz Susan, Sagiv Mooly. Precise Interprocedural Dataflow Analysis via Graph Reachability. Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '95. New York, NY, USA: ACM, 1995. P. 49–61.

[11]. Компиляторы. Принципы, технологии и инструментарий. А.В. Ахо, М.С. Лам, Рави Сети, Д.Д. Ульман. Вильямс, 2003. — 1184 с.

[12]. Rojas José Miguel, Păsăreanu Corina S. Compositional Symbolic Execution through Program Specialization. BYTECODE 2013, 8th Workshop on Bytecode Semantics, Verification, Analysis and Transformation, 2013.

[13]. Godefroid Patrice. Compositional Dynamic Test Generation. SIGPLAN Not. 2007. 42(1). P. 47–54.

[14]. Summary-based inference of quantitative bounds of live heap objects. Víctor Brabermana, Diego Garbervetskya, Samuel Hymc, Sergio Yovinea Science of Computer Programming, 92, 2013. P. 56–84.

[15]. Xu Zhenbo, Zhang Jian, Xu Zhongxing. Melton: a practical and precise memory leak detection tool for C programs. Frontiers of Computer Science in China, 9(1), 2015. P. 34–54.

[16]. Clang Static Analyzer. http://clang-analyzer.llvm.org/

[17]. Strom, Robert E.; Yemini, Shaula. Typestate: A programming language concept for enhancing software reliability. IEEE Transactions on Software Engineering (IEEE), 12, 1986. P. 157–171.

[18]. Xu Zhongxing, Kremenek Ted, Zhang Jian. A Memory Model for Static Analysis of C Programs. Proceedings of the 4th International Conference on Leveraging Applications of Formal Methods, Verification, and Validation. Volume Part I. ISoLA'10. Berlin, Heidelberg: Springer-Verlag, 2010. P. 535–548.

Summary-based method of implementing arbitrary context-sensitive checks for source-based analysis via symbolic execution

A. Dergachev <dergachev.a@samsung.com>
A. Sidorin <a.sidorin@samsung.com>
Samsung R&D Institute Russia,
Dvintsev 12, 127018 Moscow, Russia

Abstract. A specific approach to summary-based interprocedural symbolic execution is described. The approach is suitable for analysis of program source code developed with high-level programming languages and allows executing arbitrarily complex checks during symbolic execution, including throwing reports in the callee function about defects that only become certain within the caller context. The structure of the function summary, procedure of applying the summary in a particular context, composition of symbolic values for particular contexts, effect of summary-based analysis on complexity of implementing specific checker modules, procedure for constructing path-sensitive bug reports, and other aspects of the implementation are discussed in detail. A particular implementation of the approach, based

on Clang Static Analyzer, is described. The implementation is scalable enough to allow analysis of large-scale software projects in reasonable time, finding bugs faster than the existing implementation of the inlining-based interprocedural analysis, without sacrificing correctness and soundness of the analysis. Particular checker modules, which find various defects, such as integer overflows, modifications of constant-qualified memory, multithreading issues, array bound checks, exception safety checks, and file stream errors, were updated to use the summary-based approach, demonstrating flexibility of the technique proposed. The implementation was tested by running full intra-unit inter-procedural analysis of the Android Open Source Project codebase.

Keywords: static analysis, symbolic execution, interprocedural analysis, context-sensitive analysis, summary-based analysis, C, C++, Clang Static Analyzer.

DOI: 10.15514/ISPRAS-2016-28(1)-3

For citation: Dergachev A.V., Sidorin A.V.. Summary-based method of implementing arbitrary context-sensitive checks for source-based analysis via symbolic execution. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 1, 2016, pp. 41-62 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-3

References

[1]. David A. Wheeler. How to Prevent the next Heartbleed. 2015. http://www.dwheeler.com/essays/heartbleed.html
[2]. John Carmack. Static Code Analysis. 2011. http://www.viva64.com/en/a/0087/
[3]. King James C. Symbolic Execution and Program Testing. Commun. ACM, 19(7), 1976. P. 385–394.
[4]. Godefroid Patrice, Klarlund Nils, Sen Koushik. DART: Directed Automated Random Testing. Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI '05. New York, NY, USA: ACM, 2005. P. 213–223.
[5]. Saswat Anand, Păsăreanu Corina S, Willem Visser. JPF-SE: A Symbolic Execution Extension to Java Pathfinder. International conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2007.
[6]. V.P. Ivannikov, A.A. Belevancev, A.E. Borodin et. al. Staticheskij analizator Svace dlja poiska defektov v ishodnom kode program. [Static analyzer Svace for finding defects in a source program code]. Trudy ISP RAN [Proceedings of ISP RAS]. 2014, vol. 26, issue 1, pp. 231–250 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-7
[7]. Matsumoto Hiroo. Applying Clang Static Analyzer to Linux Kernel. 2012 LinuxCon Japan. Yokohama: 2012. 6.
[8]. Cadar Cristian, Dunbar Daniel, Engler Dawson. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008. P. 209–224.
[9]. Android Open Source Project. http://source.android.com/
[10]. Reps Thomas, Horwitz Susan, Sagiv Mooly. Precise Interprocedural Dataflow Analysis via Graph Reachability. Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium

on Principles of Programming Languages. POPL '95. New York, NY, USA: ACM, 1995. P. 49–61.

[11]. Compilers: Principles, Techniques, and Tools (2nd Edition). A.V. Aho, M.S. Lam, Ravi Sethi, D.D. Ullman. Pearson Education, Inc, 2006.

[12]. Rojas José Miguel, Păsăreanu Corina S. Compositional Symbolic Execution through Program Specialization. BYTECODE 2013, 8th Workshop on Bytecode Semantics, Verification, Analysis and Transformation, 2013.

[13]. Godefroid Patrice. Compositional Dynamic Test Generation. SIGPLAN Not. 2007. 42(1). P. 47–54.

[14]. Summary-based inference of quantitative bounds of live heap objects. Víctor Brabermana, Diego Garbervetskya, Samuel Hymc, Sergio Yovinea Science of Computer Programming, 92, 2013. P. 56–84.

[15]. Xu Zhenbo, Zhang Jian, Xu Zhongxing. Melton: a practical and precise memory leak detection tool for C programs. Frontiers of Computer Science in China, 9(1), 2015. P. 34–54.

[16]. Clang Static Analyzer. http://clang-analyzer.llvm.org/

[17]. Strom, Robert E.; Yemini, Shaula. Typestate: A programming language concept for enhancing software reliability. IEEE Transactions on Software Engineering (IEEE), 12, 1986. P. 157–171.

[18]. Xu Zhongxing, Kremenek Ted, Zhang Jian. A Memory Model for Static Analysis of C Programs. Proceedings of the 4th International Conference on Leveraging Applications of Formal Methods, Verification, and Validation. Volume Part I. ISoLA'10. Berlin, Heidelberg: Springer-Verlag, 2010. P. 535–548.

Оптимизация динамической загрузки библиотек на архитектуре ARM

Е.А. Кудряшов <eugene.a.kudryashov@gmail.com>
Д.М. Мельник <dm@ispras.ru>
А.В. Монаков <amonakov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В статье рассматривается подход к оптимизации вызовов внешних функций в позиционно-независимом коде, основанный на выдаче вызовов непосредственно через глобальную таблицу смещений (GOT), минуя таблицу компоновки процедур (PLT). Стандартные механизмы кодогенерации на операционной системе Linux предполагают создание PLT не только для основного модуля (который является позиционно-зависимым и полагается на механизм PLT для вызовов внешних процедур), но и для динамических библиотек, где PLT используется также для организации ленивого связывания; однако, использование PLT требует дополнительной инструкции перехода, может иметь низкую локальность по кешу и на некоторых архитектурах накладывает дополнительные ограничения на работу компилятора в месте вызова. Реализация вызовов внешних функций в виде косвенных переходов на адреса, загруженные непосредственно из GOT в месте вызова, позволяет избежать недостатков вызовов через PLT, ценой отказа от возможности ленивого связывания и, возможно, увеличением размера кода. Была исследована реализация этой оптимизации для архитектур x86 и ARM в компиляторе GCC. Было обнаружено, что на архитектуре ARM отсутствуют типы релокаций, которые позволили бы генерировать оптимальный код для загрузок из GOT. Для решения этой проблемы в GCC и Binutils (в ассемблере и компоновщике) были реализованы недостающие типы релокаций, позволяющие построить адрес позиции в GOT относительно счетчика команд, используя инструкции movt, movw. Проведенное тестирование свидетельствует, что предложенная оптимизация позволяет получить увеличение производительности, несмотря на увеличение размеров динамических библиотек.

Ключевые слова: оптимизация программ; динамический загрузчик; глобальная таблица смещений; таблица компоновки процедур; релокация; архитектура ARM

DOI: 10.15514/ISPRAS-2016-28(1)-4

Для цитирования: Кудряшов Е.А., Мельник Д.М., Монаков А.В. Оптимизация динамической загрузки библиотек на архитектуре ARM. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 63-80. DOI: 10.15514/ISPRAS-2016-28(1)-4

1. Введение

Операционные системы обычно поддерживают два типа библиотек: статические, которые становятся частью загружаемого образа программы при ее сборке, и динамические, которые загружаются в память отдельно (если их там нет) при запуске программы.

Выбор между статическими или динамическими библиотеками требует компромиссов. Использование динамических библиотек позволяет сократить расходование оперативной памяти, когда запущены одновременно несколько различных программ, использующих один набор библиотек, но это достигается ценой как правило более сложного позиционно-независимого кода (PIC) в библиотеках [1], и сложным процессом динамического связывания библиотек.

Загрузка и связывание динамических библиотек выполняется динамическим загрузчиком, который получает управление после того, как ядро операционной системы разместило в памяти программу. Перед началом выполнения программы он выполняет поиск и загрузку всех файлов с динамическими библиотеками, которые использует программа. Загрузка выполняется на заранее неизвестные адреса, и поэтому часть данных в загруженных образах библиотек подвергается процедуре перемещения: динамический загрузчик должен переписать часть загруженных данных в зависимости от адреса, на который загружена библиотека, в соответствии со специальными аннотациями – релокациями. Кроме того, библиотека может использовать функции из других загруженных модулей, и для них нужно выполнить динамическое связывание, то есть подставить конкретные адреса в соответствие символическим именам. Существует два подхода к решению данной задачи: разрешение ссылок во время запуска или во время исполнения. Оба имеют негативные эффекты: разрешение во время запуска увеличивает время запуска программы, разрешение во время исполнения создаёт накладные расходы на каждый вызов функции.

Цель данной работы – оптимизировать взаимодействие программ с динамическими библиотеками в режиме PIC, с помощью поиска компромисса между временем запуска программы и накладными расходами на вызов функций. В дополнении к этому планируется добиться генерации более эффективного кода посредством ввода новых типов релокаций.

Дальнейшее изложение будет построено следующим образом. Сначала будет описано использование позиционно-независимого кода в динамических библиотеках. После этого следует описание оптимизации и её особенности реализации на архитектуре ARM. В заключении будут приведены результаты оптимизации, протестированные на встраиваемой реляционной базе SQLite.

2. Особенности позицинно-независимого кода

Режим позиционно-независимого кода подразумевает возможность загрузки модуля по произвольному базовому адресу без изменений загруженного

программного кода (именно это позволяет разделять одну копию кода библиотеки между всеми использующими ее процессами). Это означает, что все инструкции, обращающиеся к динамическим адресам, не содержат их как операнд, а загружают их косвенно из области данных, и все динамические релокации применяются к секции с данными, а не с кодом.

2.1 Смещение относительно text и data секций

Одним из ключевых аспектов PIC является то, что код полагается на разницу в адресном пространстве между text и data секциями программы, которые известны компоновщику. Когда он соединяет несколько объектных файлов вместе, он собирает все их секции (т.е. все text секции различных файлов будут собраны в одну большую). Следовательно, компоновщик имеет информацию как о размерах секций, так и об их относительном положении.

Например, data секция может идти непосредственно сразу после text секции, тогда смещение любой инструкции в text секции до начала data секции будет составлять длину text секции за вычетом смещения данной инструкции от начала text секции. Обе составляющие (длина text секции и смещение инструкции относительно начала text секции) известны компоновщику.

На рис. 1 text секция была загружена по определенному адресу (неизвестный во время компоновки) 0xXXXX0000 (где XXXX – некий адрес, который в данном контексте не имеет значения), сразу после неё была загружена data секция со смещением 0xXXXXA000. Если некой инструкции со смещением 0x100 в text секции понадобится что-то в data секции, компоновщик знает относительный сдвиг (в данном случае 0x9F00) и может использовать это в инструкции.

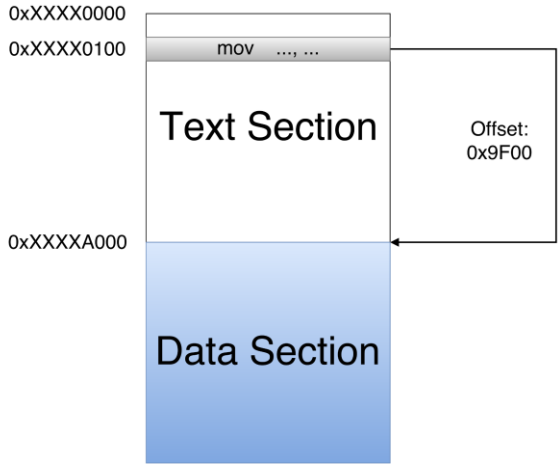


Рис. 1. Смещения относительно text и data секций

Fig. 1. Offsets between text and data sections

Стоит заметить, что компоновщик сможет высчитать верное смещение и в случаях, когда имеются дополнительные секции между text и data, и в случаях, когда data предшествует text секции: достаточно лишь того, что секции не перемещаемы друг относительно друга после завершения компоновки.

2.2 Глобальная таблица смещений

Глобальная таблица смещений (Global Offset Table, GOT) это таблица абсолютных адресов [1]. Когда инструкция кода обращается к глобальной переменной, то вместо обращения по абсолютному адресу (которое требует релокации в text секции), она обращается к GOT по относительному. Таким образом, смещение в коде относительное, и оно известно компоновщику. На рис. 2 представлена схема GOT с адресами на глобальные переменные.

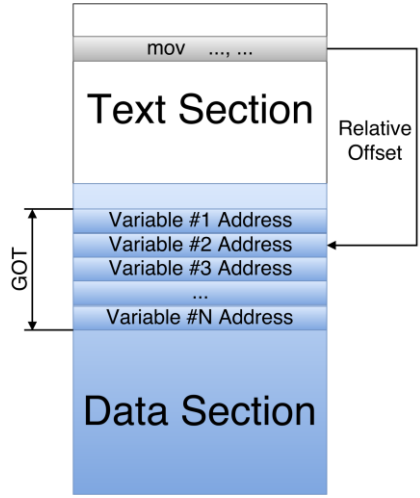


Рис. 2. Обращение к глобальной таблице смещений

Fig. 2. Reference to the the Global Offset Table

Данный метод позволяет вынести релокации из секции кода в секцию data путём перенаправления ссылок на переменные через GOT.

2.3 Вызов функций в режиме PIC

До этого речь шла о взаимодействии с переменными в режиме PIC. Теоретически, точно такой же подход можно использовать и для работы с вызовом функций: глобальная таблица смещений будет содержать адреса функций, которые будут заполняться во время загрузки программы. Но в действительности вызов функций работает иначе и немного сложнее.

2.3.1 Ленивое связывание

Когда программа ссылается на некоторую внешнюю функцию, то настоящий адрес данной функции неизвестен во время загрузки. Определение и присваивание данного адреса называется связыванием, и этим занимается динамический загрузчик, когда динамическая библиотека загружается в память. Процесс связывания требует времени, т.к. загрузчик должен пройтись по специальным таблицам в поисках необходимой функции. Время для поиска одной функции не велико, но библиотеки обычно содержат большое количество функций. Кроме того, большинство из этих поисков будет выполняться напрасно, потому что программы редко используют полный функционал библиотек. Также не стоит забывать о различных функциях обработки ошибок, которые, как правило, не вызываются вовсе.

Для ускорения данного процесса был разработан подход ленивого связывания. Прилагательное «ленивое» является обобщающим названием оптимизаций, в которых выполнение операций откладывается на последний момент, когда её выполнение действительно необходимо. Данный подход позволяет избежать выполнения ненужных операций, которые никогда не понадобятся во время исполнения программы, однако это достигается ценой усложнения динамического загрузчика и опасностью аварийного завершения программы, если при ленивом связывании нужный символ не может быть найден. Ленивое связывание реализуется с помощью таблицы компоновки процедур.

2.3.2 Таблица компоновки процедур

Таблица компоновки процедур (Procedure Linkage Table, PLT) – часть секции text [1]. Она состоит из множества небольших одинаковых фрагментов исполняемого кода, по одному для каждой внешней функции. Вместо вызова функции напрямую происходит перенаправление на код из таблицы процедур, который совершает вызов необходимой функции. Данный участок кода иногда называется «трамплином». В случае, если используется ленивое связывание, трамплин в случае первого вызова функции разрешает и вызывает её, иначе сразу вызывает её, т.к. разрешение уже было произведено и абсолютный адрес известен. Использование трамплина для каждой функции обуславливается необходимостью загрузки адреса функции и подготовкой аргументов для её разрешения.

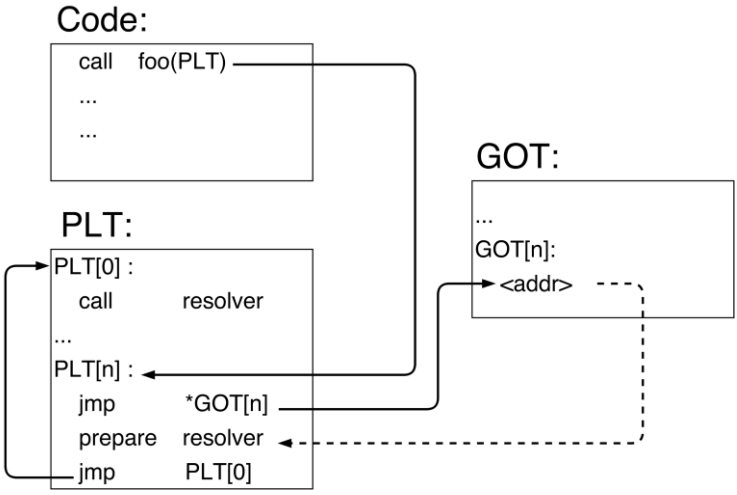


Рис. 3. Первый вызов функции из динамической библиотеки

Fig. 3. First invocation of a function from a dynamic library

Процесс первого вызова функции изображен на рис. 3, рассмотрим его детальнее:

- когда происходит вызов внешней функции *foo*, то компилятор заменяет данный вызов на вызов кода в PLT (в данном примере *foo(PLT)*);
- таблица компоновки процедур состоит из специального нулевого элемента (о нём будет сказано позже) и из последовательности структурно одинаковых фрагментов кода; каждый фрагмент кода отвечает за динамическую загрузку определенной функции;
- каждый элемент PLT, кроме нулевого, состоит из следующих частей:
  - переход по соответствующему адресу глобальной таблицы смещений;
  - подготовка аргументов для процедуры *resolver;*
  - переход на нулевой элемент PLT;
- нулевой элемент PLT – это вызов процедуры *resolver*, которая расположена в самом динамическом загрузчике; данная процедура занимается получением абсолютного адреса функций;

- до того, как адрес функции будет разрешен, *n*-ый элемент глобальной таблицы смещений имеет адрес следующей инструкции в PLT; именно поэтому на рис. 3 стрелка обозначена пунктиром.

При вызове *foo* в первый раз происходит следующее:

- вызывается PLT[*n*], и происходит переход на адрес, расположенный в GOT[*n*];
- адрес указывает на следующую инструкцию в PLT[*n*], для подготовки аргументов процедуры *resolver*;
- вызывается процедура *resolver*;
- *resolver* получает абсолютный адрес *foo*, и помещает его в GOT[*n*] и вызывает *foo*.

При последующих вызовах *foo*, в силу ленивого связывания, картина изменится (см. рис. 4):

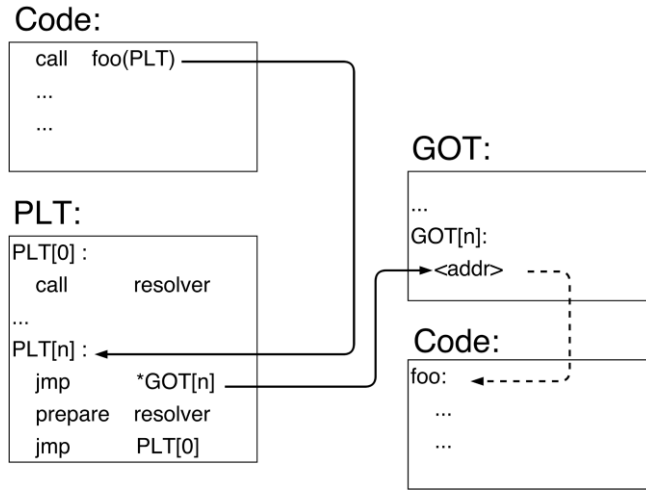


Рис. 4. Последующие вызовы функций после связывания

Fig. 4. Subsequent invocations of bound dynamic functions

- вызывается PLT[*n*], и происходит переход на адрес, расположенный в GOT[*n*];
- GOT[*n*] указывает на *foo*, соответственно происходит вызов *foo*.

Другими словами, все последующие вызовы *foo* происходят в обход процедуры *resolver* и стоят одного дополнительного перехода.

Описанный подход позволяет использовать ленивое связывание функций. Также он делает text секцию позиционно независимой, т.к. единственное место, где используется абсолютный адрес – это в глобальной таблице смещений, которая расположена в data секции.

3. Отказ от использования PLT

Хотя использование PLT необходимо в программных модулях, скомпонованных из позиционно-зависимого кода (в них все места вызова используют абсолютный адрес, на место которого компоновщик вынужден подставить адрес PLT-трамплина), в коде динамических библиотек использовать PLT не обязательно: как отмечалось раньше, позиционно-независимый код мог бы загружать адрес вызываемых функций непосредственно из GOT-таблицы. Выгода от использования PLT заключается исключительно в возможности ленивого связывания функций. Но для программ, где производительность является критически важной, в этом нет особой необходимости. Исключение составляет случай, когда программа используется очень часто, её время исполнения крайне невелико, и большая часть функций не используется: тогда полное динамическое связывание при каждом запуске программы будет вносить заметный вклад в общее время работы; однако в таком случае статическая компоновка позволит существенно ускорить загрузку.

С другой стороны, вызовы через PLT требуют выполнения дополнительных инструкций, и переход на трамплин может не обладать локальностью по кэшу инструкций. Таким образом, генерация кода, который выполняет вызов внешних функций непосредственно через GOT, минуя PLT, может быть оптимальнее.

Оптимизация заключается в следующем: когда компилятор встречает вызов внешней функции, то он не формирует обычный вызов, который будет преобразован в PLT вызов (см. рис. 3, *foo(PLT)*), а записывает в регистр адрес функции из глобальной таблицы смещений и делает вызов функции по регистру. Для этого необходимо, чтобы GOT была уже проинициализирована, т.е. она заполняется адресами функций во время запуска. Как следствие, необходимость в ленивом связывании пропадает.

Данная оптимизация была реализована нами на архитектурах x86, x86-64, позднее она появилась и на архитектуре AArch64 (64-битный ARM) [2]. Однако, реализация данной оптимизации на архитектуре ARM имеет существенные отличия.

4. Устранение PLT-переходов на x86-архитектурах

Изначально вариант генерации кода, избегающий использования PLT, был реализован в GCC для процессорных архитектур IA-32 и AMD64. Было отмечено, что в компиляторе присутствует функциональность, называемая «function CSE», решающая похожую задачу: обеспечить возможность оптимизации нескольких вызовов одной и той же функции путем загрузки адреса этой функции на регистр; это выделяет взятие адреса функции как отдельную инструкцию во внутреннем представлении компилятора и позволяет автоматически удалить дублирующиеся загрузки в ходе дальнейших оптимизационных проходов. Для большинства архитектур эта функциональность не активна, вероятно из-за исключительной редкости случаев, когда за счет нее возможно существенное улучшение кода.

Соответственно, при выдаче RTL-кода для вызовов функций, при активных флагах -fno-plt и -fPIC, вызов выдается как последовательность двух RTL-инструкций: первая вычисляет абсолютный адрес вызываемой функции в псевдорегистр, а вторая выполняет косвенный вызов по этому регистру. Поскольку генерируется PIC-код, первая инструкция соответствует загрузке из таблицы GOT (в позиционно-зависимом коде это было бы записью абсолютного адреса непосредственно в регистр).

Для 32-битных x86 архитектур помимо исключения перехода на PLT-трамплин есть и второй фактор улучшения кода: согласно соглашению о вызовах, при переходе на PLT-трамплин, регистр ebx должен указывать на GOT-таблицу. Поскольку на x86 нет относительной адресации относительно текущей инструкции (регистра eip), PLT-трамплины используют ebx как базу для загрузки целевого адреса из GOT. Однако ebx является call-saved регистром: вызываемая функция должна восстанавливать его значение перед возвратом. Таким образом, оптимизация хвостовых вызовов при переходе на PLT-трамплин невозможна. Использование же переходов через GOT позволяет использовать любой регистр общего назначения в качестве базы для позиционно-независимой адресации, так что для хвостовых переходов компилятору достаточно распределить адрес GOT на стираемый при вызовах регистр, например, eax. Кроме того, открываются и другие возможности оптимизации за счет переноса инструкции загрузки из GOT: вынос из циклов, планирование, удаление избыточности.

При разработке и тестировании этой оптимизации были обнаружены и исправлены упущения в компиляторе. В частности, порядок перечисления регистровых классов был неоптимален для 32-битного x86, что иногда не позволяло хорошо выбрать регистр для адресации GOT. При выдаче хвостовых переходов без надобности была запрещена косвенная адресация по регистру eax.

Для оценки улучшения производительности использовался Clang/LLVM, скомпилированные так, что динамически загружается более 100 библиотек с более 24000 динамических символов. Для компиляции тривиального C++

файла, Clang с -fno-plt на 20% медленнее, чем базовый, из-за отсутствия ленивого связывания, но уже на 10% быстрее, когда оно запрещено. При компиляции большого файла наблюдается существенное (10-12%) ускорение [3].

5. Применение на архитектуре ARM

Существует несколько особенностей, которые препятствуют применению данной оптимизации на архитектуре ARM:

- обратная оптимизация на этапе combine;
- отсутствие необходимых типов релокаций для генерации эффективного кода.

5.1 Combine

Особенность на этапе combine заключается в следующем: после того, как была произведена запись адреса функции из глобальной таблицы в регистр (инструкция вызова функции разделилась на несколько инструкций), происходит объединение данных инструкций в одну, которая была изначально.

Combine – это оптимизационный проход в компиляторе GCC, выполняющий объединение нескольких инструкций в одну. Могут объединяться как две инструкции, так и триплеты и даже четвёрки команд. Комбинирование происходит подстановкой значений регистров в более поздних инструкциях, которые ссылаются на данные регистры. Если итогом такого комбинирования является допустимая инструкция (инструкция подошла к какому-либо из шаблонов инструкций), то результат сохраняется, с удалением предшествующих инструкций (из которых подставлялись значения регистров) и обновлением информации о потоке данных [4].

Когда компилятору необходимо сделать вызов по регистру, то данный регистр получает пометку о том, что он эквивалентен ссылке на функцию (данная пометка может использоваться в других оптимизациях). Тогда, combine видит, что есть инструкция по помещению адреса функции в регистр, а затем вызов функции по этому регистру, и он совершенно справедливо пытается объединить их в одну инструкцию вызова функции, помещая значения регистра в аргумент функции вызова. Данная команда корректна, комбинирование завершается успешно, и в итоге снова получается вызов через PLT, которого хотели избежать.

Решение данной проблемы было заимствовано у архитектуры AArch64 [2]: во время проверки допустимости инструкции вызова функции были добавлены условия:

- компилируется PIC;

- не используется PLT для всех или для конкретно данного вызова;
- происходит вызов внешней функции.

Выполнение данных условий означает, что функция должна быть вызвана через регистр, а это значит, что скомбинированная инструкция не подойдёт к шаблонам вызова и combine не сможет их объединить в одну.

5.2 Генерация эффективного кода

Архитектура ARM имеет несколько систем команд. Одна из них это стандартная ARM, в данной системе используются только 32-битные команды. Система команд THUMB состоит из команд, взятых из ARM и преобразованных до 16-разрядных кодов. Данный режим сокращает объем используемой памяти, повышая плотность компилируемого кода. Этот набор команд имеет ограничение: работать напрямую можно только с 8-ю младшими регистрами общего назначения. Компромиссом между режимами ARM и THUMB является система команд THUMB-2, в которой сохранены 16-битные инструкции для повышения плотности кода, а также добавлены 32-битные инструкции, которые позволят приблизиться к производительности полного 32-битного набора команд ARM. Далее будут рассмотрены системы команд ARM и THUMB-2.

5.2.1 Система команд ARM

После исправления обратного комбинирования, получаем необходимый эффект (см. рис. 5).
Рассмотрим данный код немного детальнее:

- в регистр *r3* загружается смещение до глобальной таблицы смещений относительно метки *.LPIC0* (в данной метке это смещение будет сложено с регистром *pc*; таким образом будет получено смещение до глобальной таблицы смещений);
- в регистр *r2* загружается смещение функции *foo* в таблице смещений;
- в регистр *r3* помещается адрес функции *foo*.

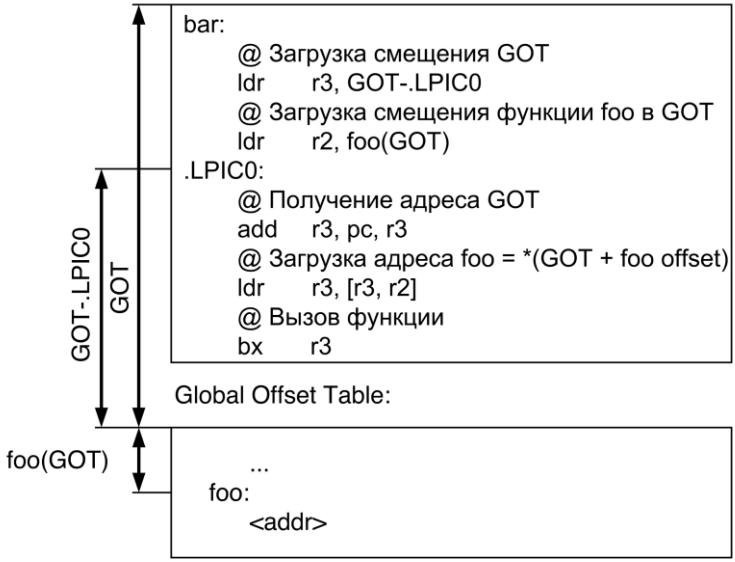


Рис. 5. Загрузка адреса функции и вызов

Fig. 5. Loading the address of the function prior to call

Таким образом мы получаем загрузку функций с помощью GOT без использования PLT. Но полученный код не выглядит оптимальным: на вызов функции необходимо выполнить три инструкции загрузки из памяти, которые являются дорогостоящей операцией.

Как было упомянуто ранее, компоновщик знает размеры секций и их смещения. Но данная информация никак не используется при генерации ассемблерного кода. Ввод релокации с адресом смещения до ячейки функции в глобальной таблице смещений невозможен, т.к. в архитектуре ARM нельзя записать 32-битную константу в регистр за одну инструкцию. Поэтому мы будем использовать две инструкции:

- *movw* (от англ. move wide) – помещает 16 битовую константу в регистр, обнуляя 16 старших битов регистра назначения;
- *movt* (от англ. move top) – помещает 16 битовую константу в 16 старших битов регистра назначения, при этом не изменяя младшие 16 битов.

```
.equ    label, 0x12345678
movw    r0, #:lower16:label
movt    r0, #:upper16:label
```

Рис. 6. Использование метки в инструкциях movw/movt

Fig. 6. Using a label in movw/movt instructions

Поводом для использования данных инструкций является возможность загрузки значения через метку (см. рис. 6). Данный код говорит о наличии функционала, который может быть использован при реализации релокаций. Причем данный функционал присутствует как в компиляторе, так и в компоновщике. Необходимо помнить, что использование инструкций *movw* / *movt* накладывает ограничение на версию архитектуры – они были введены сравнительно недавно и доступны на архитектурах ARMv6T2 и выше.

Заметим, что порядок данных команд имеет значение. Таким образом, потребуется две новые релокации, которые линкер должен будет разрешить: первая необходима для предоставления младших 16 бит смещения до ячейки функции в GOT, вторая – старших 16 бит.

На рис. 7 представлен псевдокод, в стиле ассемблера архитектуры ARM. Рассмотрим его подробнее:

- *#:lower16:got:foo* – младшие 16 битов смещения относительно текущего положения до ячейки функции *foo* в GOT;
- *#:upper16:got:foo* – старшие 16 битов смещения относительно текущего положения до ячейки функции *foo* в GOT;
- символ «.» означает смещение относительно начала программы до текущей исполняемой программы, аналогично метке *.LPIC0,* которая содержит смещение до инструкции, следующей за ней;
- разность (*.LPIC - .)* даёт смещение относительно текущего положения инструкции до инструкции, в которой данный адрес будет использоваться (в нашем случае *ldr r3, [pc, r3]*).

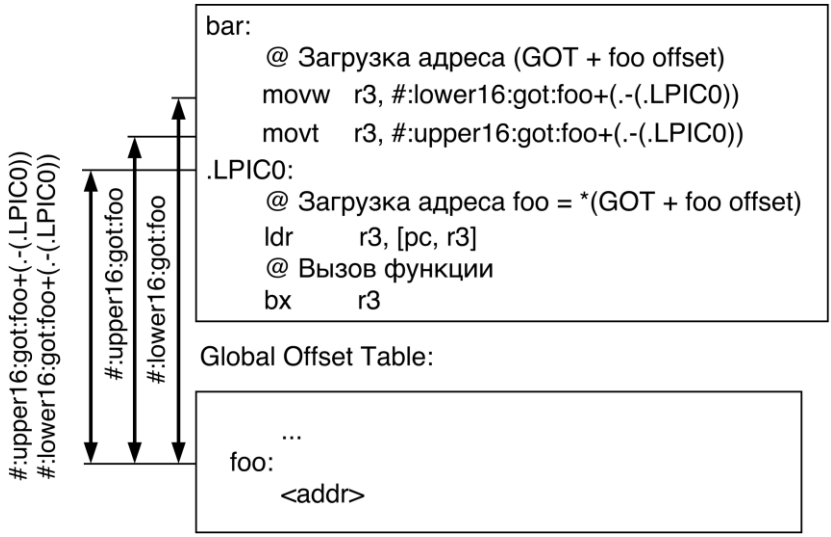


Рис. 7. Оптимизированная загрузка адреса функции

Fig. 7. Optimized function address load sequence

Таким образом, с помощью двух инструкций получилось избавиться от двух загрузок из памяти и одного сложения во время исполнения за счет расчета адресов ячеек в глобальной таблице смещений во время компоновки.

5.2.2 Система команд THUMB-2

Изначальная генерация кода схожа с системой команд ARM, и мало чем отличается от той, что представлена на рис. 5. Инструкции *movw* / *movt* также имеются в наборе команд. Но возникает проблема другого рода. Если при загрузке адреса ранее использовалась команда *ldr r3, [pc, r3]*, то в режиме THUMB-2 она невозможна. Согласно спецификации, при сложении с регистром *pc*, в качестве первого слагаемого, возможен только следующий случай: *ldr r0, [PC, imm12]*, где *imm12* – константное 12-битовое значение. В случае же если мы захотим поменять порядок сложения воспользовавшись коммутативностью, то результат выполнения команды вида *ldr r0, [r0, pc]* является неопределенным и непредсказуемым [5].

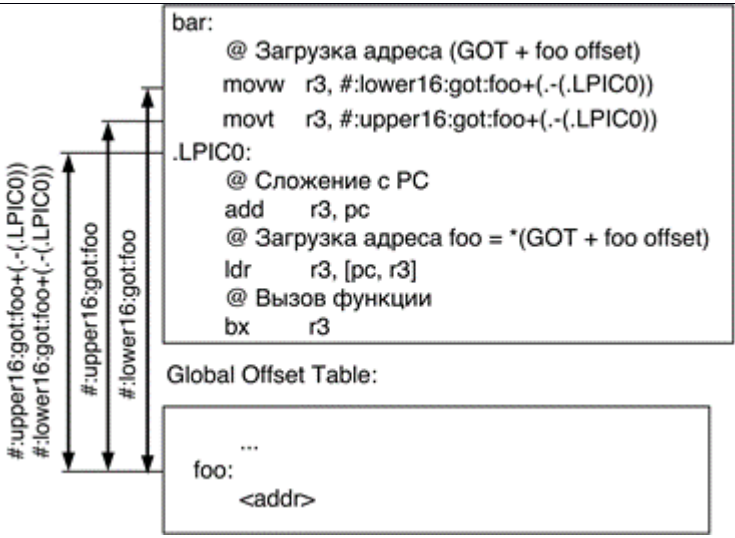


Рис. 8. Оптимизированная загрузка в режиме THUMB-2

Fig. 8. Optimized function address load sequence for THUMB2

Для решения данной проблемы необходимо использовать дополнительное сложение. На рис. 8 представлен псевдокод оптимизированной загрузки.

6. Результаты тестирования

Для тестирования было решено использовать встраиваемую реляционную базу данных SQLite [6], собранную в режиме динамической библиотеки. Во всех тестовых случаях были включены следующие флаги:

-O2 -march=armv7-a -mtune=cortex-a9 -mfpu=neon -mfloat-abi=softfp.

Для тестирования использовался стандартный набор тестов, поставляемый с исходными кодами SQLite и содержащий все основные операции с базой данных: INSERT, SELECT по числовым и строковым значениям, UPDATE числовых и строковых значений, INSERT таблиц друг в друга, сложные SELECT запросы.

В табл. 1 приведено среднегеометрическое время выполнения тестов, а также улучшения по отношению к базовому случаю – PLT. В табл. 2, аналогично табл. 1, приведены размеры динамической библиотеки при тестах.

Табл. 1. Результаты тестирования на SQLite. Время выполнения в секундах (среднегеометрическое по тестам).
В скобках указано улучшение по сравнению с базовым вариантом (с PLT).

Table 1. Evaluation results on SQLite. Times in seconds (geometric mean over multiple individual tests).
In parenthesis: improvement relative to base time (using PLT).

Набор команд	PLT	Без PLT	Без PLT с оптимизацией
ARM	1.28	1.26 (1.6%)	1.19 (7.0%)
THUMB-2	1.30	1.31 (-0.8%)	1.21 (6.9%)

Рассмотрим сначала ARM режим: время исполнения, как и предполагалось, уменьшается при отказе от PLT, причем, оно уменьшается, даже если каждый вызов функции будет требовать три загрузки из памяти вместо одного прыжка по таблице компоновки процедур. Когда включается оптимизация вызова через инструкции *movw* / *movt* происходит ускорение на 7.0%.

Табл. 2. Результаты тестирования на SQLite. Размер динамической библиотеки в килобайтах.
В скобках указано улучшение по сравнению с базовым вариантом.

Table 2. Evaluation results on SQLite. Dynamic shared library size, in kilobytes.
In parenthesis: improvement relative to base size.

Набор команд	PLT	Без PLT	Без PLT с оптимизацией
ARM	548	629 (-14.8%)	621 (-13.3%)
THUMB-2	400	445 (-11.3%)	458 (-14.5%)

С размером кода не так всё однозначно, после отказа от использования PLT размер динамической библиотеки растёт, это связано с тем, что теперь вместо одной инструкции прыжка на PLT таблицу генерируется 5 инструкций, размер кода увеличивается на 14.8%, после применения оптимизации кода на один вызов генерируется 4 инструкции, соответственно, размер кода, относительно PLT, увеличивается на 13.3%.

При использовании команд THUMB-2, отключение PLT приводит к незначительному замедлению, но после оптимизации кода даёт ускорение на 6.9%. Размер кода также увеличивается во всех случаях, но в отличии от ARM, не удалось сократить количество инструкций, получилось лишь их поменять на менее дорогостоящие по времени исполнения.

7. Заключение

В рамках данной работы проведён анализ работы позиционно-независимого кода и динамического загрузчика в операционной системе Linux с использованием компилятора GCC. Была рассмотрена оптимизация по отказу от использования таблицы компоновки процедур для вызовов функций. Также

были обнаружены трудности в реализации данной оптимизации на архитектуре ARM.

Проблемы были преодолены с помощью улучшения оптимизации комбинирования инструкций, а также добавления новых релокаций в GCC и GNU Binutils для генерации оптимизированного ассемблерного кода.

Изменения, внесенные в компилятор и динамический загрузчик, повлекли за собой ускорение выполнения тестов на встраиваемой реляционной базе данных SQLite, собранной с динамической библиотекой. Получилось ускорить время выполнения на 7.0% за счет отказа от ленивого связывания во время выполнения исполняемого файла.

Данная оптимизация полезна в программах, которые используют динамические библиотеки и для которых производительность является критически важным параметром.

Список литературы

[1]. J. Levine. Linkers and Loaders. Morgan-Kauffman, p. 256, October 1999.
[2]. J. Greenhalgh. [AArch64] Tighten direct call pattern to repair -fno-plt. https://gcc.gnu.org/ml/gcc-patches/2015-08/msg00152.html.
[3]. A. Monakov. PIC calls without PLT, generic implementation. https://gcc.gnu.org/ml/gcc-patches/2015-05/msg00225.html.
[4]. D. Melnik. Developing Interblock Combine Pass in GCC. GNU Tools Cauldron 2013. https://gcc.gnu.org/wiki/cauldron2013.
[5]. ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition, Section A2.8.6.60, 04 June 2009.
[6]. Веб-сайт SQLite. http://www.sqlite.org/about.html.

Dynamic loader optimization for ARM

E.A. Kudryashov <eugene.a.kudryashov@gmail.com>
D.M. Melnik <dm@ispras.ru>
A.V. Monakov <amonakov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The paper discusses an optimization approach for external calls in position-independent code that is based on loading the callee address immediately at the call site from the Global Offset Table (GOT), avoiding the use of the Procedure Linkage Table (PLT). Normally the Linux toolchain creates the PLT both in the main executable (which comprises position-dependent code and has to rely on the PLT mechanism to make external calls) and in shared libraries, where the PLT serves to implement lazy binding of dynamic symbols, but is not required otherwise. However, calls via the PLT have some overhead due to an extra jump instruction and poorer instruction cache locality. On some architectures, binary interface of PLT calls constrains compiler optimization at the call site. It is possible to avoid the overhead of PLT calls by loading the callee address from the GOT at the call site and performing an indirect call, although it prevents lazy symbol resolution and may cause increase in code size.

We implement this code generation variant in GCC compiler for x86 and ARM architectures. On ARM, loading the callee address from the GOT at call site normally needs a complex sequence with three load instructions. To improve that, we propose new relocation types that allow to build a PC-relative address of a given GOT slot with a pair of movt, movw instructions, and implement these relocation types in GCC and Binutils (assembler and linker) for both ARM and Thumb-2 modes. Our evaluation results show that proposed optimization yields performance improvements on both x86 (up to 12% improvement with Clang/LLVM built with multiple shared libraries, on big translation units) and ARM (up to 7% improvement with SQLite, average over several tests), even though code size on ARM also grows by 13-15%.

Keywords: program optimizations; dynamic loader; global offset table; procedure linkage table; relocations; ARM

DOI: 10.15514/ISPRAS-2016-28(1)-4

For citation: Kudryashov E.A., Melnik D.M., Monakov A.V. Dynamic loader optimization for ARM. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 1, 2016, pp. 63-80 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-4

References

[1]. J. Levine. Linkers and Loaders. Morgan-Kauffman, p. 256, October 1999.
[2]. J. Greenhalgh. [AArch64] Tighten direct call pattern to repair -fno-plt. https://gcc.gnu.org/ml/gcc-patches/2015-08/msg00152.html.
[3]. A. Monakov. PIC calls without PLT, generic implementation. https://gcc.gnu.org/ml/gcc-patches/2015-05/msg00225.html.
[4]. D. Melnik. Developing Interblock Combine Pass in GCC. GNU Tools Cauldron 2013. https://gcc.gnu.org/wiki/cauldron2013.
[5]. ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition, Section A2.8.6.60, 04 June 2009.
[6]. Веб-сайт SQLite. http://www.sqlite.org/about.html.

Перекрытие коммуникаций и вычислений в итерационных методах решения систем линейных уравнений на GPU*

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

Аннотация. Методы подпространства Крылова, такие как метод сопряжённых градиентов и стабилизированный метод бисопряжённых градиентов, давно используются для решения симметричных и несимметричных систем линейных алгебраических уравнений. Это находит широкое применение при численном решении дифференциальных уравнений, которые возникают, например, в задачах вычислительной физики. Однако при увеличении размеров расчетной сетки и, соответственно, количества вычислительных процессов значительную часть времени работы могут занимать коммуникации, во время которых расчёты простаивают. Это происходит из-за того, что в оригинальных формулировках методов результат скалярного произведения, которое требует редукции, требуется уже на следующем шаге метода, что приводит к барьерной синхронизации всех потоков. При значительном количестве итераций это может привести к деградации производительности. В статье рассматривается использование альтернативных формулировок методов подпространства Крылова, позволяющих перекрыть часть вычислений и параллельных коммуникаций, часто за счет увеличения объема вычислений. Нами предложены собственные реализации этих подходов для использования гибридного решателя с графическими ускорителями, использующими технологию CUDA, в рамках программного пакета OpenFOAM, а также описаны особенности их переноса на акселераторы. Для дальнейшей оптимизации используются асинхронные коллективные операции, предоставленные стандартом межпроцессного взаимодействия MPI-3, которые позволяют избавиться от барьерной синхронизации и снизить латентность операций обмена. Представлены результаты тестирования нашего подхода на одной из станадартных задач пакета OpenFOAM с расчётными сетками в 2 и 4 миллиона ячеек с использованием нескольких графических ускорителей.

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

* Работа поддержана грантом РФФИ 13-07-12102

DOI: 10.15514/ISPRAS-2016-28(1)-5

Для цитирования: Платонов В.А., Монаков А.В. Перекрытие коммуникаций и вычислений в итерационных методах решения систем линейных уравнений на GPU. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 81-92. DOI: 10.15514/ISPRAS-2016-28(1)-5

1. Введение

В настоящее время большинство высокопроизводительных гетерогенных систем дополняется акселераторами, чья специализированная архитектура может давать существенный прирост производительности на некоторых классах задач. Наиболее популярным типом подобных ускорителей являются графические акселераторы (GPU), архитектура которых оптимизирована для задач с высоким параллелизмом.

Решение задач вычислительной гидродинамики в конечном итоге сводится к решению нескольких систем линейных алгебраических уравнений, при этом коэффициенты системы представляют собой разреженную матрицу. Наиболее популярные численные методы для решения подобных систем – это методы подпространства Крылов, такие как метод сопряжённых градиентов или стабилизированнный метод бисопряжённых градиентов. Сам по себе перенос вычислений на графический ускоритель способен дать существенный прирост производительности [4], однако при расчёте на большом количестве вычислительных узлов заметную роль начинают играть коммуникации между процессами. Любое вычисление скалярного произведения подразумевает обмен частичными суммами между всеми процессами, что сопряжено с необходимостью синхронизации расчётов на всех процессах. Один из подходов к решению этой проблемы состоит в том, чтобы перекрывать коммуникации и вычисления, однако оригинальные формулировки методов подпространства Крылова строго последовательны и не предоставляют такой возможности. В данной работе рассматриваются альтернативные формулировки этих методов, которые позволяют перекрытие за счёт увеличения требуемой памяти и количества вычислений. Реализация выполнена в рамках открытого пакета для задач вычислительной гидродинамики OpenFOAM [1]. Кроме того, используется ранее разработанная нами библиотека, содержащая все необходимые операции (умножение матрицы на вектор, скалярное произведение, линейные комбинации) для реализации методов и использующая автоматическую настройку для повышения производительности на GPU [3,4].

2. Классические методы подпространства Крылова

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

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$

2: $\boldsymbol{p} = \boldsymbol{0}$

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

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

Классическая формулировка требует хранить 6 векторов, а за одну итерацию цикла происходит два умножения матрицы на вектор, 3 линейных (вида ax+y) комбинации и 3 скалярных произведения (вместе с расчётом невязки).

Классический вариант стабилизированного метода бисопряжённых градиентов с предобуславливанием, использующийся в OpenFOAM для решения системы $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$:

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$
2: $\boldsymbol{p} = 0$
3: $\boldsymbol{rw} = \boldsymbol{r}$
4: do
5: $\rho_i = \rho_{i-1}$
6: $\rho_i = (\boldsymbol{r}, \boldsymbol{rw})$
7: $\beta = (\alpha\,\rho_i)/(\omega\rho_{-1})$
8: $\boldsymbol{p} = \boldsymbol{r} + \beta\boldsymbol{p} - \beta\omega\boldsymbol{v}$
9: $\boldsymbol{ph} = \boldsymbol{M}^{-1}\boldsymbol{p}$
10: $\boldsymbol{v} = \boldsymbol{A}\,\boldsymbol{ph}$
11: $\alpha = \rho_i/(\boldsymbol{rw}, \boldsymbol{v})$
12: $\boldsymbol{s} = \boldsymbol{r} - \alpha\boldsymbol{v}$
13: $\boldsymbol{sh} = \boldsymbol{M}^{-1}\boldsymbol{s}$
14: $\boldsymbol{t} = \boldsymbol{A}\,\boldsymbol{sh}$
15: $\omega = (\boldsymbol{t}, \boldsymbol{s})/(\boldsymbol{t}, \boldsymbol{t})$
16: $\boldsymbol{x} = \boldsymbol{x} + \alpha*\boldsymbol{ph} + \omega*\boldsymbol{sh}$
17: $\boldsymbol{r} = \boldsymbol{s} - \omega*\boldsymbol{t}$
18: while (!stopCondition(|***r***|, ++i))

Классическая формулировка метода требует хранить 9 векторов, а за одну итерацию цикла происходит 4 умножения матрицы на вектор, 6 линейных (вида ax+y) комбинаций и 4 скалярных произведения (вместе с расчётом невязки).

3. Перекрытие MPI-коммуникаций

При параллельной работе OpenFOAM происходит декомпозиция расчётной на сетки на отдельные, в простейшем случае, непересекающиеся подобласти, каждая из которых отдаётся одному из процессов. Таким образом большую часть расчётов можно выполнять независимо. Тем не менее, некоторые параметры счёта требуют информации со всей расчётной области. Если рассматривать непосредственно задачу решения СЛАУ, то переход к многопоточности означает следующее: каждый процесс в качестве матрицы использует диагональный блок оригинальной матрицы (соответствующий подобласти) и свою часть общих векторов решения и правой части. В процессе решения каждый процесс обновляет своё локальное решение и финальным ответом становится объединение всех локальных ответов в один вектор. Соответственно, линейные комбинации над локальными векторами можно производить без взаимодействия с другими процессами. Сложнее обстоит ситуация со скалярными произведениями и умножением матрицы на вектор. В первом случае результатом операции является сумма всех локальных ответов, вызывая общую синхронизацию, что негативно влияет на производительность, так как необходимо пассивно ожидать выполнения операции и получения результатов со всех процессов. Частным случаем скалярного произведения можно считать и подсчёт невязки. В случае умножения матрицы на вектор, локальной операции также недостаточно. Как несложно заметить, в результате декомпозиции, не все элементы оригинальной матрицы попали в одну из локальных матриц. Эти элементы матрицы, называемые граничными коэффициентами, возникают за счёт взаимодействия на границе между подобластями. Таким образом, для получения ответа, процессу необходимо произвести локальное умножение и добавить результат умножения граничных условий на соответствующую часть вектора соседнего (имеющего общую границу с текущим) процесса. Важно отметить, что синхронизация происходит только с соседями, а не со всеми процессами.

Таким образом во время многопроцессорного запуска OpenFOAM, можно выделить три разных типа коммуникаций внутри решателя СЛАУ:

- Обмен граничными коэффициентами при умножении матрицы на вектор
- Редукция результатов скалярного произведения
- Подсчёт невязки

При увеличении количества параллельных процессов, коммуникации начинают играть всё большую роль в производительности системы. Во-первых, повышается стоимость обмена данными. Во-вторых, глобальные синхронизации вызывают простой всех потоков до завершения обмена данными. Эффективным способом борьбы со второй проблемой является перекрытие коммуникаций и вычислений, то есть выполнение расчётов во время асинхронной передачи данных. Например, при умножении матрицы на вектор, обмен граничными коэффициентами можно производить параллельно с локальным произведением, а редукцию невязки выполнять параллельно с выполнением следующей итерации. Сложнее обстоит ситуация со скалярными произведениями: в классических формулировках методов подпространства Крылова применить данный подход не представляется возможным, так как алгоритм строго последователен в том смысле, что результат выполнения любого шага алгоритма требуется на следующем шаге и соответственно все необходимые коммуникации должны быть выполнены к тому моменту. Для гетерогенных систем с акселераторами дополнительная сложность заключается в том, что для запуска редукции необходимо дождаться завершения соответствующего ядра на ускорителе, при этом крайне желательно запустить новые независимые ядра на время коммуникации.

В рамках нашей работы мы использовали асинхронные глобальные редукции, предоставленные стандартом MPI-3 и реализованные в актуальных версиях пакетов OpenMPI и mvapich. Нами были реализованы необходимые абстракции для использования в рамках пакета OpenFOAM.

4. Переформулированные методы подпространства Крылова

Существует несколько реализаций метода сопряженных градиентов, направленных на увеличение перекрытия вычислений и коммуникаций. В них возникает компромисс между степенью перекрытия и количеством дополнительно проводимых вычислений и хранимых векторов. Для использования на GPU нами был использован следующий вариант [5]:

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$
2: $\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{r}$
3: $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{u}$
4: $\gamma_{-1} = \infty$
5: $\gamma_0 = (\boldsymbol{r}, \boldsymbol{u})$
6: $\delta = (\boldsymbol{w}, \boldsymbol{u})$
7: do
8: $\boldsymbol{m} = \boldsymbol{M}^{-1}\boldsymbol{w}$
9: $\boldsymbol{n} = \boldsymbol{A}\boldsymbol{m}$
10: $\beta = \gamma_i / \gamma_{i-1}$

11: $\alpha = \gamma_i / (\delta - \beta\, \gamma_i / \alpha)$
12: $\boldsymbol{z} = \boldsymbol{n} + \beta \boldsymbol{z}$
13: $\boldsymbol{q} = \boldsymbol{m} + \beta \boldsymbol{q}$
14: $\boldsymbol{s} = \boldsymbol{w} + \beta \boldsymbol{s}$
15: $\boldsymbol{p} = \boldsymbol{u} + \beta \boldsymbol{p}$
16: $\boldsymbol{x} = \boldsymbol{x} + \alpha \boldsymbol{p}$
17: $\boldsymbol{r} = \boldsymbol{r} - \alpha \boldsymbol{s}$
18: $\boldsymbol{u} = \boldsymbol{u} - \alpha \boldsymbol{q}$
19: $\boldsymbol{w} = \boldsymbol{w} - \alpha \boldsymbol{z}$
20: $\gamma_i = (\boldsymbol{r}, \boldsymbol{u})$
21: $\delta = (\boldsymbol{w}, \boldsymbol{u})$
22: while (!stopCondition(|***r***|, ++i))

В переформулированном варианте алгоритма можно выделить важные особенности:

- глобальную редукции невязки и скаляров в строках 20-21 можно выполнить одной асинхронной all-reduce коммуникацией на трехкомпонентном векторе $\{\gamma_i, \delta, |\boldsymbol{r}|\}$, которая будет перекрыта с умножением матрицы и предобуславливанием в строках 8-9
- за счёт удачного расположения операций вычисления в строках 12-22 требуют запуска всего двух вычислительных ядер, что уменьшает количество накладных расходов на запуск ядер, а также выигрывает за счёт переиспользования данных
- обновление граничных коэффициентов при умножении на матрицу можно перекрыть с обновлением векторов ***q***, ***s***, ***p***, ***x***, ***r***, ***u***, если вынести обновление векторов ***z*** и ***w*** в отдельное ядро
- новая формулировка требует хранить 11 векторов, а за одну итерацию цикла происходит два умножения матрицы на вектор, 8 линейных (вида ax+y) комбинаций и 3 скалярных произведения (вместе с расчётом невязки).

Таким образом, нам удалось серьёзно улучшить эффективность коммуникаций алгоритма, поскольку за счёт их объединения, а также перекрытия с вычислениями удалось избавить от всех точек синхронизации процессов. Ценой этой эффективности стали значительно возросшие требования к памяти, а также количество линейных комбинаций, таким образом для систем с малой стоимостью коммуникаций новый вариант алгоритма может показывать производительность ниже, чем классический метод.

Для стабилизированного метода бисопряжённых градиентов также существует несколько переформулированных вариантов, повышающих эффективность коммуникаций. Первым был предложен вариант с двумя

точками синхронизации [6], затем только с одной точкой синхронизации [7] и наконец, вариант не содержащий явных точек синхронизации [8].

1: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$

2: $\boldsymbol{rw} = \boldsymbol{r}$

3: $\boldsymbol{z} = \boldsymbol{M}^{-1}\boldsymbol{r}$

4: $\boldsymbol{vh} = \boldsymbol{z}$

5: $\rho = (\boldsymbol{r}, \boldsymbol{rw})$

6: $\rho_{i-1} = \rho_i$

7: do

8: $\boldsymbol{v} = \boldsymbol{A}\,\boldsymbol{vh}$

9: $\gamma = (\boldsymbol{v}, \boldsymbol{rw})$

10: $\boldsymbol{s} = \boldsymbol{M}^{-1}\boldsymbol{v}$

11: $\alpha = \rho_i / \gamma$

12: $\boldsymbol{th} = \boldsymbol{z} - \alpha\boldsymbol{s}$

13: $\boldsymbol{t} = \boldsymbol{A}\,\boldsymbol{th}$

14: $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{vh}$

15: $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{v}$

16: $\theta = (\boldsymbol{t}, \boldsymbol{r})$

17: $\varphi = (\boldsymbol{t}, \boldsymbol{t})$

18: $\psi = (\boldsymbol{t}, \boldsymbol{rw})$

19: $\boldsymbol{z} = \boldsymbol{M}^{-1}\boldsymbol{t}$

20: $\omega = \theta/\varphi$

21: $\boldsymbol{x} = \boldsymbol{x} + \omega\boldsymbol{th}$

22: $\boldsymbol{r} = \boldsymbol{r} - \omega\boldsymbol{t}$

23: $\rho_{i-1} = \rho_i$

24: $\rho_i = -\omega\psi$

25: $\beta = (\alpha\rho_i) / (\rho_{i-1}\omega)$

26: $\boldsymbol{z} = \boldsymbol{th} - \omega\boldsymbol{z}$

27: $\boldsymbol{vh} = \boldsymbol{z} + \beta(\boldsymbol{vh} - \omega\boldsymbol{s})$

28: while (!stopCondition(|$\boldsymbol{r}$|, ++i))

В переформулированном варианте алгоритма можно выделить важные особенности:

- глобальную редукцию скаляра в строке 9 и невязки в строке 22 можно перекрыть с последующим предобуславливанием в строке 10
- глобальную редукцию в строках 16-18 можно выполнить одной коммуникацией на трехкомпонентном векторе $\{\theta, \varphi, \psi\}$, которая будет перекрыта с последующим предобуславливанием в строке 19
- операции 14-18 и 20-27 возможно объединить в два ядра для уменьшения накладных расходов и уменьшения доступов в память
- новая формулировка требует хранить 9 векторов, а за одну итерацию цикла происходит четыре умножения матрицы на вектор, 8 линейных (вида ax+y) комбинаций и 5 скалярных произведений (вместе с расчётом невязки)

Аналогично варианту с методом сопряжённых градиентов, переформулированный метод значительно повышает качество коммуникаций (все коммуникации перекрываются вычислениями) ценой увеличения количества операций.

5. Результаты тестирования

Описанные оптимизации были реализованы и протестированы в модуле решения линейных систем для проекта FOAM-Extend. Для тестирования производительности использовались гетерогенные узлы с процессорами Xeon E5 2620 (6 ядер, 2.0 GHz) и акселераторами GeForce GTX TITAN. В качестве эталонной процессорной реализации использовались методы PCG и BiCGStab с предобуславливателем DIC. При запуске на центральном процессоре использовались все его физические ядра, по одному потоку на ядро.

На тесте cavity (сетка с 2 млн. ячеек) с приложением icoFoam были получены следующие результаты:

Количество процессоров/ ускорителей	PCG	BiCGStab	GPUPCG	GPUBCGS
1	235	301	125	143
2	113	145	63	75
Ускорение	2.07	2.07	1.98	1.90

На тесте cavity (сетка с 4 млн. ячеек) с приложением icoFoam были получены следующие результаты:

Количество процессоров/ ускорителей	PCG	BiCGStab	GPUPCG	GPUBCGS
1	602	791	290	348

2	303	400	155	187
Ускорение	1.98	1.97	1.87	1.86

Из полученных результатов можно заметить, что текущая реализация позволяет достичь ускорения около 2 раз на одном ускорителе относительно 6-ти ядерного процессора. При этом увеличение размера сетки положительно сказывается на приросте производительности. Проведённые оптимизации коммуникаций позволяют сохранять хорошую масштабируемость при увеличении количества ячеек.

7. Заключение

В данной работе представлено несколько оптимизаций для гибридной реализации метода сопряженных градиентов и стабилизированного метода бисопряжённых градиентов. С предложенными оптимизациями на GPU-акселераторах достигается значительно большая производительность по сравнению с многоядерными процессорами, но только для задач достаточно большого размера. Проведённые оптимизации коммункаций позволяют повысить масштабирования решения. В будущем планируется разработать гибридный вариант для алгебраического многосеточного метода и добавить возможность использовать его в качестве предобуславливателя уже существующих решателей.

Список литературы

[1]. Страница OpenFOAM — http://openfoam.org/
[2]. Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM. 2003.
[3]. А. Монаков, Е. Велесевич, В. Платонов, А. Аветисян. Инструменты анализа и разработки эффективного кода для параллельных архитектур. Труды Института системного программирования РАН, том 26 (выпуск 1), 2014 г.. стр. 357-374. DOI: 10.15514/ISPRAS-2014-26(1)-14
[4]. А.В. Монаков, В.А. Платонов, Оптимизация метода решения линейных систем уравнений в OpenFOAM для платформы MPI + CUDA. Труды Института системного программирования РАН, том 26 (выпуск 3), 2014 г., стр. 91-102. DOI: 10.15514/ISPRAS-2014-26(3)-4
[5]. P. Ghysels, W. Vanroose. Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm. Submitted to Parallel Computing, 2012.
[6]. Thierry Jacques, Laurent Nicolas, Christian Vollaire, 7th International Conference, HPCN Europe 1999 Amsterdam, The Netherlands, April 12–14, 1999 Proceedings, pp 1025-1031
[7]. L. Yang, R. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02).

Proceedings., IEEE ComputerSociety, Los Alamitos, CA, USA, 2002, pp. 324–328. doi:10.1109/ICAPP.2002.1173595

[8]. Boris Krasnopolsky, The reordered BiCGStab method for distributed memory computer systems, Procedia Computer Science, Volume 1, Issue 1, May 2010, Pages 213-218, ISSN 1877-0509, http://dx.doi.org/10.1016/j.procs.2010.04.024.

Overlapping communications and computations in GPU-based iterative linear solvers [*]

V.A. Platonov <soh@ispras.ru>
A.V. Monakov <amonakov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. Krylov subspace methods such as Conjugate Gradient and Biconjugate Gradient Stabilized methods are well known approaches for solving symmetric and asymmetric systems of linear algebraic equations, such as systems usually arising from partial differential equations in computational mathematics problems, like Navier-Stokes equations in fluid dynamics. With increasing sizes of meshes and numbers of computational nodes, network communications time may become an issue: stalls during global reductions have increasing duration, preventing useful computations. This happens because, in original formulations of methods, computing a dot product requires a global reduce operation, and its value is required on the next step, so each process has to stop until all others reach this point, like in a barrier synchronization. We research alternative formulations of conjugate gradient methods (Preconditioned Conjugate Gradient and BiCGStab) for GPU-based iterative linear system solvers. They allow to have an overlap of parallel computations and communications, at the cost of increased amount of computations and memory requirements. We describe an implementation of our approach for GPU-accelerated hybrid systems in OpenFOAM, an open source framework for computational fluid dynamics. Asynchronous collective communications from MPI-3 parallel programming API are used to avoid full barrier synchronization and reduce latency. Experimental results on 2 and 4 million cases from standard OpenFOAM problems are presented.

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

DOI: 10.15514/ISPRAS-2016-28(1)-5

For citation: Platonov V.A., Monakov A.V. Overlapping communications and computations in GPU-based iterative linear solvers**.** *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 1, 2016, pp. 81-92 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-5

References

[1]. OpenFOAM — http://openfoam.org/
[2]. Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM. 2003.

[*] The paper is supported by RFBR grant 13-07-12102

[3]. A. Monakov, E. Velesevich, V. Platonov, A. Avetisyan. Instrumenty analiza i razrabotki jeffektivnogo koda dlja parallel'nyh arhitektur (Analysis and development tools for efficient programs on parallel architectures). Trudy ISP RAN [Proceedings of ISP RAS], volume 26 (issue 1), 2014, pp. 357-374 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-14

[4]. A. Monakov, V. Platonov. Optimizacija metoda reshenija linejnyh sistem uravnenij v OpenFOAM dlja platformy MPI + CUDA (Optimizations for linear solvers in OpenFOAM for MPI + CUDA platform). Trudy ISP RAN [Proceedings of ISP RAS], volume 26 (issue 3), 2014, pp. 91-102 (in Russian). DOI: 10.15514/ISPRAS-2014-26(3)-4

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

[6]. Thierry Jacques, Laurent Nicolas, Christian Vollaire, 7th International Conference, HPCN Europe 1999 Amsterdam, The Netherlands, April 12–14, 1999 Proceedings, pp 1025-1031

[1] L. Yang, R. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02). Proceedings., IEEE ComputerSociety, Los Alamitos, CA, USA, 2002, pp. 324–328. doi:10.1109/ICAPP.2002.1173595

[7]. Boris Krasnopolsky, The reordered BiCGStab method for distributed memory computer systems, Procedia Computer Science, Volume 1, Issue 1, May 2010, Pages 213-218, ISSN 1877-0509, http://dx.doi.org/10.1016/j.procs.2010.04.024.

Равномерное распределение нагрузки аппаратно-программного ядра в UNIX-системах

Е.В. Пальчевский <teelxp@inbox.ru>
А.Р. Халиков <khalikov.albert.r@gmail.com>
ФГБОУ ВО Уфимский государственный авиационный технический университет,
450000, Россия, г. Уфа, ул. К. Маркса, 12

Аннотация. В данной статье рассматривается задача максимального увеличения пропускной способности сетевого стека при взаимодействии с аппаратно-программным ядром для обеспечения стабильности работы физического сервера. Разработаны алгоритмы и программные коды для оптимизации распределения нагрузки на центральные процессоры методом параллелизации ядра. Также рассмотрены статистические данные повышающейся мощности распределенных атак, влияющих на сетевую инфраструктуру. Показано влияние приложений, имеющих выход во внешнюю глобальную сеть, на производственную часть физического сервера, представляемой в виде физических ресурсов. С помощью разработанного и реализованного алгоритма (на языке «*BASH*»), произведено распределение нагрузочной способности по физическим ядрам сервера с целью дальнейшего снижения нагрузочной способности на вычислительную мощность центрального процессора. Продемонстрированы блок-схемы алгоритмов, а также итоговые результаты тестирования каждого этапа разработки. Реализована оптимизация сетевого режима «*AF_PACKET*», благодаря которой появилась возможность принимать внешние сетевые пакеты без каких-либо блокировок, что, в свою очередь, повышает эффективность достижения заданной цели (при запросе от сервера к клиенту). Анализируется возможность принимать до десяти миллионов входящих сетевых пакетов с помощью программных средств физического сервера, что позволяет обеспечить стабильную обработку информации для бесперебойной работы при *DDoS*-атаках «*SYN*-флуда», у которых реализована возможность перегрузки многомиллионными сетевыми пакетами. Подобное количество входящих сетевых пакетов может привести к заполнению внешнего сетевого канала с последующим повышением нагрузочной способности сетевого TCP/IP стека, что перекрывает возможность зоны удаленного управления физическим сервером в кратчайшие сроки, а также нарушает работоспособность рабочей среды.

Ключевые слова: нагрузочная способность; *CPU*; параллелизация; параллелизация процессорной нагрузки; оптимизация; защита от *DDoS*-атак; фильтрация трафика; фильтрация вредоносного трафика.

DOI: 10.15514/ISPRAS-2016-28(1)-6

Для цитирования: Пальчевский Е.В., Халиков А.Р. Равномерное распределение нагрузки аппаратно-программного ядра в UNIX-системах. Труды ИСП РАН, 2016, том 28, вып. 1, с. 93-102.DOI: 10.15514/ISPRAS-2016-28(1)-6

1. Введение

При массированных *DDoS*-атаках в программном ядре происходят многочисленные сбои, ошибки и перегрузки [1, 2]. Это приводит к замедленной работе всего компьютера и происходит вызываету величение нагрузки на центральные процессоры [3]. Данные по мощности внешних сетевых угроз, за счет которых можно произвести сетевую перегрузку, представлены на рис. 1.

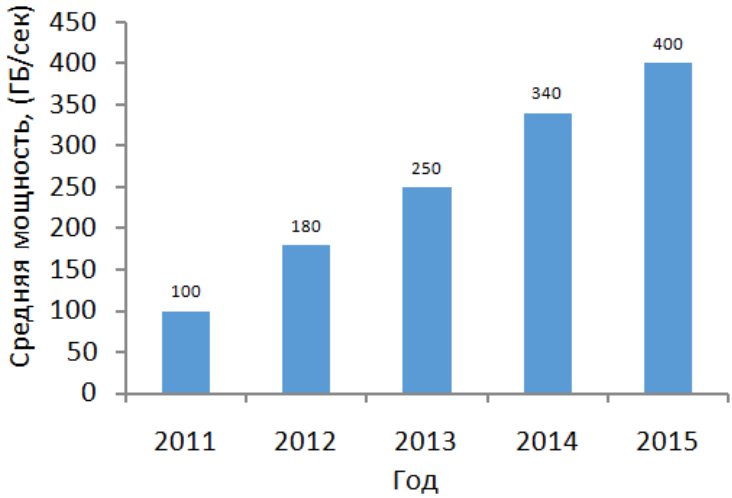


Рис.1. График DDoS-атак в годовом эквиваленте за 2011-2015 [4]
Fig. 1. Schedule of DDoS-attacks in annual terms for the 2011-2015 [4]

Параллелизация программного ядра – это равномерное распределение нагрузочной способности центральных процессоров по физическим ядрам сервера. Подобная операция дает возможность повысить производительность. Появляется возможность повышения устойчивости против разносторонней отправки вредоносного сетевого трафика, направленного с разных ЭВМ [4, 5].

2. Перераспределение нагрузки по физическим ядрам

Ядро – это централизованная часть операционной системы, которая обеспечивает различным программам и приложениям доступ к физическим ресурсам ЭВМ, а также обеспечивает взаимосвязь с сетевым стеком [6].

Для перераспределения нагрузки по ядрам необходимо включить технологию «*PROMISC*», за счет этого в сетевом стеке срабатывает ускорительный режим

для одного ядра. Становится возможным многократное увеличение входящих сетевых пакетов. Запуск производится командой «*ifconfig eth*6 *promisc*». При *DDoS*-атаке после выполненной процедуры возникает максимально возможная загруженность первого ядра. Это позволяет распределять нагрузки *TCP*/*IP*-стека по всем ядрам сервера.

Для того, чтобы организовать просмотр скорости (кбит/с) внешнего сетевого интерфейса нужно применить разработанный скрипт, который написан на языке программирования «*BASH*» [7]:

Блок-схема «.*sh*-скрипта» показана на рис. 2.

```
#!/bin/bash
 INTERVAL="1"  # update interval in seconds
 if [ -z "$1" ]; then
      echo
      echo usage: $0 [network-interface]
      echo
      echo e.g. $0 eth0
      echo
      echo shows packets-per-second
      exit
fi
 IF=$1
 while true
do
      R1=`cat /sys/class/net/$1/statistics/rx_packets`
      T1=`cat /sys/class/net/$1/statistics/tx_packets`
      sleep $INTERVAL
      R2=`cat /sys/class/net/$1/statistics/rx_packets`
      T2=`cat /sys/class/net/$1/statistics/tx_packets`
      TXPPS=`expr $T2 - $T1`
      RXPPS=`expr $R2 - $R1`
      echo "TX $1: $TXPPS pkts/s RX $1: $RXPPS pkts/s"
done
```

Пример использования вышеприведенного кода в скриптовом файле: «*bash /root/speed.sh eth0*». Файл «*speed.sh*» создается в системной папке «*root*» с помощью команды «*touch speed.sh*» со вставкой программных строк, которые приведены выше. В ответ, при *DDoS*-атаке, будет выведено сообщение о том, что скорость на внешнем интерфейсе около четырех миллионов пакетов в секунду. Количественная размерность сетевых пакетов зависит от настроек физического сервера.

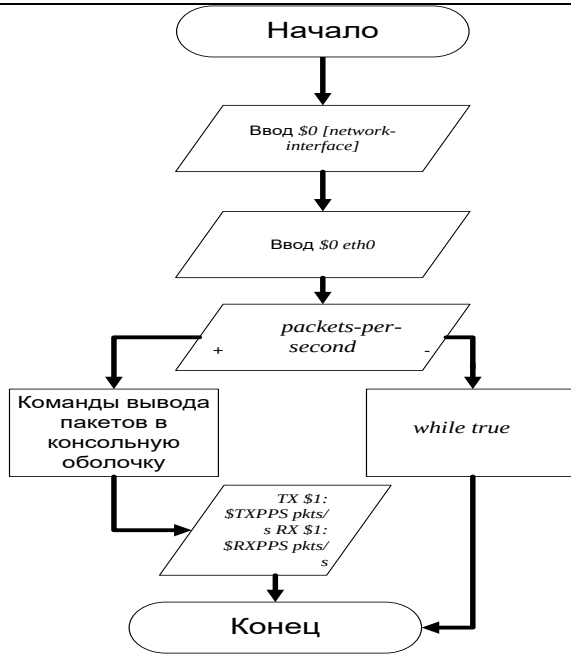


Рис. 2. Блок-схема вывода пакетов и увеличения пропускной способности.

Fig. 2. Block diagram of the output packets, and increase throughput.

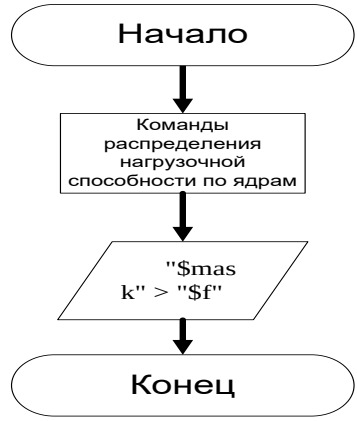


Рис. 3. Блок-схема распределения нагрузки по нескольким ядрам.

Fig. 3. A block diagram of load-balancing across multiple cores.

Для того, чтобы уменьшить нагрузку на первом ядре, был написан специальный скрипт на языке программирования «*BASH*», который позволяет реализовать возможность равномерного распределения нагрузки по всем физическим ядрам:

Блок-схема «*.sh*-кода» показана на рис. 3.

```
#!/bin/bash
ncpus=`grep -ciw ^processor /proc/cpuinfo`
test "$ncpus" -gt 1 || exit 1
n=0
for irq in `cat /proc/interrupts | grep eth | awk '{print $1}' | sed s/\://g`
do
   f="/proc/irq/$irq/smp_affinity"
   test -r "$f" || continue
   cpu=$[$ncpus - ($n % $ncpus) - 1]
   if [ $cpu -ge 0 ]
        then
           mask=`printf %x $[2 ** $cpu]`
           echo "Assign SMP affinity: eth queue $n, irq $irq, cpu $cpu, mask
0x$mask"
           echo "$mask" > "$f"
           let n+=1
   fi
done
```

После запуска вышеприведенного кода при атаке в $6x10^6$ входящих пакетов в секунду производительность *CPU* повысилась в несколько раз (за счет распределения нагрузочной способности на центральные процессоры), что представлено в таблице 1.

Табл. 1.- Нагрузка на ядра при применении оптимизации распределенной нагрузки на физический сервер

Table 2. The load on the core when applying optimization of distributed load on a physical server

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Нагрузка, %	55,1	52,5	62,5	62,5	57,7	47,7	55,9	61,4	55,1	52,5	62,5	62,5
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Нагрузка, %	57,7	47,7	55,9	61,4	55,1	52,5	62,5	62,5	57,7	47,7	55,9	61,4

С помощью двух скриптов «*BASH*» мы достигли повышенной производительности при принятии сетевых пакетов *TCP*/*IP* стеком.

3. Разработка и использование алгоритма для равномерной нагрузки на ядра

Для оптимизации аппаратно-программного ядра написан код на языке программирования *C++*, алгоритм которого предназначен для сдерживания атаки до $10x10^6$ пакетов в секунду, посредством равномерно распределенной нагрузки на физические серверы.

Многократное тестирование показало, что разработанный алгоритм не вызывает ошибок в ядре, фрагмент кода, отвечающий за переключения свитча в режим «*PROMISC*», приведен ниже. Режим «*PROMISC*» позволяет принимать сетевой плате все входящие пакеты, направленные различным адресатам, которые находятся на физическом сервере.

```
 struct packet_mreq sock_params;
memset(&sock_params, 0, sizeof(sock_params));
sock_params.mr_type = PACKET_MR_PROMISC;
sock_params.mr_ifindex = interface_number;
   int set_promisc = setsockopt(packet_socket, SOL_PACKET, PACKET_ADD_MEMBERSHIP, (void *)&sock_params, sizeof(sock_params));
if (set_promisc == -1) {
   printf("Can't enable promisc mode\n");
   return -1;    }
struct sockaddr_ll bind_address;
memset(&bind_address, 0, sizeof(bind_address));
bind_address.sll_family = AF_PACKET;
bind_address.sll_protocol = htons(ETH_P_ALL);
bind_address.sll_ifindex = interface_number;
struct tpacket_req3 req;
memset(&req, 0, sizeof(req));
req.tp_block_size = blocksiz;
req.tp_frame_size = framesiz;
req.tp_block_nr = blocknum;
req.tp_frame_nr = (blocksiz * blocknum) / framesiz;
req.tp_retire_blk_tov = 60; // Timeout in msec
req.tp_feature_req_word = TP_FT_REQ_FILL_RXHASH;
int setsockopt_rx_ring = setsockopt(packet_socket, SOL_PACKET , PACKET_RX_RING , (void*)&req , sizeof(req));
   if (setsockopt_rx_ring == -1) {
   printf("Can't enable RX_RING for AF_PACKET socket\n");
   return -1;    }
```

Операционная система на ядре *LINUX* версии «4.5.2» в вышеприведенном алгоритме увеличила показатели производительности фактически в 2.5 раза (см. таблицу 2). Тестирование проводилось на следующей конфигурации [8]:

- *CPU* 2 x *Intel Xeon* 5660 (в сумме 12 ядер / 24 потока по 2.8*GHz*, 12*Mb Cache*, 6.40 *GT*/*s*)
- *RAM* 48*Gb DDR*3-10600 *ECC REG*;
- *Intel S*3710 *SSDSC*2*BA*012*T*401.

Табл. 2. Оптимизированный вариант нагрузки на ядра сервера
Table. 2 - The optimized version of the load on the Server Core

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Нагрузка, %	25,1	22,5	32,5	32,5	27,7	17,7	25,9	31,4	25,1	22,5	32,5	32,5
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Нагрузка, %	27,7	17,7	31,4	25,9	31,4	25,1	22,5	32,5	32,5	27,7	17,7	31,4

4. Оптимизация «AF_PACKET» для решения проблемы с блокировкой сетевых пакетов

«*AF_PACKET*» – это режим, обладающий большим быстродействием и требующий наличие двух внешних сетевых интерфейсов. Он разрешает ядрам *UNIX*-подобных систем принимать и отправлять различные сокеты, что позволяет контактировать с внешней глобальной сетью. Главным компонентом работы является настройка системы в качестве шлюза. Такой алгоритм работы позволяет организовать полную блокировку входящих и исходящих пакетов вредоносного трафика при *DDoS*-атаке.

Чтобы избежать ложных блокировок и оптимизировать «*AF_PACKET*», рекомендуется подключить функцию «*FANOUT*», что поможет разделить входящий поток данных. Оптимальные значения рассчитаны и приведены в таблице 3.

В каждом ядре нужен как минимум один разветвитель входящих пакетов, чтобы распределить нагрузочную способность физического и логического потоков в равномерном порядке. Необходимость подключения двух разделителей в первом, двенадцатом и двадцать четвертом ядрах объясняется тем, что в операционных системах вида «*UNIX*» используется упорядоченная система нагрузки: при снижении производительности в первом ядре, загруженность которого будет приближаться к 80%, процесс в автоматическом режиме переносится на следующее ядро. На ядре под номером «двенадцать» заканчиваются физические ядра сервера по технологии «*HT*»: необходимое количество подключений «FANOUT» приравнивается к двум, с целью минимизировать нагрузочную способность на логические потоки, начинающиеся с номера «тринадцать».

Табл. 3. Расчет значений опции «FANOUT» для физических и логических ядер сервера
Table. 3. Calculation «FANOUT» option values for physical and logical server cores

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Число включений, ед.	2	1	1	1	1	1	1	1	1	1	1	2
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Число включений, ед.	1	1	1	1	1	1	1	1	1	1	1	2

Включение «*FANOUT*» производится добавлением истинного значения «*true*», к переменной «*use_multiple_fanout_processes*». Пример кода включения вышеописанного параметра: «*bool use_multiple_fanout_processes = true*».

В конечном итоге, оптимизировав «*AF_PACKET*», нагрузочная способность снизилась на несколько процентов и повлекло за собой повышенную продуктивную способность физического сервера справляться с сетевыми перегрузками, что показано в таблице 4.

Табл. 4. Нагрузочная способность логических и физических потоки сервера, при оптимальном режиме «AF_PACKET»
Table. 4. Load capacity logic and physical server threads with optimal «AF_PACKET

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Нагрузка, %	20,1	18,5	27,5	19,5	24,7	13,7	15,9	26,4	22,1	20,5	29,5	30,5
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Нагрузка, %	24,7	12,7	27,4	23,9	19,4	22,1	20,5	22,5	22,5	21,2	14,6	28,3

Наши результаты позволяют сделать вывод о эффективности параллелизации аппаратно-программного ядра в *UNIX*-системах посредством использования специализированных режимов, кодов и алгоритмов. Т.о. равномерная нагрузка на физических ядрах позволяет увеличить производительность физического сервера и повышает пропускную сетевую способность в несколько раз.

5. Заключение

В данной статье была рассмотрена возможность увеличения нагрузочной способности физического сервера для обеспечения отказоустойчивости к *DDoS*-атакам типа «*SYN*», в которых имеется многомиллионное сочетание сетевых пакетов. Это делает возможным в кратчайшие сроки заполнить канал и сетевой стек и вывести физический сервер из зоны удаленного обслуживания. Были приведены как примеры кодов на языках «*C++*» и «*BASH*», которые помогли организовать мульти-оптимизацию. Представлены

результаты тестирования разработанных алгоритмов. На начальной стадии получена возможность обработки сетевых пакетов, скорость поступления которых составляет порядка 4 миллионов в секунду. После применения двух специальных скриптов нагрузка на физические ядра снизилась и, за счет этого, стало возможным принимать сетевые пакеты до 6 миллионов в секунду. Разработанный алгоритм на языке высокого уровня «*C++*» позволил более эффективно снизить нагрузку на физические ядра сервера и обрабатывать сетевые пакеты до $10x10^6$ в секунду. В конечном результате представленный алгоритм может быть применим для сдерживания *DDoS*-атаки различных типов и видов.

Список литературы

[1]. Crist E.F., Keijser J.J. Mastering OpenVPN / E.F. Crist., Keijser J.J. – Изд-во: «Packt Publishing», – 2015. – 364 с.

[2]. Ватаманюк, А.К. Создание, обслуживание и администрирование сетей на 100% – Изд-во: «Питер», – 2010. – 350 с.

[3]. Dubrova E. Fault-Tolerant Design / E. Dubrova – Изд-во: «Springer», 2013. – 185 с.

[4]. Palchevsky E., Khalikov A. TCP/IP network STACK optimization under high load on UNIX-like systems – DSPTech'2015. Proceedings v.1. USATU, UFA, 2015. – С. 130-135.

[5]. Раго С., Стивенс У. UNIX. Профессиональное программирование, 3-е издание – С. Раго, У. Стивенс – Изд-во: «Санкт-Петербург», 2013. – 1104 с.

[6]. Krylov V., Kravtsov K. DDoS attack and interception resistance IP fast hopping based protocol – 23rd international conference on software engineering and data engineering, sede 2014. New Orleans, LA, 2014. – С. 43-48.

[7]. Блум Р., Бреснахэн К. Командная строка Linux и сценарии оболочки / Р. Блум, К. Бреснахэн – Изд-во: «Вильямс», 2013. – 784 с.

[8]. Е.В. Пальчевский, А.Р. Халиков. Техника инструментирования кода и оптимизация кодовых строк при моделировании фазовых переходов на языке C++. Труды ИСП РАН, том 27 (выпуск 6), 2015 г., стр. 87-96. DOI: 10.15514/ISPRAS-2016- 27(6)-5.

Uniformly distributed load of hardware and software core in the UNIX-based systems

E.V Palchevsky <teelxp@inbox.ru>
A.R Khalikov <khalikov.albert.r@gmail.com>
USATU, 450000, Russia, Ufa, st. Marx, 12

Abstract. In this article we consider the problem of maximizing the capacity of the network stack to the interaction of hardware and software core to ensure the stability of the physical server. The algorithms and program codes are proposed to optimize the load capacity of the

CPU by core parallelization. The paper also considers statistics of improved power of distributed attacks affecting the network infrastructure. It proved the impact of any application with access to the external global network to the production of the physical server presented in the form of physical resources. With the help of the developed and implemented the algorithm (in the language of «BASH»), produced by the distribution of the load capacity of the physical server cores, to further reduce the load capacity on the processing power of the CPU is provided. Showcased flowcharts, as well as the final test results of each stage of development are discussed. Implemented network optimization mode «AF_PACKET», which has given the opportunity to accept external network packets without any locks that, in turn, increases the efficiency of achievement of the set goals (upon request from the server to the client). The possibility of taking up to ten million incoming network packets by software physical server, which allows for stable processing of information for the smooth operation under DDoS-attacks «SYN-flood" who realized the possibility of overload multimillion network packets. A similar number of incoming network packets provides an opportunity to fill the external network channel, with a consequent increase in the load capacity of the network TCP / IP stack that covers the remote control area physical server as soon as possible. Also adversely affect the performance of the working environment.

Keywords: carrying capacity; CPU; parallelization; parallelization of CPU load; optimization; protection against DDoS-attacks; traffic filtering; filtering malicious traffic.

DOI: 10.15514/ISPRAS-2016-28(1)-6

For citation Palchevsky E.V. Khalikov A.R. Uniform load distribution of hardware and software core in the UNIX-based systems. Trudy ISP RAN /Proc. ISP RAS, 2016, vol. 28, issue 1, 2016. pp. 93-102 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-6

References

[1]. Crist E.F., Keijser J.J. Mastering OpenVPN / E.F. Crist., Keijser J.J. - Publishing house: «Packt Publishing», - 2015. - 364 p.
[2]. Vatamanyuk, A.K. Establishment, maintenance and administration of networks 100% / A.K. Vatamanyuk - Publishing house "Peter" - 2010. - 350 p.
[3]. Dubrova E. Fault-Tolerant Design / E. Dubrova - Publishing house: «Springer», 2013. - 185 p.
[4]. Palchevsky E., Khalikov A. TCP/IP network STACK optimization under high load on UNIX-like systems // DSPTech'2015. Proceedings v.1. USATU, UFA, 2015. – C. 130-135.
[5]. Rago C., Stevens W. UNIX. Vocational Programming, 3rd Edition - / C. Rago, W. Stevens - Publishing House: "St. Petersburg", 2013 - 1104.
[6]. Krylov V., Kravtsov K. DDoS attack and interception resistance IP fast hopping based protocol // 23rd international conference on software engineering and data engineering, sede 2014. New Orleans, LA, 2014. - S. 43-48.
[7]. Bloom R., Bresnahen K. Linux command line and shell / R. Bloom scenarios K. Bresnahen - Publishing House "Williams", 2013. - 784 p.
[8]. E.V. Palchevsky, A.R. Khalikov. Tehnika instrumentirovanija koda i optimizacija kodovyh strok pri modelirovanii fazovyh perehodov na jazyke C++ [Code instrumentation technique and optimization of lines of code in the simulation of phase transitions in the language C ++]. Trudy ISP RAN [Proceedings of ISP RAS], volume 27 (issue 6), 2015, pp. 87-96 (in Russian). DOI: 10.15514/ISPRAS-2015- 27(6)-5.

Тестирование системы автоматов с буферизацией сообщений

И.Б. Бурдонов <igor@ispras.ru>
А.С. Косачев <kos@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Статья посвящена проблеме тестирования составных систем, компоненты которых моделируются конечными автоматами, а взаимодействие между ними – обменом сообщениями по симплексным каналам связи. Система описывается ориентированным графом связей, вершины которого соответствуют автоматам компонентов, а дуги – каналам связи. Предполагается выполненной следующая гипотеза о связях: граф связей статический, а отображаемая им структура связей не содержит ошибок. Автомат, находящийся в вершине графа, в каждом состоянии может принимать несколько сообщений по входным дугам (не более одного по каждой дуге) и посылать несколько сообщений по выходным дугам (не более одного по каждой дуге). Целью тестирования является покрытие переходов автоматов компонентов, которые достижимы при работе этих автоматов в системе. Предполагается, что при тестировании возможно наблюдение изменения состояний автоматов в вершинах графа и сообщений на дугах графа. Сначала рассматривается упрощённая модель системы, в которой циркулирует только одно сообщение. На её примере показывается, что гипотеза о связях позволяет существенно сократить время тестирования. Полное тестирование системы автоматов без учёта гипотезы о связях может потребовать число тестовых воздействий порядка произведения чисел состояний автоматов компонентов, а с учётом гипотезы о связях – порядка суммы этих чисел. При равном числе состояний всех автоматов это даёт экспоненциальное уменьшение числа тестовых воздействий. Затем рассматривается более общая модель, когда в системе может быть одновременно много сообщений, но не более одного на каждой дуге. Определяется композиция автоматов системы и показывается, при каких ограничениях на автоматы их композиция детерминирована. Для детерминированной композиции предлагается алгоритм генерации тестов, основанный на фильтрации тестов, генерируемых для покрытия всех переходов композиции. Тест отбрасывается, если он покрывает только такие переходы в компонентах системы, которые покрываются остающимися тестами. В заключение определяются направления дальнейших исследований.

Ключевые слова: ориентированные графы, покрытие графа, взаимодействующие автоматы, распределенные системы, тестирование, сети.

DOI: 10.15514/ISPRAS-2016-28(1)-7

Для цитирования: Бурдонов И.Б., Косачев А.С. Тестирование системы автоматов с буферизацией сообщений. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 103-130. DOI: 10.15514/ISPRAS-2016-28(1)-7

1. Введение

Большинство сложных, особенно распределённых, систем представляет собой набор взаимодействующих компонентов. В данной статье компоненты моделируются конечными автоматами, а взаимодействие – обменом сообщениями между автоматами. Мы будем предполагать, что структура связей между компонентами моделируется ориентированным графом (будем называть его *графом связей*), в вершинах которого находятся автоматы, а дуги соответствуют симплексным каналам передачи сообщений. Дугу можно понимать как очередь длины 1, буферизующую сообщение, передаваемое из автомата в начале дуги в автомат в конце дуги. Кроме *внутренних* дуг, то есть дуг, соединяющих автоматы между собой, существуют также *внешние* дуги, связывающие систему с её *окружением*. Если такая дуга ведёт из окружения в некоторый автомат, то будем называть её *внешней входной* дугой, а если дуга ведёт из автомата в окружение, то – *внешней выходной* дугой. При тестировании тест подменяет собой окружение: он передаёт сообщения в систему по внешним входным дугам и принимает от системы сообщения по внешним выходным дугам.

Мы будем считать, что граф связей статический, то есть не меняющийся в процессе работы системы. В этом случае система (также как её компоненты) может моделироваться конечным автоматом, получающимся из автоматов-компонентов с помощью подходящего оператора композиции, учитывающего граф связей. Частью состояния системы является набор состояний входящих в неё автоматов, поэтому число состояний системы не меньше произведения $n_1 n_2 ... n_k$, где n_i – число состояний i-го автомата, а k – число автоматов. Даже если учитывать только *достижимые состояния* системы, то есть те, которые достижимы из её начального состояния, то их число может иметь тот же порядок, что и произведение $n_1 n_2 ... n_k$.

Система работает правильно, если структура её связей правильна, и каждый автомат-компонент работает правильно. Обратное, вообще говоря, не верно, если требования к системе не однозначно определяют её структуру, например, функциональные требования к системе, связывающие получаемую от системы реакцию с последовательностью подаваемых в систему воздействий. В данной статье целью тестирования является покрытие переходов автоматов компонентов, которые достижимы при работе этих автоматов в системе. Поэтому, если в структуре связей нет ошибок (будем называть это *гипотезой о связях*), то есть граф связей автоматов совпадает с заданным, то такое тестирование системы сводится к проверке правильности переходов каждого автомата. Проблема в том, что каждый автомат может тестироваться только как часть системы. Это означает, что тест не имеет непосредственного доступа

к автомату-компоненту, и вынужден осуществлять тестовые воздействия с помощью сообщений, посылаемых по внешним входным дугам, которые ведут, быть может, в другие автоматы. Тестирование компонента такой системы похоже на тестирование в контексте ([1], [2], [3], [4]), когда этот компонент рассматривается как тестируемая система, а остальные – как контекст. Существенное отличие, однако, в том, что в таком контексте тоже могут быть ошибки, хотя, если верна гипотеза о связях, то только в компонентах, а не в структуре связей между ними. С другой стороны, такое тестирование может проверять работу сразу нескольких компонентов, через которые проходят сообщения.

Поскольку компонент тестируется как часть системы, не все переходы автомата компонента, которые можно проверить при автономном тестировании с прямым доступом к компоненту, можно проверить при тестировании системы. Поэтому речь должна идти только о *достижимых переходах* автоматов, то есть таких переходах, которые могут быть выполнены при работе автомата как части системы.

Тестирование автомата системы (получающегося композицией автоматов компонентов для заданного графа связей) обеспечивает проход по всем достижимым переходам автомата системы. При этом, конечно, проверяются все достижимые переходы автоматов компонентов, но, вообще говоря, делается много «лишней работы». Гипотеза о связях позволяет получить существенный выигрыш во времени тестирования. В следующем разделе статьи приведён пример, для которого соотношение времени тестирования системы без учёта и с учётом правильности графа связей такое же, как соотношение произведения $n_1n_2...n_k$ (число состояний автомата системы) и суммы чисел состояний автоматов-компонентов $n_1+n_2+...+n_k$. В частности, если $n_1=n_2=...=n_k=n$, имеем соотношение n^k и nk, то есть для фиксированного числа k компонентов получаем экспоненциальное уменьшение времени тестирования.

В данной статье предлагается алгоритм построения набора тестов, который является полным (проверяет все переходы автоматов компонентов, достижимые при работе этих компонентов в системе) при выполнении двух условий: 1) верна гипотеза о связях, 2) система детерминирована. Дополнительно этот алгоритм определяет недостижимые переходы автоматов компонентов. Предполагается, во-первых, что нам известно, каким должен быть автомат каждого компонента (задан граф переходов автомата с точностью до изоморфизма) и именно это должно проверяться при тестировании. Во-вторых, тест может наблюдать как состояния автоматов системы, так и сообщения, передаваемые по дугам графа связей. Поскольку мы не налагаем никаких ограничений на связность графов переходов автоматов в вершинах, вообще говоря, полный набор тестов содержит более одного теста. Это означает, что требуется рестарт системы при переходе от одного теста к другому. Такие предположения могут быть оправданы,

например, при имитационном тестировании аппаратуры (simulation-based verification) (см. например, [5]).

Сначала, в разделе 2, изучается простейшая модель системы, в которой может циркулировать только одно сообщение. Даётся формальное определение автомата компонента и формулируются требования к автомату, обеспечивающие детерминизм системы. В разделе 3 модель усложняется: в системе может одновременно передаваться несколько сообщений, но не более одного сообщения по каждой дуге графа связей. Фактически, дуга представляет собой очередь сообщений длины 1. Обобщается формальное определение автомата компонента и формулируются требования к автомату, обеспечивающие детерминизм системы. В заключении намечаются направления дальнейших исследований.

2. Первая модель: только одно сообщение в системе

2.1. Определение модели

В этом разделе мы рассмотрим простейшую модель системы, в которой одновременно может быть не более одного сообщения; если в данный момент времени сообщение есть в системе, оно находится на одной из дуг. Если сообщение передаётся по внутренней дуге $a \to b$, то оно было послано автоматом, находящимся в начале дуги, в вершине a, и будет принято автоматом, находящимся в конце дуги, в вершине b. Если $a \to b$ – это внешняя входная дуга, то a – это окружение системы. Если $a \to b$ – это внешняя выходная дуга, то b – это окружение системы.

Дугу будем рассматривать как очередь сообщений длины 1. По дуге можно послать сообщение, если дуга *пуста* (на ней нет сообщений), и с дуги можно принять сообщение, если дуга не пуста (на ней есть сообщение). Принимая сообщение с дуги, автомат может сам послать любое сообщение, но только одно и только по одной из дуг, выходящих из его вершины, либо не посылать никаких сообщений. Автомат предполагается детерминированным.

Введём формальные определения и обозначения.

Автомат – это набор (S,X,Y,T,s_0), где

S – конечное множество *состояний* автомата,

X – множество *стимулов* (входных символов),

Y – множество *реакций* (выходных символов),

$T \subseteq S \times X \times Y \times S$ – конечное множество *переходов* автомата,

$s_0 \in S$ – *начальное состояние*.

Обозначим: $s \text{—}?x!y \to t \triangleq (s,x,y,t) \in T$, $s \text{—}?x!y \to \triangleq \exists t\,(s,x,y,t) \in T$.

Там, где это не приведёт к недоразумению, мы будем в неформальном тексте сам переход (s,x,y,t) обозначать стрелкой $s \text{—}?x!y \to t$. Состояние t будем

называть *постсостоянием* перехода. Если постсостояние несущественно, то переход *(s,x,y,t)* будем обозначать стрелкой $s—?x!y\rightarrow$.

В графе связей мы допускаем кратные дуги, для различения которых будем помечать их символами из алфавита *Z*. Для простоты будем считать, что все вершины перенумерованы *0,1,2,...,k*, где *k* – число вершин, в которых находятся автоматы, а номер *0* соответствует окружению.

Граф связей определяется как набор $G = (V,Z,E)$, где $V = \{0,1,...,k\}$ – конечное множество вершин, $E \subseteq (V\times Z\times V) \setminus (\{0\}\times Z\times\{0\})$ – конечное множество дуг (у окружения *0* нет дуг-петель). Дуга *(v,z,w)* задаётся начальной вершиной v, пометкой *z* и конечной вершиной *w*. Дуга *(0,z,w)* – внешняя входная дуга, *(v,z,0)* – внешняя выходная дуга. Обозначим:

$I_v = \{(a,z,v) | (a,z,v) \in E\}$ – множество дуг, заканчивающих в вершине *v*,

$O_v = \{(v,z,b) | (v,z,b) \in E\}$ – множество дуг, начинающихся в вершине *v*.

Если $v=0$, то I_0 – это множество внешних входных дуг, O_0 – это множество внешних выходных дуг. Множество внутренних дуг равно $E \setminus (I_0 \cup O_0)$. См. рис. 1.

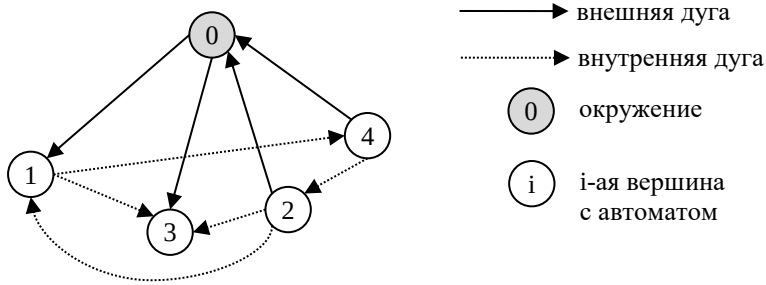


Рис. 1. Граф связей системы автоматов
Fig.1. Communication graph for a system of automatons

Обозначим через *M* множество всех возможных *сообщений*.

Система автоматов – это граф связей *G*, для которого каждой вершине *v* поставлен в соответствие автомат (S_v,X_v,Y_v,T_v,s_{v0}) такой, что

$X_v = \{(i,m) \mid i \in I_v \,\&\, m \in M\}$ и $Y_v = \{(j,m) \mid j \in O_v \,\&\, m \in M\} \cup \{\varnothing\}$.

Такой автомат будем называть автоматом в вершине *v*. Дугу, заканчивающуюся в вершине *v*, будем называть *входной дугой автомата*, а дугу, начинающуюся в вершине *v*, будем называть *выходной дугой автомата*. Стимул автомата – это пара *(i,m)*, где *i* – входная дуга автомата, а *m* – сообщение, принимаемое автоматом по этой дуге. Реакция автомата – это либо пара *(j,m)*, где *j* – выходная дуга автомата, а *m* – сообщение, посылаемое автоматом по этой дуге, либо пустое множество $\varnothing$, означающее отсутствие реакции (пустая реакция).

Определим условия выполнения перехода автомата $s—?x!y\rightarrow t$: 1) автомат находится в состоянии *s*, 2) если $x = (i,m)$, то на входной дуге *i* находится

сообщение m, 3) если $y=(j,m`)$, то выходная дуга $j$ должна быть пуста. В результате выполнения перехода автомат оказывается в постсостоянии $t$, сообщение $m$ принимается и входная дуга $i$ становится пустой, если $y=(j,m`)$, то на выходную дугу j помещается сообщение $m`$.

Замечание 1: о дуге-петле. В рассматриваемой модели в системе имеется не более одного сообщения. Поэтому переход $s—?x!y\rightarrow t$, где $x = (i,m)$ и $y=(j,m`)$, не выполняется из-за того, что выходная дуга j занята, только в том случае, когда $i=j$, то есть сообщение должно быть послано по той же дуге-петле, с которой принимается сообщение. Такой переход (когда $i=j$) никогда не может быть выполнен, и его можно удалить из автомата.

Будем говорить, что автомат *детерминирован*, если в любой момент времени может выполниться не более одного перехода. В этой модели это означает, что состояние и стимул однозначно определяют реакцию и постсостояние перехода: $\forall s,x,y,y`,t,t` \; (s—?x!y\rightarrow t \;\&\; s—?x!y`\rightarrow t` \Rightarrow y=y` \;\&\; t=t`)$. Далее в этом разделе мы будем рассматривать только детерминированные автоматы.

Состоянием системы является набор состояний её автоматов $s_1,s_2,...,s_k$, а также указание, имеется ли какое-либо сообщение на дуге и, если имеется, то какое именно и на какой дуге. Формально состояние системы – это набор $(s_1,s_2,...,s_k,(e,m))$, если на дуге e есть сообщение m, или набор $(s_1,s_2,...,s_k,\varnothing)$, если нет сообщений на дугах. *Финальным* состоянием системы назовём такое её состояние, когда на дугах нет сообщений, то есть состояние вида $(s_1,s_2,...,s_k,\varnothing)$. Начальным состоянием системы будем считать финальное состояние, в котором каждый автомат находится в своём начальном состоянии, то есть состояние $(s_{10},s_{20},...,s_{k0},\varnothing)$.

Тестирование такой системы выполняется как подача в систему внешних стимулов, то есть посылка сообщений из окружения по внешним входным дугам. Для того чтобы оставаться в рамках первой модели (не более одного сообщения в системе), на работу окружения (теста – при тестировании) наложим ограничение: окружение может подавать внешний стимул только тогда, когда система находится в финальном состоянии (в частности, в начальном состоянии), и только один внешний стимул, то есть только одно сообщение по одной внешней входной дуге. Если сообщение в системе находится на внешней выходной дуге, то такое состояние нефинально, но окружение может принять это сообщение, освободив дугу и переведя систему в финальное состояние, а потом посылать следующий стимул. Однако окружение может подать внешний стимул и в том случае, когда система переходит в финальное состояние из-за того, что некоторый автомат принимает стимул, но не выдаёт никакой реакции. Очевидно, что для детерминированных автоматов в вершинах такое ограничение на поведение окружения гарантирует, что система не выйдет за пределы первой модели, то есть в системе будет циркулировать не более одного сообщения.

2.2. Композиция автоматов системы

Определим композицию автоматов при заданном графе связей, то есть автомат, отражающий работу системы в целом. Входными дугами такого автомата системы будут внешние входные дуги из I_0, а выходными дугами – внешние выходные дуги из O_0. Состояния системы описаны выше. Переходы делятся на три категории: 1) *переходы по стимулам* (без выдачи реакции) вида $s—?x!\varnothing\rightarrow t$, 2) *переходы по реакциям* (без приёма стимулов), которые будем понимать как переходы по пустому стимулу и обозначать $s—?\varnothing!y\rightarrow t$, 3) *внутренние переходы* (без приёма стимулов и без выдачи реакций), которые мы будем понимать как переходы по пустому стимулу и пустой реакции вида $s—?\varnothing!\varnothing\rightarrow t$, которые для краткости будем обозначать $s\rightarrow t$.

Определим переходы формально:

1. В финальном состоянии $s=(s_1,s_2,...,s_k,\varnothing)$ окружение может послать по любой внешней входной дуге любое сообщение. Если по дуге $(0,z,v)$ посылается сообщение m, то посылается непустой внешний стимул $x=((0,z,v),m)$. В автомате системы имеется переход $s—?x!\varnothing\rightarrow t$, где $t=(s_1,s_2,...,s_k,x)$ нефинальное состояние.
2. Пусть есть нефинальное состояние $s=(s_1,s_2,...,s_k,((v,z,w),m))$, когда на внешней входной или внутренней дуге (v,z,w) находится сообщение m, то есть $w\neq 0$. Пусть автомат в вершине w находится в состоянии s_w. Обозначим $x=((v,z,w),m)$.
   2.1. Пусть в состоянии s_w имеется переход по приёму стимула x, то есть переход вида $s_w—?x!y\rightarrow t_w$. Если реакция отсутствует, то есть $y=\varnothing$, то в автомате системы имеется внутренний переход $s\rightarrow t$, где $t=(s_1,s_2,...,s_{w-1},t_w,s_{w+1},...,s_k,\varnothing)$ финальное состояние. Если реакция $y=((w,z,w`),m`)$, то в автомате системы имеется внутренний переход $s\rightarrow t`$, где $t`=(s_1,s_2,...,s_{w-1},t_w,s_{w+1},...,s_k,y)$ нефинальное состояние. В состоянии s будет только один переход.
   2.2. Пусть в состоянии s_w нет перехода по стимулу x, то есть, нет перехода вида $s_w—?x!y\rightarrow t_w$. Тогда в состоянии s нет переходов.
3. Пусть есть нефинальное состояние $s=(s_1,s_2,...,s_k,((v,z,0),m))$, когда на внешней выходной дуге $(v,z,0)$ находится сообщение m. Обозначим $y=((v,z,0),m)$. В состоянии s ни один автомат не выполняет перехода, окружение не может послать стимул, так как состояние не финально. Единственное, что может произойти, это приём окружением непустой внешней реакции y, то есть сообщения m, находящегося на дуге $(v,z,0)$. В автомате системы это изображается переходом $s—?\varnothing!y\rightarrow t$, где $t=(s_1,s_2,...,s_k,\varnothing)$ финальное состояние.

Будем говорить, что композиция автоматов системы *детерминирована*, если в каждом её состоянии определено не более одного перехода по каждому стимулу, включая стимул $\varnothing$.

Утверждение 1: В первой модели композиция детерминированных автоматов детерминирована.

Доказательство: Из формального определения переходов композиции следует (рис. 2): 1) Из финального состояния ведут только переходы $s—?x!\varnothing\rightarrow t$ по непустым стимулам без выдачи реакций – не более одного перехода по каждому стимулу. Постсостояние такого перехода нефинально. 2) Из нефинального состояния ведёт не более одного перехода: либо внутренний переход $s\rightarrow t$ с финальным или нефинальным постсостоянием, либо переход $s—?\varnothing!y\rightarrow t$ без приёма стимула с выдачей реакции и финальным постсостоянием.

Утверждение доказано.

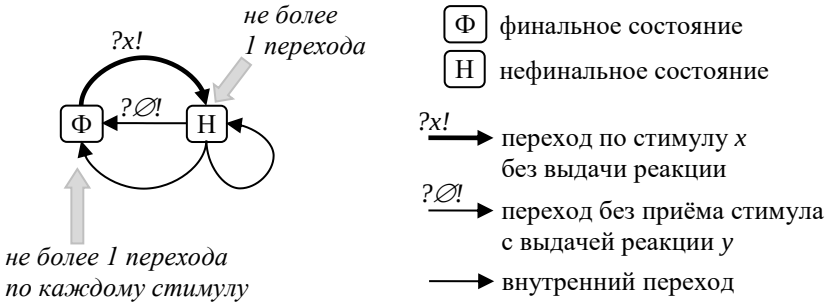


Рис. 2. Типы состояний и переходов автомата системы
Fig. 2. Types of states and transitions of automaton system

Тупиком назовём нефинальное состояние системы, из которого нет перехода. *Состоянием зацикливания* назовём нефинальное состояние системы, расположенное на цикле переходов, не проходящем через финальные состояния. Очевидно, все переходы такого цикла внутренние. На рис. 3 изображены возможные цепочки переходов после приёма системой (непустого) внешнего стимула.

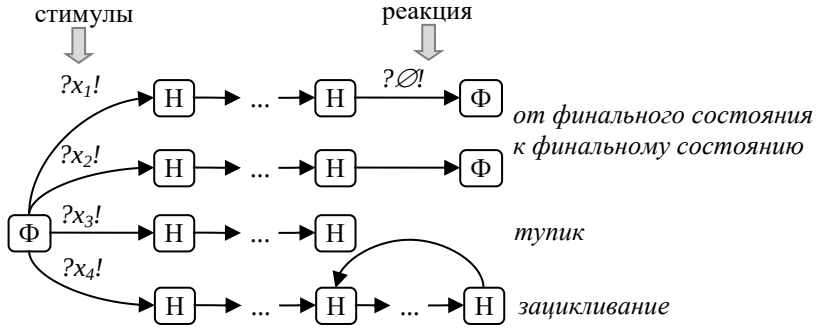


Рис. 3. Типы цепочек переходов в системе после приёма стимула
Fig.4. Types of transitions chains in the system after receiving a stimulus

Наличие пустого стимула, то есть дополнительных внутренних переходов и переходов по выдаче реакции без приёма стимула, отличает автомат системы от того типа автомата, который мы определили выше как автомат в вершине. Однако автомат системы *A* может быть с помощью следующей процедуры *финализации f* преобразован к автомату *f(A)* того же типа, что автомат в вершине:

1) Цепочка переходов $s—?x_1!\varnothing\rightarrow\rightarrow...\rightarrow—?\varnothing!y\rightarrow t_1$, ведущая из финального состояния *s* в финальное состояние t_1 заменяется на один переход $s—?x_1!y\rightarrow t_1$.
2) Цепочка переходов $s—?x_2!\varnothing\rightarrow\rightarrow...\rightarrow t_2$, ведущая из финального состояния *s* в финальное состояние t_2 заменяется на один переход $s—?x_2!\varnothing\rightarrow t_2$.
3) Цепочка переходов $s—?x_3!\varnothing\rightarrow\rightarrow...\rightarrow t_3$, ведущая из финального состояния *s* в тупиковое нефинальное состояние t_3 заменяется на один переход $s—?x_3!\varnothing\rightarrow t_3$.
4) Цепочка переходов $s—?x_4!\varnothing\rightarrow\rightarrow...\rightarrow t_4\rightarrow...\rightarrow t_4$, ведущая из финального состояния *s* в нефинальное состояние зацикливания t_4 с однократным проходом по циклу заменяется на один переход $s—?x_4!\varnothing\rightarrow t_4$. Более строго: в последовательности состояний этой цепочки переход каждое состояние, кроме t_4, встречается по одному разу, а состояние t_4 два раза.

Состояния t_1 и t_2 финальные, а состояния t_3 и t_4 после финализации будут тупиковыми, т.е. в них нет никаких переходов. Автомат *A* будем называть *нефинальной композицией* автоматов компонентов, а автомат *f(A)* – *финальной композицией*. Будем говорить, что два автомата *эквивалентны*, если любой набор тестов, покрывающий все достижимые переходы одного автомата, покрывает все достижимые переходы другого автомата.

Утверждение 2: Финальная композиция *f(A)* детерминированных автоматов детерминирована и эквивалентна нефинальной композиции *A*.

Доказательство: Поскольку в нефинальной композиции *A* из финального состояния выходит не более одного перехода по каждому стимулу, это же будет и в автомате *f(A)*. В автомате *f(A)* нефинальные состояния автомата *A*, оставшиеся достижимыми после финализации, являются терминальными. Поэтому финальная композиция *f(A)* детерминирована. Из нефинального состояния нефинальной композиции выходит не более одного перехода. Поэтому при тестировании проход какой-либо цепочки переходов в автомате *A* из тех, что указаны в процедуре финализации, вызовет в автомате *f(A)* проход соответствующего перехода. И наоборот, проход перехода в автомате *f(A)* соответствует проходу соответствующей цепочки переходов в автомате *A*. Следовательно, *A* и *f(A)* эквивалентны.

Утверждение доказано.

2.3. Генерация тестов

Целью тестирования рассматриваемой системы автоматов является покрытие всех достижимых переходов автоматов компонентов системы. Иногда приходится различать тест как конечную последовательность тестовых воздействий на тестируемую систему и тест как программу, реализующую эти тестовые воздействия, или модель такой программы. Там, где по контексту не ясно, что имеется в виду, мы будем в первом случае писать *тестовая последовательность*. В первой модели тестовая последовательность – это последовательность стимулов, подаваемых из теста (подменяющего собой окружение) в тестируемую систему.

Стимул можно подать в систему только в финальном состоянии. Если стимул приводит систему в нефинальное состояние, при котором сообщение находится на внешней выходной дуге, тест должен сначала принять это сообщение, а потом подать следующий стимул. Поскольку такие действия как приём тестом сообщения с внешней выходной дуги очевидны и обязательны, мы опускаем указания на них в тестовой последовательности.

Какие проверки выполняются при прогоне теста? В финальной композиции переходу $s—?x!y→t$, где $x≠∅$, соответствует цепочка переходов в нефинальной композиции. Первый в цепочке переход имеет вид $s—?x!∅→t`$ и означает для $x=(i,m)$ размещение тестом сообщения $m$ на внешней входной дуге $i$. Последний в цепочке переход имеет вид $s`—?∅!y→t$ и для $y=(j,m`)$ означает передачу сообщения m с внешней выходной дуги j в тест. Остальные переходы внутренние и имеют вид $s``→t``$, только такие переходы нужно проверять, поскольку только их выполнение означает выполнение перехода в каком-либо автомате-компоненте (но не более, чем в одном). Проверяется, выполнил ли этот автомат какой-либо переход или нет, и правильно ли это. Если автомат выполнил переход (и это правильно), то проверяется, правильную ли реакцию автомат выдал, и правильно ли изменилось его состояние.

Прогон теста покрывает некоторое множество переходов автоматов компонентов системы. Поскольку система детерминирована, это множество одно и то же при разных прогонах данного теста (с рестартом между прогонами), поэтому тест достаточно прогонять один раз. Мы будем рассматривать только конечные наборы тестов, после завершения прогона одного теста выполняется рестарт системы и прогоняется следующий тест из набора. Набор тестов покрывает множество переходов автоматов компонентов, которое является объединением множеств переходов автоматов компонентов, покрываемых тестами из этого набора. Набор тестов будем называть *полным*, если он покрывает все переходы автоматов компонентов, достижимые при работе этих компонентов в системе. Ставится задача генерации полного набора тестов.

Для решения этой задачи мы предлагаем использовать любой алгоритм генерации полного набора тестов для одного автомата. Таких алгоритмов предложено довольно много, по сути, они сводятся к построению набора маршрутов, покрывающих граф переходов автомата, достижимых из его начального состояния (см., например, [6]). В качестве такого автомата для наших целей выбирается автомат системы, получаемый с помощью описанной в предыдущем подразделе композиции автоматов. Понятно, что покрывая все достижимые переходы композиционного автомата системы, мы покрываем все достижимые переходы автоматов компонентов. Однако такой набор тестов может оказаться сильно избыточным для решения нашей задачи: покрытие всех достижимых переходов автоматов компонентов не обязательно требует покрытия всех достижимых переходов композиционного автомата системы.

Поэтому предлагается в процессе генерации полного набора тестов для композиционного автомата системы применять *процедуру фильтрации*, которая будет отбрасывать «лишние» тесты. Эта процедура работает следующим образом. С самого начала создаётся пустое множество T генерируемого набора тестов и множество P непокрытых переходов автоматов компонентов, которое сначала равно множеству всех переходов всех автоматов компонентов. Когда генерируется очередной i-ый тест T_i для композиционного автомата системы, вычисляется множество P_i переходов автоматов компонентов, покрываемое этим тестом. Тесту соответствует маршрут в композиционном автомате. Если рассматривается нефинальная композиция, то каждому внутреннему переходу этого маршрута соответствует один переход одного из автоматов компонентов; множество таких переходов и есть множество P_i. Напомним, что при переходе по внешнему стимулу или внешней реакции в автоматах компонентов никаких переходов не происходит. Если рассматривается финальная композиция, то каждому переходу этого маршрута соответствует множество переходов некоторых автоматов компонентов; объединение этих множеств и есть множество P_i. Далее алгоритм фильтрации проверяет, покрывает ли i-ый тест какой-либо новый, ещё не покрытый переход какого-либо автомата компонента. Если $P_i \cap P = \varnothing$, то никаких новых переходов i-ый тест не покрывает, и он отбрасывается. В противном случае тест добавляется к набору тестов $T := T \cup \{T_i\}$, а из множества непокрытых переходов удаляются новые переходы $P := P \setminus P_i$. После того как все тесты сгенерированы и отфильтрованы, получившееся множество T является полным набором тестов, а множество P – множеством недостижимых переходов автоматов компонентов.

Замечание 2: о рестарте. Необходимость в рестарте возникает из-за отсутствия сильной связности графа переходов композиции. Можно выделить два особых случая, когда прогон теста заканчивается в терминальном состоянии композиции. 1) Состояние тупика, когда сообщение «зависает» на внешней входной или внутренней дуге. «Убрать» это сообщение можно только рестартом. 2) Зацикливание, когда сообщение бесконечно циркулирует по

внутренним дугам, что, начиная с какого-то момента времени, не приводит к увеличению множества покрытых переходов автоматов компонентов. «Убрать» это зацикливание можно только рестартом.

2.4. Пример

Утверждение 3: Для любых чисел состояний автоматов компонентов $n_1,n_2,...,n_k$ существует такая система, что число внешних стимулов, которые надо дать для покрытия всех достижимых переходов композиции, равно $\Omega(n_1n_2...n_k)$, а минимальное число внешних стимулов, которых достаточно для покрытия всех достижимых переходов автоматов компонентов, равно $O(n_1+n_2+...+n_k)$.

Доказательство: Рассмотрим систему, изображённую на рис. 4.

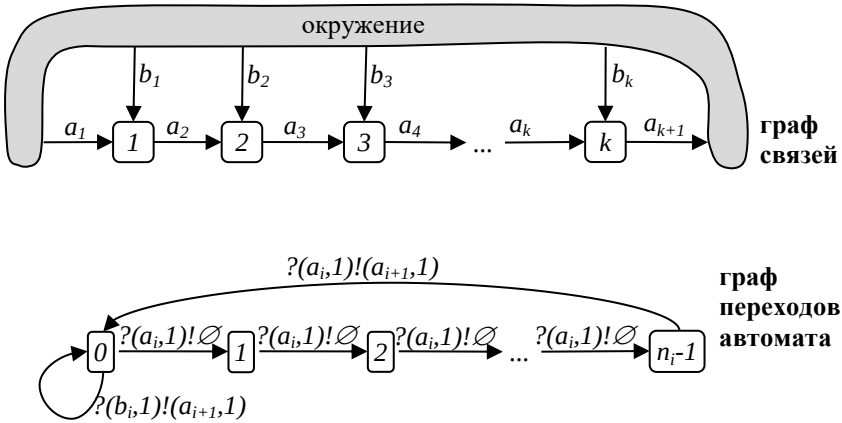


Рис. 4. Пример 1
Fig. 4. Example 1

Здесь имеется единственное сообщение *1*, $M = \{1\}$. Все автоматы в вершинах однотипные и различаются числом состояний n_i. Автомат в вершине i имеет две входные дуги, обозначенные a_i и b_i, и одну выходную дугу a_{i+1}. Соответственно, имеются два стимула $(a_i,1)$ и $(b_i,1)$ и единственная непустая реакция – $(a_{i+1},1)$. Автомат i в любом состоянии $j \neq n_i-1$, получая по входной дуге a_i сообщение *1*, переходит в следующее состояние $j+1$ без выдачи реакции. В последнем состоянии n_i-1 автомат, получая по входной дуге a_i сообщение *1*, переходит в начальное состояние *0* с посылкой сообщения *1* по единственной выходной дуге a_{i+1}. Тем самым, автомат i на каждую порцию из принятых им n_i сообщений по входной дуге a_i посылает одно сообщение следующему автомату $i+1$, или окружению, если $i=k$. Кроме того, имеется переход-петля в состоянии *0* по приёму сообщения по дуге b_i с выдачей сообщения следующему автомату $i+1$, или окружению, если $i=k$.

В финальной композиции нет тупиков и зацикливания, все состояния финальные и имеют вид $s=(s_1,s_2,...,s_k,\varnothing)$, где состояние i-ого автомата $s_i=0..n_{i-1}$. Начальное состояние $s_0=(0,0,...,0,\varnothing)$. Обозначим $s_{max}=(n_1-1,n_2-1,...,n_k-1,\varnothing)$.

Финальное состояние s можно понимать как пару $s=(\tilde{s},\varnothing)$, где $\tilde{s}$ – число, записанное в позиционной системе счисления слева направо от младшей позиции 1 к старшей позиции k: $1,2,...k$, и n_i – основание системы счисления в позиции i.

Рассмотрим в финальной композиции переходы по приёму стимула $(a_1,1)$, то есть по приёму сообщения по внешней входной дуге a_1, ведущей в автомат 1. Для каждого состояния $s\neq s_{max}$ есть переход $(\tilde{s},\varnothing)\text{—}?(a_1,1)!\varnothing\rightarrow(\tilde{s}+1,\varnothing)$, а также есть переход $(\tilde{s_{max}},\varnothing)\text{—}?(a_1,1)!(a_{k+1},1)\rightarrow(\tilde{s_0},\varnothing)$. Это не все переходы композиции, но число таких переходов равно произведению $n_1n_2...n_k$. Следовательно, число переходов в финальной композиции не менее произведения $n_1n_2...n_k$. Поскольку в финальной композиции один внешний стимул вызывает ровно один переход, число внешних стимулов, которые надо дать для покрытия композиции, равно $\Omega(n_1n_2...n_k)$.

Как наиболее быстро покрыть все переходы автомата i? Для этого достаточно 1) n_i раз послать сообщение по дуге a_i и 2) один раз по дуге b_i. Для автомата 0 пункт 1 тест может выполнить непосредственно, поскольку дуга a_1 внешняя. Для автомата $i>1$ пункт 1 можно выполнить, посылая n_i раз сообщение по внешней дуге b_{i-1} в предыдущий автомат $i-1$. Пункт 2 тест также может выполнить непосредственно, поскольку дуга b_i внешняя. В целом получается следующий тест:

1. n_1 раз посылается сообщение в автомат 1 по внешней дуге a_1,

   автомат 1 проходит цикл переходов по состояниям $0,...,n_1-1,0$, во время последнего перехода будет послано сообщение по дуге a_2, при получении этого сообщения автомат 2 переходит из состояния 0 в состояние 1, остальные автоматы находятся в состоянии 0;
2. n_2-1 раз посылается сообщение в автомат 1 по внешней дуге b_1,

   автомат 1 проходит n_2-1 раз по петле в состоянии 0, каждый раз посылая сообщение по дуге a_2, при получении всех этих сообщений автомат 2 проходит цепочку переходов из состояния 1 в состояние 0, при последнем переходе посылается сообщение по дуге a_3, при получении этого сообщения автомат 3 переходит в состояние 1, остальные автоматы остаются в состоянии 0;
3. n_3-1 раз посылается сообщение в автомат 2 по внешней дуге b_2,

   автомат 2 проходит n_3-1 раз по петле в состоянии 0, каждый раз посылая сообщение по дуге a_3, при получении всех этих сообщений автомат 3 проходит цепочку переходов из состояния 1 в состояние 0, при последнем переходе посылается сообщение по дуге a_4, при получении этого сообщения автомат 4 переходит в состояние 1, остальные автоматы остаются в состоянии 0;

...

k-1. n_{k-1}*-1* раз посылается сообщение в автомат *k-2* по внешней дуге b_{k-2}, автомат *k-2* проходит n_{k-1}*-1* раз по петле в состоянии *0*, каждый раз посылая сообщение по дуге a_{k-1}, при получении всех этих сообщений автомат *k-1* проходит цепочку переходов из состояния *1* в состояние *0*, при последнем переходе посылается сообщение по дуге a_k, при получении этого сообщения автомат *k* переходит в состояние *1*, остальные автоматы остаются в состоянии *0*;

k. n_k*-1* раз посылается сообщение в автомат *k-1* по внешней дуге b_{k-1}, автомат *k-1* проходит n_k*-1* раз по петле в состоянии *0*, каждый раз посылая сообщение по дуге a_k, при получении всех этих сообщений автомат *k* проходит цепочку переходов из состояния *1* в состояние *0*, при последнем переходе посылается внешнее сообщение по дуге a_{k+1}, которое принимается тестом, остальные автоматы остаются в состоянии *0*;

k+1. *1* раз посылается сообщение в автомат *k* по внешней дуге b_k, автомат *k* проходит петлю в состоянии *0* и по внешней выходной дуге a_{k+1} посылает сообщение.

Длина тестовой последовательности $n_1+(n_2-1)+...+(n_k-1)+1 = (n_1+...+n_k)-k+2 = \boldsymbol{O}(n_1+...+n_k)$.

Утверждение доказано.

3. Вторая модель: обобщение автомата в вершине

3.1. Определение модели

В этом разделе мы сохраняем предположения о дуге (дуга реализует очередь сообщений длины 1), но обобщаем понятие автомата в вершине. Такой обобщённый автомат может принимать одновременно несколько сообщений по нескольким (необязательно всем) входным дугам, но не более одного сообщения по каждой дуге, и посылать несколько сообщений по нескольким (необязательно всем) выходным дугам, но не более одного сообщения по каждой дуге.

Дадим формальное определение автомата в вершине. Автомат в вершине такой же, как в разделе 2, но с изменёнными множествами стимулов и реакций, поскольку стимул и реакция теперь могут содержать несколько сообщений принимаемых или посылаемых по нескольким дугам. Формально для графа связей $G=(V,Z,E)$ и множества сообщений M автомат в вершине v – это такой автомат (S_v,X_v,Y_v,T_v,s_{v0}), что

$X_v = \{x \mid \exists f\, x \subseteq f \;\&\; f \in M^{I_v}\}$,

$Y_v = \{y \mid \exists f\, y \subseteq f \;\&\; f \in M^{O_v}\}$, где через A^B обозначено множество всех отображений из B в A.

Стимул автомата – это частично определённое отображение $x{:}I_v \to M$, которое каждой входной дуге $i \in \mathbf{Dom}(x)$ ставит в соответствие сообщение $x(i)$, принимаемое по этой дуге. Реакция автомата – это частично определённое отображение $y{:}O_v \to M$, которое каждой выходной дуге $j \in \mathbf{Dom}(y)$ ставит в соответствие сообщение $y(j)$, посылаемое по этой дуге.

Для описания состояния входных и выходных дуг автомата в вершине v введём два частично-определённых отображения $x_v^{\#}{:}I_v \to M$ и $y_v^{\#}{:}O_v \to \{M\}$. Первое отображение каждой непустой входной дуге $i \in I_v$ ставит в соответствие сообщение $m \in M$, находящееся на этой дуге. Второе отображение каждой пустой выходной дуге $j \in O_v$ ставит в соответствие множество сообщений M. Эти отображения $x_v^{\#}$ и $y_v^{\#}$ порождают множества *потенциальных* стимулов и *потенциальных* реакций, которые автомат может, соответственно, принять и послать при данном состоянии входных и выходных дуг, если, конечно, в текущем состоянии автомата определены соответствующие переходы:

$X_v^{\#} = \{x{:}I_v \to M \mid \mathbf{Dom}(x) \subseteq \mathbf{Dom}(x_v^{\#}) \;\&\; \forall i \in \mathbf{Dom}(x)\ x(i) = x_v^{\#}(i)\}$,

$Y_v^{\#} = \{y{:}O_v \to M \mid \mathbf{Dom}(y) \subseteq \mathbf{Dom}(y_v^{\#})\}$.

Теперь можно формально определить условие выполнения перехода $s \text{—}?x!y \to t$: $x \in X_v^{\#} \;\&\; y \in Y_v^{\#}$.

3.2. Детерминированный автомат

Для того чтобы система автоматов была детерминирована, потребуем детерминированности каждого из этих автоматов. Прежде всего, как и для первой модели, состояние и стимул должны однозначно определять реакцию и постсостояние перехода:

1) $\forall s,x,x`,y,y`,t,t`\quad s \text{—}?x!y \to t \;\&\; s \text{—}?x!y` \to t` \Rightarrow y=y` \;\&\; t=t`$.

В отличие от первой модели во второй модели распределение сообщений по входным дугам автомата не однозначно определяет стимул, который может принять автомат, поскольку автомат может не принимать сообщения с некоторых входных дуг, даже если дуги не пусты. Например, если на входных дугах i_1 и i_2 находятся сообщения m_1 и m_2, соответственно, то может быть принят любой из стимулов $\{(i_1,m_1),(i_2,m_2)\}$, $\{(i_1,m_1)\}$, $\{(i_2,m_2)\}$ или $\varnothing$. Если в автомате в текущем состоянии есть переходы хотя бы по двум из этих стимулов, то поведение автомата недетерминировано: может быть выбран приём любого из этих стимулов.

Будем говорить, что два стимула x и $x`$ *совместимы*, и обозначать $x \approx x`$, если при некоторых сообщениях на входных дугах автомата, т.е. при некотором отображении $x_v^{\#}$, автомат может принять любой из этих стимулов, т.е. $x \in X_v^{\#}$ и $x` \in X_v^{\#}$. Формально: $x \approx x` \triangleq \forall i \in \mathbf{Dom}(x) \cap \mathbf{Dom}(x`)\ x(i) = x`(i)$. Заметим, что все стимулы во множестве $X_v^{\#}$ совместимы друг с другом.

Отсюда вытекает второе требование детерминизма автомата:

2) Нет переходов из одного состояния по разным совместимым стимулам:

$\forall s,x,x`,y,y` \quad x \neq x` \,\&\, x \approx x` \Rightarrow \neg(s \text{—} ?x!y\rightarrow \,\&\, s \text{—} ?x`!y`\rightarrow)$.

Утверждение 4: Если выполнены оба требования детерминизма, то состояние s_v автомата в вершине v и отображения $x_v^{\#}$ и $y_v^{\#}$ однозначно определяют, выполняет ли автомат какой-либо переход, и, если выполняет, то сам переход, т.е. однозначно определяют принимаемый стимул $x_v^{\wedge}$, посылаемую реакцию $y_v^{\wedge}$ и постсостояние $t_v^{\wedge}$.

Доказательство: По определению отображения $x_v^{\#}$ и $y_v^{\#}$ однозначно определяют множества $X_v^{\#}$ и $Y_v^{\#}$. Из второго требования детерминизма следует, что при любом распределении $x_v^{\#}$ сообщений на входных дугах автомата, порождающим множество $X_v^{\#}$ потенциальных стимулов, не более одного из этих стимулов $x_v \in X_v^{\#}$ может быть принято автоматом в данном состоянии s_v. Такой стимул x_v будем называть *выбираемым*, он может отсутствовать. Правда, это не означает, что выбираемый стимул x_v (если он есть) обязательно будет принят автоматом, поскольку выполнение перехода с приёмом этого стимула обусловлено возможностью послать реакцию. Рассмотрим все случаи поведения автомата в состоянии s_v при заданных $x_v^{\#}$ и $y_v^{\#}$ (однозначно определяющих $X_v^{\#}$ и $Y_v^{\#}$), определяя, будет ли выполнен переход и, если будет, то сам переход, т.е. принимаемый стимул $x_v^{\wedge}$, посылаемую реакцию $y_v^{\wedge}$ и постсостояние $t_v^{\wedge}$.

1. Имеется выбираемый стимул $x_v \in X_v^{\#}$, он единственный по 2-ому требованию детерминизма.

1.1. Для некоторой реакции y есть переход $s_v \text{—} ?x!y\rightarrow t$ и $y \in Y_v^{\#}$, т.е. реакция y может быть послана (нужные выходные дуги пусты). Автомат выполнит этот переход (он единственный по 1-ому требованию детерминизма), $x_v^{\wedge}=x$, $y_v^{\wedge}=y$, $t_v^{\wedge}=t$.

1.2. Для любой реакции y либо нет перехода $s \text{—} ?x!y\rightarrow$, либо $y \notin Y_v^{\#}$. Автомат не выполнит никакого перехода, $x_v^{\wedge}=\varnothing$, $y_v^{\wedge}=\varnothing$, $t_v^{\wedge}=s_v$.

2. Нет выбираемого стимула. Автомат не выполнит никакого перехода, $x_v^{\wedge}=\varnothing$, $y_v^{\wedge}=\varnothing$, $t_v^{\wedge}=s_v$.

Утверждение доказано.

3.3. Интерпретация 1-ой модели в терминах 2-ой модели

Утверждение 5: Автомат 1-ой модели (раздел 2) – частный случай автомата 2-ой модели (раздел 3).

Доказательство: Утверждение очевидно, если применить интерпретацию 1-ой модели в терминах 2-ой модели так, как отображено на табл.1., и сравнить определения выполнимости перехода автомата в обеих моделях.

Табл. 1. Интерпретация 1-ой модели в терминах 2-ой модели
Table. 1. Interpretation of the first model in terms of the second model

	1-ая модель	интерпретация	особенности 1-ой модели
стимул	$x=(i,m)$	$x=\{(i,m)\}$	принимается ровно одно сообщение
реакция	$y=(j,m)$ или $y=\varnothing$	$y=\{(j,m)\}$ или $y=\varnothing$	посылается не более одного сообщения
переход	$s—?x!y\rightarrow t$	$s—?x!y\rightarrow t$	

Утверждение доказано.

В то же время детерминированный автомат в 1-ой модели, вообще говоря, не является детерминированным автоматом во 2-ой модели. Например, в 1-ом случае могут быть два перехода $s—?(i,m)!y\rightarrow t$ и $s—?(i`,m`)!y\rightarrow t$, где $i\neq i`$, наличие которых во 2-ой модели означает нарушение 2-го требования детерминизма. Это объясняется тем, что 1-ая модель ограничивает циркуляцию сообщений в системе, допуская только одно сообщение, поэтому требования детерминизма в 1-ой модели оказываются слабее, чем во 2-ой.

3.4. Композиция автоматов системы

Определим композицию детерминированных автоматов с данным графом связей. Результатом композиции будет автомат (S,X,Y,T,s_0), отражающий работу системы в целом, включая все автоматы-компоненты и все дуги. Поскольку теперь, в отличие от первой модели, сообщения могут быть на нескольких дугах одновременно, но не более одного сообщения на каждой дуге, состояние системы – это набор состояний её автоматов $s_1,s_2,...,s_k$, а также распределение сообщений по дугам графа связей. Формально состояние системы – это набор $s = (s_1,s_2,...,s_k,D)$, где $D:E\rightarrow M$ – частично-определённое отображение, которое для каждой непустой дуги указывает находящееся на ней сообщение. Начальным состоянием системы, как и для первой модели, будем считать состояние, в котором каждый автомат находится в своём начальном состоянии, а сообщений на дугах нет, то есть состояние $s_0 = (s_{10},s_{20},...,s_{k0},\varnothing)$.

Переходы композиции из T определяются в зависимости от того, предполагается ли синхронный или асинхронный режим работы автоматов. В каждом состоянии системы имеется множество A автоматов, которые могут выполнять переходы. В синхронном режиме за один такт срабатывают все автоматы из A, а в асинхронном – только один автомат (вообще говоря, некоторое подмножество A), выбираемый недетерминированным образом. Поскольку в рамках данной статьи нас интересуют только детерминированные системы, асинхронный режим мы далее не рассматриваем. Заметим, что для первой модели эти два режима работы не различаются, поскольку в каждом состоянии системы может сработать не

более одного автомата. В отличие от первой модели теперь окружение может подавать в систему следующий стимул не только в финальном состоянии, когда все дуги пусты, но и в любом состоянии, при котором пусты те внешние входные дуги, по которым окружение хочет послать сообщения.

Определим переходы композиции формально. В состоянии системы s окружение может послать в систему сообщение по любой пустой внешней входной дуге, а также принять сообщение с любой занятой внешней выходной дуги. Это определяет допустимые стимулы и реакции. Стимул $x:I_0 \to M$ *допустимо послать* из окружения в систему, если $\mathbf{Dom}(x) \cap \mathbf{Dom}(D) = \varnothing$, в частности, всегда допустим пустой стимул $x=\varnothing$. Реакцию $y:O_0 \to M$ *допустимо принять* из системы в окружение, если $y \subseteq D$, в частности, всегда допустима пустая реакция $y=\varnothing$. Если стимул x и реакция y допустимы, то в композиции определяется переход $s \text{—}?x!y\to t^\wedge$, где $t^\wedge = (t_1^\wedge, t_2^\wedge, ..., t_k^\wedge, D^\wedge)$. Определим для каждой вершины v постсостояние $t_v^\wedge$, а также определим $D^\wedge$.

Для $v=1..k$ рассмотрим автомат в вершине v. Для данного состояния системы s отображение $x_v^\#$ определяется однозначно как сужение отображения D на множество I_v входных дуг v-ого автомата: $x_v^\# = \{(i,m) | (i,m) \in D \ \& \ i \in I_v\}$, и отображение $y_v^\#$, которое каждой пустой выходной дуге v-го автомата ставит в соответствие множество M всех сообщений: $y_v^\# = \{(j,M) \mid j \in O_v \setminus \mathbf{Dom}(D)\}$. По утверждению 4 при заданных s_v, $x_v^\#$ и $y_v^\#$ автомат выполняет не более одного перехода, и однозначно определяются принимаемый стимул $x_v^\wedge$ и посылаемая реакция $y_v^\wedge$, а также постсостояние $t_v^\wedge$. Именно это постсостояние и записывается в $t^\wedge$.

Определим $D^\wedge$.

Сначала положим $D^\wedge := D$.

Рассмотрим, как должно меняться расположение сообщений на дуге $e=(i,z,j)$.

1) В состоянии s дуга e была пустой, т.е. $e \notin \mathbf{Dom}(D)$. Тогда при $j \neq 0$ j-ый автомат не принимает с неё сообщения, т.е. $e \notin \mathbf{Dom}(x_j^\wedge)$. При $i \neq 0$ i-ый автомат посылает по этой дуге сообщение m, если $e \in \mathbf{Dom}(y_i^\wedge)$ & $y_i^\wedge(e)=m$; тогда пара (e,m) добавляется в $D^\wedge$. Если $j=0$, то $e \notin \mathbf{Dom}(y)$. Если $i=0$, то $e \in \mathbf{Dom}(x) \Rightarrow e \in \mathbf{Dom}(D^\wedge)$ & $D^\wedge(e)=x(e)$, т.е. если окружение посылает по внешней входной дуге e сообщение $x(e)$, то пара $(e,x(e))$ добавляется в $D^\wedge$.

2) В состоянии s на дуге e было сообщение m, т.е. $e \in \mathbf{Dom}(D)$ & $D(e)=m$. Тогда при $j \neq 0$ j-ый автомат принимает это сообщение, если $e \in \mathbf{Dom}(x_j^\wedge)$ & $x_j^\wedge(e)=m$; тогда пара $(e,x_j^\wedge(e))$ удаляется из $D^\wedge$. При $i \neq 0$ i-ый автомат не может послать по этой дуге никакого сообщения, поскольку дуга в состоянии s занята, т.е. $e \notin \mathbf{Dom}(y_i^\wedge)$. Если $j=0$, то $e \in \mathbf{Dom}(y) \Rightarrow e \notin \mathbf{Dom}(D^\wedge)$ & $D(e)=y(e)$, т.е. если окружение принимает по внешней выходной дуге e сообщение $y(e)$, то пара $(e,y(e))$ удаляется из $D^\wedge$. Если $i=0$, то $e \notin \mathbf{Dom}(x)$.

Тем самым, $D^\wedge = (D \cup x \cup y_1^\wedge \cup ... \cup y_k^\wedge) \setminus (y \cup x_1^\wedge \cup ... \cup x_k^\wedge)$.

Будем говорить, что такая композиция детерминированных автоматов детерминирована, если в каждом достижимом (из начального состояния) состоянии каждая пара допустимых стимула и реакции однозначно определяет постсостояние системы, т.е. выполняемый переход.

Утверждение 6: Во второй модели композиция детерминированных автоматов детерминирована.

Доказательство: Достаточно показать, что 1) каждый автомат вершины графа связей может выполнить не более одного перехода, и 2) распределение $D^{\wedge}$ сообщений по дугам определяется однозначно. И то, и другое следует из утверждения 4 и определения композиции.

Утверждение доказано.

3.5. Генерация тестов

При тестировании на каждом такте тест посылает в тестируемую систему сообщения по пустым внешним входным дугам (не обязательно всем) и принимает от системы по занятым внешним выходным дугам (не обязательно всем) имеющиеся на них сообщения. Определим композицию системы и теста. Состояние композиции – это пара состояний системы и теста. Переход композиции соответствует паре из допустимого стимула и допустимой реакции, что определяет возможные постсостояния системы и теста, т.е. возможные постсостояния композиции. Сам переход композиции является внутренним, т.е. ничем не помечен, поскольку композиция системы и теста замкнута и ни с чем не взаимодействует. Формально в композиции системы и теста переходы определяются следующим правилом композиции: $s—?x!y\rightarrow t \;\&\; s`—?y!x\rightarrow t` \vdash ss`\longrightarrow tt`$.

Если тест и система оба детерминированы, то их композиция тоже будет детерминирована в следующем смысле: в каждом её состоянии определено не более одного перехода.

Тестовая последовательность – это конечная последовательность пар $(x_1,y_1),...,(x_n,y_n)$, которой в тестируемой системе соответствует маршрут (цепочка смежных переходов) $s_0—?x_1!y_1\rightarrow s_1—...\rightarrow s_{n-1}—?x_n!y_n\rightarrow s_n$, а в тесте – маршрут $s`_0—?y_1!x_1\rightarrow s`_1—...\rightarrow s`_{n-1}—?y_n!x_n\rightarrow s`_n$. Каждое состояние s_i и $s`_i$ соответствует префиксу тестовой последовательности длиной i.

Для такой тестовой последовательности детерминированной тест состоит только из указанной выше цепочки переходов. Детерминированный тест выбирает только одну допустимую реакцию, принимаемую от системы, как подмножество сообщений на занятых внешних выходных дугах системы, и посылает в систему только один допустимый стимул как набор сообщений, размещаемых на подмножестве пустых внешних входных дуг системы.

Какие проверки выполняются при прогоне теста? На каждом i-ом такте проверяется следующее. 1) Выполнился ли в тесте переход $s`_i—?y_{i+1}!x_{i+1}\rightarrow s`_{i+1}$; если не выполнился, то фиксируется ошибка. 2) Для

каждого автомата в системе проверяется, правильно ли изменилось его состояние, правильно ли он выполнил приём стимула (с тех или не тех входных дуг принял сообщения), правильно ли он выполнил выдачу реакции (на те или не на те выходные дуги послал сообщения, и правильные ли эти сообщения).

Как и в случае первой модели, целью тестирования является покрытие всех достижимых переходов автоматов компонентов системы, а генерация тестов основана на фильтрации набора тестов, сгенерированного для тестирования композиционной системы. В то же время тестирование для второй модели имеет ряд отличий от тестирования для первой модели.

Во-первых, теперь тест может посылать сообщения в систему не только в её финальном состоянии, а в любом состоянии.

Во-вторых, если в первой модели сообщение не посылается в систему в её финальном состоянии, то ничего не происходит: система не меняет своего состояния. Во второй модели система в финальном состоянии (т.е. при отсутствии сообщений на дугах), вообще говоря, может продолжать работать, поскольку допустимы переходы, при которых сообщения не принимаются.

В-третьих, если в первой модели сообщение (единственное в системе) находится на внешней выходной дуге, то также ничего не происходит, т.е. система не меняет своего состояния, до тех пор, пока тест не примет это сообщение и, тем самым, не переведёт систему в финальное состояние. Именно поэтому такое действие теста очевидно и обязательно, и в тестовой последовательности для 1-ой модели опущены принимаемые от системы сообщения, т.е. реакции системы. Во второй модели, с одной стороны, тестируемая система может выполнять переходы и в том случае, когда на внешних выходных дугах имеются сообщения, а, с другой стороны, тест может принимать часть сообщений с внешних выходных дуг.

В-четвёртых, в первой модели выполнимость перехода зависит только от наличия нужного сообщения на нужной входной дуге, поскольку выходные дуги всегда свободны. По замечанию 1 исключение составляет только переход $s—?(i,m)!(i,m`)\rightarrow t$ по приёму сообщения с дуги-петли i с одновременной посылкой сообщения по этой же дуге, такой переход никогда не выполним. Во второй модели также всегда невыполним аналогичный переход $s—?x!y\rightarrow t$, где некоторая дуга-петля $i \in \boldsymbol{Dom}(x) \cap \boldsymbol{Dom}(y)$, с приёмом сообщения $x(i)$ с дуги-петли и посылкой сообщения $y(i)$ по ней же. Однако во второй модели в системе может быть несколько сообщений и выполнимость перехода всё же зависит от того, заняты или нет выходные дуги, по которым нужно послать сообщения.

Замечание 3: о приёме тестом сообщений от системы. Приём или не приём тестом сообщений с внешних выходных дуг системы, по сути, является дополнительным тестовым воздействием на систему, поскольку меняет её поведение (выполнимость тех или иных переходов). Для того чтобы описать это формально, такое тестовое воздействие нужно трактовать как часть

стимула, т.е. нужно приём тестом сообщений от системы заменить посылкой тестом сообщений в систему. Поведение дуги как очереди длины 1 можно изобразить в виде автомата с одним входом и одним выходом, изображённого на рис. 5 вверху.

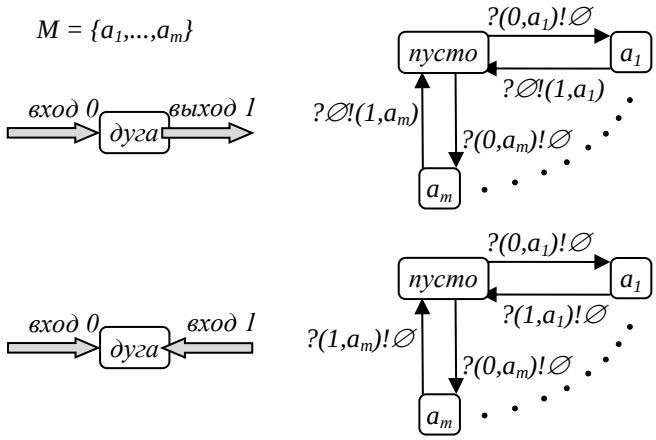


Рис. 5. Автомат очереди длины 1
Fig. 5. Automata of a queue of length 1

Такой автомат дуги взаимодействует синхронно с автоматами начальной и конечной вершин дуги: либо одновременно происходит посылка сообщения по дуге из автомата начала дуги и приём этого сообщения в автомат дуги, либо одновременно происходит посылка сообщения из автомата дуги и приём этого сообщения по дуге в автомате конца дуги. Для того чтобы в тесте заменить приём посылкой, достаточно преобразовать автоматы внешних выходных дуг, объявляя выход *1* входом *1* и заменяя на переходах автомата дуги пометку $?\varnothing!(1,a_i)$ – на пометку $?(1,a_i)!\varnothing$ (рис. 5 внизу). Соответственно, в автомате теста все входы превращаются в выходы, а на переходах вместо *?y!x* записывается $?\varnothing!(y \cup x)$, т.е. для тестируемой системы вместо стимула *x* будет стимул $y \cup x$.

Замечание 4: о внешних реакциях. Для теста всё равно, какую именно реакцию *y* посылает система, поскольку правильность реакции *y* проверяется во время проверки правильности перехода системы, т.е. правильности сообщений, которые автоматы компонентов посылают на свои выходные дуги, которые являются внешними выходными дугами. Поэтому после преобразования, когда реакции становятся частью стимулов, для стимула $y \cup x$ в паре $(j,m) \in y$, где *j* – выходная дуга теста, бывшая до преобразования входной дугой теста, сообщение *m* можно заменить на специально выделенное сообщение «*принять*».

4. Заключение

В заключение сформулируем направления дальнейших исследований.

1) Предложенный алгоритм фильтрации не обязательно даёт оптимальный полный набор тестов. Оптимальность может пониматься как минимальное число или минимальная суммарная длина тестовых последовательностей. Соответственно, возникает задача оптимизации, т.е. поиска оптимального набора, которая, вообще говоря, сводима к задаче о поиске минимального покрытия ([7], [8]).

2) В этой статье определена композиция автоматов системы, наиболее удобная для тестирования. Её результатом является автомат без входных и выходных дуг, поскольку они «погружены» внутрь системы. Поэтому такой автомат не может использоваться как компонент при построении более сложной системы. В дальнейшем мы предполагаем (в рамках более общей модели) определить композицию другим способом так, чтобы её результатом был автомат с входными и выходными дугами, т.е. автомат, который можно помещать в вершину графа связей.

3) В этой статье мы предполагали, что дуга реализует очередь длины 1. Это легко обобщается на случай очередей большей длины, в том числе неограниченных очередей; более того, разные дуги могут реализовывать очереди разной длины. Длина очереди является статическим атрибутом дуги и задаётся при задании графа связей. Для второй модели отображение *D*, описывающее состояние дуг, теперь должно отображать дугу в последовательность сообщений, находящихся в очереди, в частности, пустая последовательность соответствует пустой дуге.

Однако для очереди длины больше 1 возникает другая проблема. В конце 3-го раздела определён автомат дуги, которая понимается как очередь длины 1. Здесь нам, что называется, «повезло», поскольку очередь большей длины невозможно изобразить в виде автомата того типа, что определён в разделе 3. Дело в том, что в «промежуточном» состоянии, когда очередь не пуста, но и не полностью заполнена, вместе с каждым переходом $s—?(0,x)!(1,y)\rightarrow t$, при котором в конец очереди вставляется сообщение x, а из головы очереди удаляется сообщение y, должны быть два других перехода. Второй переход – это переход $s—?(0,x)!\varnothing\rightarrow t$, при котором в конец очереди вставляется сообщение x, а из головы очереди не удаляется сообщение y, поскольку конец дуги его не принимает. Третий переход – это переход $s—?\varnothing!(1,y)\rightarrow t$, при котором из головы очереди удаляется сообщение y, но в конец очереди не вставляется никакого сообщения, поскольку начало дуги не посылает по дуге никакого сообщения. Однако в рамках второй модели наличие первого и второго переходов противоречит 1-ому требованию детерминизма: одному стимулу $(0,x)$ соответствуют две реакции $(1,y)$ и $\varnothing$. Кроме того, третий переход – это переход по пустому стимулу, который совместим с любым другим стимулом, в том числе со стимулом $(0,x)$, что противоречит 2-ому

требованию детерминизма. Решение этой проблемы мы предполагаем искать на пути подходящего обобщения модели автомата.

4) Кроме очереди длины больше 1, могут быть и другие дуги графа связей. Например, очередь с приоритетами, стек и т.п. Мы предполагаем обобщить понятие дуги с помощью определения автомата дуги. Для детерминизма системы, по-видимому, к автомату дуги придётся предъявить дополнительные (по сравнению с автоматом вершины) требования. Композиция должна быть определена для пары автоматов, выход одного из которых соединён с входом другого. На таком соединении происходит синхронное взаимодействие автоматов, когда один автомат посылает сообщение тогда и только тогда, когда другой автомат это сообщение принимает.

5) При тестировании, как обычно, возникает задача «огрубления», когда проверяется не «всё подряд», а проверка делается выборочно. Это называют факторизацией тестирования [9]. Например, для проверки правильности реализации числовой функции домены её аргументов разбиваются на поддомены, и из каждого поддомена выбирается хотя бы одно число для проверки. В терминах рассматриваемой здесь задачи факторизация означает, что на переходах автомата компонента определяется отношение эквивалентности, и считается, что достаточно проверить хотя бы один переход из класса эквивалентности.

6) В данной статье предлагается решение проблемы тестирования компонентов только детерминированной системы автоматов. Одним из направлений дальнейших исследований могло бы стать исследование проблемы недетерминизма. Более точно, ставится задача определить такие ограничения на недетерминизм системы и/или составляющих её автоматов, которые позволяли бы выполнять полное тестирование за конечное время. Для автономного тестирования, когда автомат находится под непосредственным управлением теста (не в контексте окружающей его части системы), предложенные неплохие решения этой задачи ([10], [11], [12], [13]).

7) В данной статье предполагается, что известно, каким должен быть автомат каждого компонента: задан граф переходов автомата с точностью до изоморфизма. В более общем случае между автоматом компонента в реализации и автоматом компонента в спецификации задаётся то или иное отношение конформности, которое слабее изоморфизма: квази-редукция, симуляция и т.п. Это требует более сложного алгоритма тестирования. Если тестирование компонента автономное, т.е. не в контексте других компонентов системы и связующих их дуг, то возможно полное тестирование за конечное время конформности типа редукции или слабой симуляции ([2], [3], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]). Для составной системы возникает проблема декомпозиции системных требований, называемая также проблемой несохранения конформности. Она заключается в том, что композиция реализаций компонентов, конформных спецификациям этих компонентов, в общем случае неконформна спецификации системы, в

частности, композиции спецификаций компонентов. Этой проблеме посвящён ряд работ ([2], [4], [21]), но возникает задача переосмысления предложенных решений для рассматриваемой в этой статье задачи тестирования компонентов системы при условии, что верна гипотеза о связях.

Список литературы

[1]. Revised Working Draft on “Framework: Formal Methods in Conformance Testing”. JTC1/SC21/WG1/Project 54/1, ISO Interim Meeting, ITU-T on, Paris, 1995 г.

[2]. И.Б.Бурдонов, Косачев А.С., В.В.Кулямин. Теория соответствия для систем с блокировками и разрушением. «Физ-мат лит» Наука, Москва, 2008 г., 412 стр.

[3]. И.Б.Бурдонов. Теория конформности (функциональное тестирование программных систем на основе формальных моделей). LAP Lambert Academic Publishing, 2011 г., 428 стр.

[4]. И.Б.Бурдонов, А.С.Косачев. Пополнение спецификации для ioco. Программирование, 2011 г., №1, стр. 3-18.

[5]. А. Камкин, М. Чупилко. Обзор современных технологий имитационной верификации аппаратуры. Программирование, 2011 г., №3, стр. 42-49.

[6]. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай. Программирование, 2003 г., №5, стр. 59-69.

[7]. Ананий В. Левитин. Алгоритмы: введение в разработку и анализ. М.: «Вильямс», 2006 г., стр. 160-163, ISBN 0-201-74395-7.

[8]. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ. 2-ое издание. М.: «Вильямс», 2006 г., стр. 456-458, ISBN 0-07-013151-1.

[9]. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ. Программирование. 2000 г., № 2, стр.12-28.

[10]. И.Б. Бурдонов, А.С. Косачев. Полное тестирование с открытым состоянием ограниченно недетерминированных систем. Программирование, 2009 г., №6, стр. 3-18.

[11]. И.Б.Бурдонов, А.С.Косачев. Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования. Вестник Томского Государственного Университета, № 2(15), 2011 г., стр. 89-98.

[12]. И.Б. Бурдонов, А.С. Косачев. Тестирование конформности на основе соответствия состояний. Труды ИСП РАН, volume 18, 2010 г., стр. 183-220.

[13]. И.Б.Бурдонов, А.С.Косачев, Безопасное тестирование симуляции систем с отказами и разрушением. Моделирование и анализ информационных систем, том 17(4), 2010 г., стр. 27-40.

[14]. I.B.Bourdonov, A.S.Kossatchev, V.V.Kuliamin. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proceedings of the Workshop on Model Based Testing (MBT 2004), Elsevier, 2006.

[15]. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Формализация тестового эксперимента. Программирование, 2007 г., №5, стр. 3-32.

[16]. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Безопасность, верификация и теория конформности. Материалы второй международной научной конференции по проблемам безопасности и противодействия терроризму. МГУ 2006, М., МЦНМО, 2007 г., стр. 135-158.

[17]. И.Б.Бурдонов, Косачев А.С. Системы с приоритетами: конформность, тестирование, композиция. Труды ИСП РАН, том 14 (1), 2008 г., стр.23-54.
[18]. И.Б. Бурдонов, А.С. Косачев. Тестирование с преобразованием семантик. Труды ИСП РАН, том 17, 2009 г., стр.193-208.
[19]. A.Kossachev, I.Burdonov. Formal Conformance Verifcation, Short Papers of the 22nd IFIP ICTSS, Alexandre Petrenko, Adenilso Simao, Jose Carlos Maldonado (eds.), Nov. 08-10, 2010, Natal, Brazil, pp.1-6.
[20]. А.С. Косачев, И.Б.Бурдонов. Семантики взаимодействия с отказами, дивергенцией и разрушением. Программирование, 2010 г., №5, стр. 3-23.
[21]. И.Б.Бурдонов, А.С.Косачев. Согласование конформности и композиции. Программирование, 2013 г., №6, стр. 3-15.

Testing of automata system

I.B. Burdonov <igor@ispras.ru>
A.S. Kossatchev <kos@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The problem of testing of aggregate systems is considered. The system is described with an oriented graph of links. The nodes correspond to automata of the components and arcs correspond to simplex communication channels. The hypothesis of the links is assumed: the graph of links is static and the link structure is error-free. In each state, the automaton can accept and send multiple messages through incoming and outgoing arcs (at most one message through each arc). The goal of testing is to cover transitions of the automata reachable during the system work. It assumed that during testing it is possible to observe the state changes of automata and the messages on the arcs. A simplified system model with only one message circulating is considered at the beginning. On its example we show that the hypothesis on links allows considerably reduce the number of required testing actions from the multiplication of numbers of the component automata states to the sum of these numbers. If the numbers of states of all automata are equal, it gives exponential reduction of the number of test actions. Then the more general model is considered when the system can simultaneously contain multiple messages, but not more than one on each arc. A composition of the system automata is defined and the restrictions on automata making the system deterministic are described. An algorithm of test generation is proposed basing on test filtration generated for covering all transitions of the deterministic composition system. Test is rejected if it covers only such transitions of the components that are covered by the remaining tests. In conclusion, the directions of future research are described.

Keywords: directed graphs; graph coverage, communicating automata, distributed systems, testing, networks.

DOI: 10.15514/ISPRAS-2016-28(1)-7

For citation: Burdonov I.B., Kossatchev A.S. Testing of automata system. Trudy ISP RAN /Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 103-130 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-7

References

[1]. Revised Working Draft on “Framework: Formal Methods in Conformance Testing”. JTC1/SC21/WG1/Project 54/1, ISO Interim Meeting, ITU-T on, Paris, 1995.

[2]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Teoriya sootvetstviya dlya system s blokirovkami I razrusheniem [Conformance theory of the systems with Refused Inputs and Forbidden Actions]. Moscow, «Nauka», 2008, 412 p. (in Russian)

[3]. Bourdonov I. Teoriya konformnosti (funkcional'noe testirovanie prorammny'kh system na osnove formal'ny'kh modelej [Conformance theory (functional testing on formal model base)]. LAP LAMBERT Academic Publishing, Saarbrucken, Germany, 2011, ISBN 978-3-8454-1747-9, 428 p. (in Russian)

[4]. Bourdonov I.B., Kossatchev A.S. Specification Completion for IOCO. Programming and Computer Software, vol. 37(1), 2011, pp. 1-14. *DOI*: 10.1134/S0361768811010014

[5]. A. S. Kamkin, M. M. Chupilko. Survey of modern technologies of simulation-based verification of hardware. Programming and Computer Software, vol. 37 (3), 2011, pp. 147-152. *DOI*: 10.1134/S0361768811030017

[6]. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Deterministic Case. Programming and Computer Software, vol. 29(5), 2003, pp. 245-258. *DOI*: 10.1023/A:1025733107700

[7]. A. Levitin. Algoritmy: vvedenie v razrabotku i analiz [Introduction to The Design and Analysis of Algorithms]. M.: «Viliams», 2006, pp. 160-163, ISBN 0-201-74395-7. (in Russian)

[8]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms. 2-nd Edition. MIT Press Cambridge, MA, USA, 2001, ISBN 0-262-03293-7.

[9]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Application of Finite Automatons for Program Testing. Programming and Computer Software, vol. 26 (2), 2000, pp. 61-73. *DOI*: 10.1007/BF02759192

[10]. Bourdonov I.B., Kossatchev A.S. Complete Open-State Testing of Limitedly Nondeterministic Systems. Programming and Computer Software, vol. 35 (6), 2009, pp.301-313. *DOI*: 10.1109/HASE.2014.39

[11]. Bourdonov I.B., Kossatchev A.S. Semantiki vzaimodejstviya s otkazami, divergentsiej i razrusheniem. Chast' 2. Usloviya konechnogo polnogo testirovaniya. [Semantics of Interaction with Refused Inputs, Divergence and Forbidden Actions. Part 2. The condition of finite complete testing]. Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika. [Tomsk State University. Journal of Control and Computer Science], 2011, №2, pp. 89-98. (in Russian)

[12]. Bourdonov I.B., Kossatchev A.S. Testirovanie konformnosti na osnove sootvetstviya sostoyanij [Conformance testing based on a state relation]. Trudy ISP RAN [Proceeding of ISP RAS], vol. 18, 2010, pp. 183-320 (in Russian)

[13]. 84. Bourdonov I.B., Kossatchev A.S.. Safe simulation testing of systems with refusals and destructions. Automatic Control and Computer Sciences, vol. 45(7), 2011, pp. 380-389. *DOI*: 10.3103/S0146411611070042

[14]. I.B.Bourdonov, A.S.Kossatchev, V.V.Kuliamin. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proceedings of the Workshop on Model Based Testing (MBT 2004), Elsevier, 2006. *DOI*: 10.1016/j.entcs.2006.09.008

[15]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Formalization of Test Experiments. Programming and Computer Software, vol. 33(5), 2007, pp. 239-260. 10.1134/S0361768807050015

[16]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Bezopasnost', verifikatsiya i teoriya konformnosti [Safety, Verification and Conformance Theory]. Materialy Vtoroj mezhdunarodnoj nauchnoj konferentsii po problemam bezopasnosti i protivodejstviya terrorizmu [The proceeding of the Second international conference on the problems of safety and counteraction against terrorizm], Moscow, MNCMO, 2007, pp. 135-158. (in Russian)

[17]. Bourdonov I.B., Kossatchev A.S. Sistemy s prioritetami: konformnost', testirovanie, kompozitsiya [Systems with priority: conformance, testing, composition]. Trudy ISP RAN [Proceeding of ISP RAS], vol. 14(1), 2008, pp.23-54 (in Russian)

[18]. Bourdonov I.B., Kossatchev A.S. Testirovanie s preobrazovaniem semantic [Testing with Semantics Conversion] Trudy ISP RAN [Proceeding of ISP RAS], vol. 17, 2009, pp. 193-208. (in Russian)

[19]. A.Kossachev, I.Burdonov. Formal Conformance Verifcation, Short Papers of the 22nd IFIP ICTSS, Alexandre Petrenko, Adenilso Simao, Jose Carlos Maldonado (eds.), Nov. 08-10, 2010, Natal, Brazil, pp.1-6.

[20]. Bourdonov I.B., Kossatchev A.S. Interaction Semantics with Refusals, Divergence, and Destruction. Programming and Computer Software, vol. 36(5), 2010, pp. 247-263. *DOI*: 10.1134/S0361768810050014

[21]. Bourdonov I.B., Kossatchev A.S. Agreement between Conformance and Composition. Programming and Computer Software, vol. 39(6), 2013, pp. 269–278. *DOI*: 10.1134/S0361768813060029

Система автоматов: композиция по графу связей

И.Б. Бурдонов <igor@ispras.ru>
А.С. Косачев <kos@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. Статья посвящена проблеме моделирования и композиции составных систем. Компоненты системы моделируются конечными автоматами с несколькими входами и выходами, а взаимодействие между ними – обменом сообщениями по симплексным каналам связи. Система описывается ориентированным графом связей, вершина которого соответствует автомату компонента, а дуга – каналу связи, соединяющему выход одного автомата со входом другого автомата. Автомат в вершине графа в каждом состоянии может принимать несколько сообщений по своим входам (не более одного по каждому входу) и посылать несколько сообщений по своим выходам (не более одного по каждому выходу). Входы (выходы) автоматов, которые не соединяются с выходами (входами) автоматов, являются внешними, через них осуществляется связь системы с её окружением. Автоматы системы работают синхронно: на каждом такте каждый автомат выполняет один переход. Переход автомата, предъявляя требования к состоянию всех входов и выходов автомата (указываются сообщения на них), отдельно указывает ту часть входов и выходов, по которым сообщения соответственно принимаются и посылаются, соответственно. Синхронность взаимодействия автоматов означает, что для каждого соединения требования автоматов, связанных этим соединением, должны быть согласованы. Это даёт возможность описывать более широкий спектр поведений автомата. Например, приоритетный приём сообщений: если на входах автомата имеется несколько сообщений, автомат может принять сообщения с наивысшим приоритетом, не принимая остальные сообщения. Также это позволяет автомату выполнять приём сообщений независимо от того, удаётся или не удаётся одновременно послать некоторое сообщение на некоторый выход. Определяется композиция автоматов системы по графу связей и доказывается её ассоциативность. В заключение определяются направления дальнейших исследований.

Ключевые слова: ориентированные графы, взаимодействующие автоматы, композиция автоматов, распределенные системы, сети.

DOI: 10.15514/ISPRAS-2016-28(1)-8

Для цитирования: Бурдонов И.Б., Косачев А.С. Система автоматов: композиция по графу связей. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 131-150. DOI: 10.15514/ISPRAS-2016-28(1)-8

1. Введение

Большинство сложных, особенно распределённых, систем представляет собой набор взаимодействующих компонентов. В данной статье, как и в нашей предыдущей статье [1], компоненты моделируются конечными автоматами, а взаимодействие – обменом сообщениями между автоматами. Мы будем предполагать, что автомат имеет несколько входов для приёма сообщений и может принимать сразу несколько сообщений; также автомат имеет несколько выходов для посылки сообщений и может посылать сразу несколько сообщений. Структура связей между компонентами моделируется ориентированным графом (будем называть его *графом связей*), вершинам которого соответствуют автоматы, а дуги называются *соединениями* и соответствуют симплексным каналам передачи сообщений. Соединение, ведущее из автомата *A* в автомат *B*, помечается выходом *j* автомата *A* и входом *i* автомата *B*. При этом каждый вход (выход) каждого автомата соединён не более чем с одним выходом (входом) другого (или того же самого) автомата. Входы (выходы) автоматов, которые не соединяются с выходами (входами) автоматов, являются *внешними*, через них осуществляется связь системы с её *окружением*. Набор сообщений на внешних входах системы можно назвать внешним *стимулом*, а набор сообщений на внешних выходах системы можно назвать внешней *реакцией*.

В [1] сообщения между автоматами, расположенными в вершинах графа связей, буферизовались соединяющими автоматы дугами графа связей. Каждая дуга представляла собой очередь длины 1 и моделировалась специальным автоматом с одним входом и одним выходом, который взаимодействовал с автоматами вершин, инцидентных этой дуге, уже синхронно. В данной статье надобность в таких буферизующих дугах и моделирующих их автоматах специального вида отпадает, или, если угодно, выполняется такое обобщение автомата дуги, которое стирает различие между автоматом дуги и автоматом вершины.

Мы будем считать, что граф связей статический, то есть не меняющийся в процессе работы системы. В этом случае система (также как её компоненты) может моделироваться конечным автоматом, получающимся из автоматов-компонентов с помощью подходящего оператора композиции, учитывающего граф связей.

В предлагаемой модели автоматы системы работают синхронно: на каждом такте каждый автомат выполняет один переход. Также на каждом такте окружение системы посылает сообщения на некоторые внешние входы системы и принимает сообщения с некоторых внешних выходов системы. Для получения замкнутой формальной системы окружение моделируется автоматом, выходы которого соединяются с внешними входами системы, а внешние выходы системы соединяются со входами окружения.

Сначала, в разделе 2, формально определяется модель автомата и модель системы. Основная особенность предлагаемой модели автомата заключается в

том, что переход автомата, предъявляя требования к состоянию всех входов и выходов автомата (указываются сообщения на них), отдельно указывает ту часть входов и выходов, по которым сообщения соответственно принимаются и посылаются. Синхронность взаимодействия автоматов означает, что для каждого соединения требования автоматов, связанных этим соединением, должны быть согласованы.

Это даёт возможность описывать более широкий спектр поведений автомата. Например, приоритетный приём сообщений: если на входах автомата имеется несколько сообщений, автомат может принять сообщения с наивысшим приоритетом, не принимая остальные сообщения. Также это позволяет автомату выполнять приём сообщений детерминированно, т.е. независимо от того, удаётся или не удаётся одновременно послать некоторое сообщение на некоторый выход. Если удаётся послать сообщение, автомат выполняет один переход, а если не удаётся, то другой. Обычно это приводит к недетерминизму: на один стимул выдаются разные реакции. Однако в нашей модели реакция – это требования к выходам, а не указание той части выходов, по которым сообщения реально передаются.

В разделе 3 формально определяется композиция переходов, композиция автоматов и композиция системы. Такая композиция выполняется по одному соединению. Доказывается, что композиция автоматов удовлетворяет требованиям, предъявляемым к автомату, а композиция системы удовлетворяет требованиям, предъявляемым к системе автоматов. Композиция системы по всем её соединениям представляет собой автомат или набор автоматов, не связанных между собой соединениями.

В разделе 4 доказывается ассоциативность композиции переходов, автоматов и системы. Ассоциативность важна для того, чтобы работа системы зависела только от множества её автоматов и соединений и не зависела от какого-либо упорядочивания этих множеств.

В разделе 5 определяется дополнительная композиция автоматов, не связанных соединениями, что позволяет докомпоновать систему ровно до одного автомата. Такой композиционный автомат, эквивалентный исходной системе, может уже использоваться как компонент более сложных систем автоматов.

В заключение намечаются направления дальнейших исследований.

2. Модель

Пусть задано некоторое конечное *множество сообщений* M, которое мы будем называть *алфавитом* (сообщений). К алфавиту сообщений добавим выделенное *пустое сообщение* $\Lambda \notin M$, и обозначим $M_\Lambda \triangleq M \cup \{\Lambda\}$. Пустое сообщение моделирует отсутствие сообщения на входе или выходе автомата.

Автомат в алфавите M – это набор $A=(M,I,J,S,T,s_0)$, где

I – конечное множество *входов* автомата,

J – конечное множество *выходов* автомата,

S – конечное множество *состояний* автомата,

$T \subseteq S \times X \times P \times Y \times Q \times S$ – множество *переходов* автомата, где

$X = \{ x \mid x: I \to M_\Lambda \}$ – множество *стимулов* (если $I=\varnothing$, то $X=\{\varnothing\}$),

$Y = \{ y \mid y: J \to M_\Lambda \}$ – множество *реакций* (если $J=\varnothing$, то $Y=\{\varnothing\}$),

$P = 2^I$ – семейство подмножеств входов (по ним принимаются сообщения),

$Q = 2^J$ – семейство подмножеств выходов (по ним посылаются сообщения),

$s_0 \in S$ – *начальное состояние*,

причём выполнены следующие условия:

1. входы и выходы не пересекаются: $I \cap J = \varnothing$,
2. приниматься по входу и передаваться по выходу может только непустое сообщение: $\forall (s, x, p, y, q, t) \in T\ (\forall i \in p\ x(i) \neq \Lambda\ \&\ \forall j \in q\ y(j) \neq \Lambda)$, что эквивалентно $\forall (s, x, p, y, q, t) \in T\ (p \subseteq x^{-1}(M)\ \&\ q \subseteq y^{-1}(M))$.

Очевидно, что из конечности множеств M, I, J и S следует конечность множеств X, Y, P, Q, T.

Для перехода $a = (s, x, p, y, q, t)$ состояние s будем называть *пресостоянием* перехода, а состояние t – *постсостоянием*, и будем обозначать $s_a=s$, $x_a=x$, $p_a=p$, $y_a=y$, $q_a=q$, $t_a=t$. Для автомата $A=(M,I,J,S,T,s_0)$ будем обозначать $I_A=I$, $J_A=J$, $S_A=S$, $T_A=T$, $s_{0A}=s_0$.

Работу автомата можно описать следующим образом. В текущем состоянии s проверяется состояние входов, т.е. какие сообщения можно принять со входов. Этим определяется стимул x. Далее рассматриваются переходы по этому стимулу, т.е. множество $T_{s,x} \subseteq T$ переходов из s по x. Каждый переход из множества $T_{s,x}$ определяет ту или иную реакцию y. Если для данных s и x имеются переходы с разными реакциями y, то недетерминированным образом выбирается реакция y как одна из реакций на переходах из множества $T_{s,x}$. После выбора реакции y проверяется состояние выходов, а именно, какие непустые сообщения, определяемые реакцией y, могут быть переданы по соответствующим выходам, а какие нет. Это определяет параметр q. Тем самым, определяется множество $T_{s,x,y,q} \subseteq T_{s,x}$ переходов с данными s, x, y, q. Если это множество содержит более одного перехода, то недетерминированным образом выбирается один из них. Если выбран переход (s, x, p, y, q, t), то параметр p определяет, какие непустые сообщения будут приняты со входов автомата. Итак, принимаются сообщения, определяемые стимулом x, по входам, определяемым параметром p, посылаются сообщения, определяемые реакцией y, по выходам, определяемым параметром q, после чего автомат переходит в состояние t. Это описание работы автомата неформально. Формальное описание есть, по существу, формальное определение композиции автоматов, которое будет дано в разделе 3.

Пусть V – конечное множество автоматов в алфавите M. Без ограничения общности можно считать, что входы, выходы и состояния автоматов разные:

$\forall A,B \in V\ (A \neq B \Rightarrow I_A \cap I_B = \varnothing\ \&\ I_A \cap J_B = \varnothing\ \&\ J_A \cap J_B = \varnothing\ \&\ S_A \cap S_B = \varnothing)$. Этого всегда можно добиться с помощью систематического переименования состояний, входов и выходов автоматов, в результате которого получаются автоматы изоморфные исходным.

Система автоматов – это набор $\boldsymbol{R} = (M,V,E)$, где $E : E_{Dom} \to E_{Im}$ – биекция, определяющая *соединения*, где $E_{Dom} \subseteq J_{\boldsymbol{R}}$, $E_{Im} \subseteq I_{\boldsymbol{R}}$, $J_{\boldsymbol{R}} = \cup\{J_A | A \in V\}$ – множество всех выходов всех автоматов, $I_{\boldsymbol{R}} = \cup\{I_A | A \in V\}$ – множество всех входов всех автоматов. Для системы $\boldsymbol{R} = (M,V,E)$ обозначим: $V_{\boldsymbol{R}}=V$, $E_{\boldsymbol{R}}=E$.

Определим функцию $\varphi:(I_{\boldsymbol{R}} \cup J_{\boldsymbol{R}}) \to V$, которая каждому входу или выходу системы $\boldsymbol{R}$ ставит в соответствие автомат, которому этот вход или выход принадлежит, т.е. $\forall A \in V\ \ \forall z \in I_A \cup J_A\ \ \varphi(z)=A$. Будем говорить, что соединение (j,i) ведёт из автомата $\varphi(j)$ в автомат $\varphi(i)$. Функция φ зависит от системы автоматов, однако для сокращения записи везде, где подразумевается система $\boldsymbol{R}$, мы её не указываем в параметрах функции φ. Если функция φ применяется для другой системы $\boldsymbol{R}` \neq \boldsymbol{R}$, мы будем писать $\varphi_{\boldsymbol{R}`}$.

Вход $i \in I_{\boldsymbol{R}}$, для которого не определено соединение, т.е. $i \notin E_{Im}$, будем называть *внешним входом системы*, и выход $j \in J_{\boldsymbol{R}}$, для которого не определено соединение, т.е. $j \notin E_{Dom}$, будем называть *внешним выходом системы*.

Системе $\boldsymbol{R} = (M,V,E)$ соответствует ориентированный граф с помеченными дугами, вершинами которого являются автоматы из V, а дуга $A \to B$ существует и помечена соединением (j,i) тогда и только тогда, когда $(j,i) \in E$, $\varphi(j)=A$ и $\varphi(i)=B$. Такой граф обозначим через $\boldsymbol{Og(R)}$. Если снять ориентацию дуг, то соответствующий неориентированный граф обозначим через $\boldsymbol{Ng(R)}$. Будем говорить, что автоматы A и B *связаны* в системе, если в графе $\boldsymbol{Ng(R)}$ существует (неориентированный) путь из A в B (возможно, пустой, если $A=B$). В противном случае разные автоматы A и B *не связаны*. Очевидно, что отношение связанности автоматов рефлексивно, симметрично и транзитивно, т.е. является отношением эквивалентности, которое разбивает множество автоматов V на классы эквивалентности, которые будем называть классами связанности.

Соединение $(j,i) \in E$ будем называть *петлёй*, если это дуга-петля в графе $\boldsymbol{Og(R)}$, то есть $\varphi(j)=\varphi(i)$.

Для удобства будем считать, что каждое состояние каждого автомата в системе $\boldsymbol{R}=(M,V,E)$ является множеством, причём состояния разных автоматов в системе являются непересекающимися множествами: $\forall A,B \in V\ \ \forall s_A \in S_A\ \ \forall s_B \in S_B\ (A \neq B \Rightarrow s_A \cap s_B = \varnothing)$. Если уже задана система, в которой это условие не выполнено, то можно преобразовать каждый автомат, заменив в нём каждое состояние s на синглетон $\{s\}$. Поскольку в системе автоматов состояния разных автоматов разные, то таким преобразованием будут получены автоматы изоморфные исходным, а для системы автоматов будет выполнено условие попарного непересечения множеств состояний автоматов.

Работу системы автоматов можно описать следующим образом. Все автоматы срабатывают одновременно, и каждый автомат выполняет не более одного перехода. При этом считается, что «внешнее окружение» автомата определяет, какое сообщение оказывается на каждом внешнем входе, включая пустое сообщение (т.е. отсутствие сообщения на входе). Также окружение определяет для каждого внешнего выхода множество «разрешённых» сообщений, включая пустое сообщение (т.е. отсутствие сообщения на выходе), где «разрешённое» сообщение – это сообщение, которое может посылаться системой по этому внешнему выходу, и для непустых сообщений окружение определяет, будет ли это сообщение принято окружением или нет. Далее, если автоматы A и B связаны таким соединением (j,i), что $j \in J_A$ и $i \in I_B$, то для выполняемых переходов $a \in T_A$ и $b \in T_B$ сообщение $y_a(j)$ и условие его передачи $j \in q_a$ на выходе j автомата A должны быть согласованы с сообщением $x_b(i)$ и условием его приёма $i \in p_b$ на входе i автомата B. Это означает, во-первых, что сообщение $y_a(j)$ на выходе j совпадает с сообщением $x_b(i)$ на входе i. И, во-вторых, сообщение либо передаётся из A по выходу j, т.е. $j \in q_a$, и принимается в B по входу i, т.е. $i \in p_b$, либо не передаётся из A по выходу j, т.е. $j \notin q_a$, и не принимается в B по входу i, т.е. $i \notin p_b$. Работа с внешними входами и выходами определяется аналогично, когда окружение моделируется тем или иным автоматом.

Это описание работы системы автоматов неформально. Формальное описание есть, по существу, формальное определение композиции системы автоматов по всем её соединениям, которое будет дано в следующем разделе 3.

3. Композиция

Существует большое разнообразие как автоматоподобных моделей, так и операторов их (параллельной) композиции, начиная с классической композиции в алгебрах процессов CSP – Communicating Sequential Processes [2] и CCS – Calculus of Communicating Systems [3]. Они определяют две разновидности композиции LTS (Labelled Transition System), в которых переходы помечены символами *действий* из заданного алфавита. Для систем ввода-вывода (IOTS – Input-Output Transition System), в которых действия разбиваются на стимулы (input) и реакции (output) предлагались различные модификации композиции [5][6][7], в частности, для обнаружения deadlock'а с целью наблюдения *отказов* (refusal) [4][5][6]. Для систем с приоритетами и моделей *событий* (систем более общего вида) нами были также предложены соответствующие операторы композиции [8][9]. Вводимая в данном разделе композиция учитывает наличие у автомата нескольких входов и выходов и моделирует взаимодействие автоматов через соединения, каждое из которых связывает выход одного автомата со входом другого автомата. Иными словами, это композиция автоматов по графу связей системы автоматов.

Введём способ сокращённой записи для преобразования функции с помощью уменьшения её домена: для произвольной функции f и произвольного множества N обозначим $f/N \triangleq f \setminus \{(z,f(z)) \mid z \in N \cap \mathbf{Dom}(f)\}$. Очевидно, что для произвольного множества $N`$ имеет место $f/(N \cup N`) = (f/N)/N`$ и для произвольной функции g такой, что $\mathbf{Dom}(f) \cap \mathbf{Dom}(g) = \varnothing$ и $N \cap \mathbf{Dom}(g) = \varnothing$, имеет место $(f \cup g)/N = (f/N) \cup g$. Будем считать, что операция "/" имеет тот же приоритет, что разность "\" и объединение "∪" множеств. Как следствие, в выражениях над множествами, где используются только операции "/", "\" и "∪", мы будем использовать бесскобочную запись и предполагать, что операции выполняются слева направо.

Логическую эквивалентность будем обозначать символом «~»: $a \sim b \triangleq (a \& b) \vee (\neg a \& \neg b)$.

Композиция определяется по соединению $(j,i) \in E$. Обозначим через A автомат, которому принадлежит выход j, т.е. $A = \varphi(j)$, а через B автомат, которому принадлежит вход i, т.е. $B = \varphi(i)$.

Композиция переходов по соединению. Для соединения (j,i) определим условие $f(a,j,i,b)$ композиции двух переходов a и b и результат композиции $a[j,i]b$:

$f(a,j,i,b) \triangleq a \in T_A \,\&\, b \in T_B \,\&\, (A = B \Rightarrow a = b) \,\&\, y_a(j) = x_b(i) \,\&\, (j \in q_a \sim i \in p_b)$,

$f(a,j,i,b)$: $a[j,i]b \triangleq (s_a \cup s_b,\ x_a \cup x_b/\{i\},\ p_a \cup p_b \setminus \{i\},\ y_a \cup y_b/\{j\},\ q_a \cup q_b \setminus \{j\},\ t_a \cup t_b)$.

Заметим, что если $A = B$, то

$f(a,j,i,a) \triangleq a \in T_A \,\&\, y_a(j) = x_a(i) \,\&\, (j \in q_a \sim i \in p_a)$,

$f(a,j,i,a)$: $a[j,i]a \triangleq (s_a,\ x_a/\{i\},\ p_a \setminus \{i\},\ y_a/\{j\},\ q_a \setminus \{j\},\ t_a)$.

Композиция переходов естественным образом распространяется на композицию множеств переходов, определяемую как множество попарных композиций переходов. Для соединения (j,i) определим композицию множеств переходов G и H: $G[j,i]H \triangleq \{a[j,i]b \mid a \in G \,\&\, b \in H \,\&\, f(a,j,i,b)\}$.

Композиция автоматов по соединению.

Мы определим композицию автоматов, в которой оставим только начальное состояние композиции и пре- и постсостояния переходов композиции. Это множество, очевидно, включает подмножество состояний, достижимых из начального состояния. Для множества переходов T обозначим множество пре- и постсостояний этих переходов: $States(T) \triangleq \{s_a \mid a \in T\} \cup \{t_a \mid a \in T\}$.

Для соединения (j,i) определим композицию автоматов A и B

$A[j,i]B \triangleq (M,\ I_A \cup I_B \setminus \{i\},\ J_A \cup J_B \setminus \{j\},\ \{s_{0A} \cup s_{0B}\} \cup States(T_A[j,i]T_B),\ T_A[j,i]T_B,\ s_{0A} \cup s_{0B})$.

Теорема 1 о композиции автоматов по соединению:

Композиция автоматов удовлетворяет условиям, налагаемым на автомат.

Доказательство:

1. Алфавит сообщений не меняется и потому остаётся конечным. Условия конечности множеств входов, выходов и состояний композиционного автомата очевидным образом следуют из определения композиции и конечности множеств входов, выходов и переходов операндов.
2. Пусть $f(a,j,i,b)=\mathbf{true}$. Для перехода $a[j,i]b \in T_{A[j,i]B}$ надо показать следующее:

2.1. $s_{a[j,i]b} \in S_{A[j,i]B}$.
$a[j,i]b \in T_{A[j,i]B}$ влечёт $s_{a[j,i]b} \in States(T_A[j,i]T_B)$, что влечёт $s_{a[j,i]b} \in S_{A[j,i]B}$.

2.2. $t_{a[j,i]b} \in S_{A[j,i]B}$. Доказывается аналогично 2.1.

2.3. $x_{a[j,i]b}$ есть отображение $I_{A[j,i]B} \to M_\Lambda$.
Из того, что $I_A \cap I_B = \varnothing$, x_a является отображением $I_A \to M_\Lambda$, x_b является отображением $I_B \to M_\Lambda$, следует, что $x_a \cup x_b/\{i\}$ является отображением $(I_A \cup I_B \backslash \{i\}) \to M_\Lambda$. Следовательно, поскольку $I_{A[j,i]B}=I_A \cup I_B \backslash \{i\}$, а $x_{a[j,i]b}=x_a \cup x_b/\{i\}$, доказываемое утверждение верно.

2.4. $y_{a[j,i]b}$ есть отображение $J_{A[j,i]B} \to M_\Lambda$. Доказывается аналогично 2.3.

2.5. $p_{a[j,i]b} \subseteq x_{a[j,i]b}^{-1}(M)$ (условие 2 для p).
$x_{a[j,i]b} = x_a \cup x_b/\{i\}$, $p_{a[j,i]b} = p_a \cup p_b \backslash \{i\}$.
Поскольку $p_b \subseteq x_b^{-1}(M)$, имеем $(p_b \backslash \{i\}) \subseteq (x_b/\{i\})^{-1}(M)$.
Поскольку $\mathbf{Dom}(x_a)=I_A$, $\mathbf{Dom}(x_b)=I_B$, а $I_A \cap I_B=\varnothing$, имеем $x_a \cap x_b = \varnothing$.
Также $p_a \subseteq x_a^{-1}(M)$.
Тем самым, $p_{a[j,i]b}=(p_a \cup p_b \backslash \{i\}) \subseteq (x_a \cup x_b/\{i\})^{-1}(M) = x_{a[j,i]b}^{-1}(M)$.

2.6. $q_{a[j,i]b} \subseteq y_{a[j,i]b}^{-1}(M)$ (условие 2 для q). Доказывается аналогично 2.5.

3. Покажем, что $s_{0A[j,i]B0} \in S_{A[j,i]B}$.
Поскольку $s_{0A[j,i]B} = s_{0A} \cup s_{0B}$, а $s_{0A} \cup s_{0B} \in S_{A[j,i]B}$, имеем $s_{0A[j,i]B} \in S_{A[j,i]B}$.
4. Покажем, что $I_{A[j,i]B} \cap J_{A[j,i]B} = \varnothing$ (условие 1).
Действительно, $I_{A[j,i]B} \cap J_{A[j,i]B} = (I_A \cup I_B \backslash \{i\}) \cap (J_A \cup J_B \backslash \{j\})$
$= (I_A \cup (I_B \backslash \{i\})) \cap ((J_A \backslash \{j\}) \cup J_B)$
$= I_A \cap (J_A \backslash \{j\}) \cup I_A \cap J_B \cup (I_B \backslash \{i\}) \cap (J_A \backslash \{j\}) \cup (I_B \backslash \{i\}) \cap J_B$
$= \varnothing \cup \varnothing \cup \varnothing \cup \varnothing = \varnothing$,
поскольку $I_A \cap J_A = \varnothing$, $I_A \cap J_B = \varnothing$, $I_B \cap J_A = \varnothing$, $I_B \cap J_B = \varnothing$.
5. Дополнительное условие на состояния: каждое из них является множеством.
Состояния автоматов-операндов это множества. Поэтому состояние композиционного автомата тоже множество по определению композиции.

Теорема 1 доказана.

Композиция системы по соединению. Для системы $\boldsymbol{R}$ и соединения (j,i) определим $\boldsymbol{R}[j,i] \triangleq (M, V\setminus\{A,B\}\cup\{A[j,i]B\}, E\setminus\{(j,i)\})$.

Теорема 2 о композиции системы по соединению:

Композиция системы удовлетворяет условиям, налагаемым на систему.

Доказательство:

1. Алфавит сообщений не меняется и потому остаётся конечным. Множество автоматов $V_{\boldsymbol{R}[j,i]} = V_{\boldsymbol{R}}\setminus\{A,B\}\cup\{A[j,i]B\}$ конечно, поскольку конечно множество $V_{\boldsymbol{R}}$.
2. Покажем, что, если система $\boldsymbol{R}=(M,V,E)$ удовлетворяла требованиям непересечения входов, выходов и состояний автоматов, то для соединения $(j,i)\in E$ этим же требованиям будет удовлетворять композиционная система $\boldsymbol{R}[j,i]$. Нам достаточно показать, что входы, выходы и состояния автомата $A[j,i]B$ не пересекаются со входами, выходами и состояниями других автоматов композиционной системы $\boldsymbol{R}[j,i]$.

2.1. Поскольку композиционный автомат $A[j,i]B$ наследует входы и выходы из автоматов-операндов, а сами автоматы-операнды удаляются из системы, требование непересечения входов и выходов автоматов сохраняется. Формально: $I_{A[j,i]B}=I_A\cup I_B\setminus\{i\}$, $J_{A[j,i]B}=J_A\cup J_B\setminus\{j\}$. Поэтому, если в исходной системе $\boldsymbol{R}=(M,V,E)$ входы и выходы не пересекались, т.е. $\forall C,D\in V\ (C\neq D \Rightarrow I_C\cap I_D=\varnothing \ \&\ I_C\cap J_D=\varnothing \ \&\ J_C\cap J_D=\varnothing)$, то входы и выходы композиционного автомата $A[j,i]B$ не пересекаются со входами и выходами других автоматов композиционной системы $\boldsymbol{R}[j,i]$: $(I_A\cup I_B\setminus\{i\})\cap I_D=\varnothing \ \&\ (I_A\cup I_B\setminus\{i\})\cap J_D=\varnothing \ \&\ (J_A\cup J_B\setminus\{j\})\cap J_D=\varnothing$, так как $D\neq A$ и $D\neq B$.

2.2. Каждое состояние автомата $A[j,i]B$ – это объединение состояний автоматов-операндов $s_A\cup s_B$. Поскольку автоматы-операнды удаляются из системы и состояния автоматов исходной системы $\boldsymbol{R}$ не пересекаются, то состояния композиционного автомата также не пересекаются с состояниями других автоматов композиционной системы $\boldsymbol{R}[j,i]$. Формально: поскольку $\forall C,D\in V\ (C\neq D \Rightarrow \forall s_C\in S_C\ \forall s_D\in S_D\ s_C\cap s_D=\varnothing)$, имеет место $(s_A\cup s_B)\cap s_D=\varnothing$, так как $D\neq A$ и $D\neq B$.

3. Покажем, что $E\setminus\{(j,i)\}$ является биекцией подмножества всех выходов всех автоматов на подмножество всех входов всех автоматов. Действительно, $E : E_{Dom}\to E_{Im}$ – биекция, где $E_{Dom}\subseteq J_{\boldsymbol{R}}$ и $E_{Im}\subseteq I_{\boldsymbol{R}}$. Следовательно, $E\setminus\{(j,i)\} : E_{Dom}\setminus\{j\}\to E_{Im}\setminus\{i\}$ – биекция. Также $E_{Dom}\subseteq J_{\boldsymbol{R}}$ и $E_{Im}\subseteq I_{\boldsymbol{R}}$ влечёт $E_{Dom}\setminus\{j\}\subseteq J_{\boldsymbol{R}}\setminus\{j\}$ и $E_{Im}\setminus\{i\}\subseteq I_{\boldsymbol{R}}\setminus\{i\}$. Поскольку $I_{A[j,i]B}=I_A\cup I_B\setminus\{i\}$ и $J_{A[j,i]B}=J_A\cup J_B\setminus\{i\}$, а входы и выходы остальных автоматов из V не меняются, $J_{\boldsymbol{R}}\setminus\{j\} = J_{\boldsymbol{R}[j,i]}$ и $I_{\boldsymbol{R}}\setminus\{i\} = I_{\boldsymbol{R}[j,i]}$. Обозначая $E_{\boldsymbol{R}[j,i]Dom} = E_{Dom}\setminus\{j\}$ и $E_{\boldsymbol{R}[j,i]Im} = E_{Im}\setminus\{j\}$, имеем $E_{\boldsymbol{R}[j,i]} = E\setminus\{(j,i)\} : E_{\boldsymbol{R}[j,i]Dom}\to E_{\boldsymbol{R}[j,i]Im}$ – биекция, где $E_{\boldsymbol{R}[j,i]Dom} = E_{Dom}\setminus\{j\}\subseteq J_{\boldsymbol{R}}\setminus\{j\} = J_{\boldsymbol{R}[j,i]}$ и $E_{\boldsymbol{R}[j,i]Im} = E_{Im}\setminus\{i\}\subseteq I_{\boldsymbol{R}}\setminus\{i\} = I_{\boldsymbol{R}[j,i]}$.

Теорема 2 доказана.

4. Ассоциативность композиции

Ассоциативность композиции важна для того, чтобы работа системы зависела только от множества её автоматов и соединений и не зависела от какого-либо упорядочивания этих множеств.

Последовательностью соединений для системы $\boldsymbol{R} = (M,V,E)$ назовём последовательность Z соединений из E, в которой каждое соединение встречается не более одного раза, т.е. $Z:[1..|Z|]\to E$ является инъекцией. Определим композицию системы $\boldsymbol{R}$ по последовательности Z как $\boldsymbol{R}[Z] \triangleq \boldsymbol{R}[Z(1)][Z(2)]...[Z(|Z|)]$, если Z не пусто, и $\boldsymbol{R}[Z] \triangleq \boldsymbol{R}$, если Z пусто.

Последовательность соединений, являющуюся биекцией, т.е. содержащую каждое соединение из E, будем называть *последовательностью всех соединений* и обозначать $Z^\wedge$. Очевидно, что в системе $\boldsymbol{R}[Z^\wedge]$ нет соединений, т.е. $E_{\boldsymbol{R}[Z^\wedge]}=\varnothing$. Автоматы такой системы не связаны друг с другом, а все входы и выходы автоматов являются внешними.

Сначала мы докажем три леммы об ассоциативности композиции. Рассмотрим два различных соединения $(j,i)\in E$ и $(l,k)\in E$ и четыре автомата $A=\varphi(j)$, $B=\varphi(i)$, $C=\varphi(l)$, $D=\varphi(k)$.

Лемма 1 об ассоциативности композиции переходов:

Пусть $a\in T_A$, $b\in T_B$, $c\in T_C$, $d\in T_D$. Переходы, которые получаются из переходов a, b, c, d при композиции по соединениям (j,i) и (l,k), а также условия существования этих переходов не зависят от того, в каком порядке производится композиция.

Доказательство. Доказательство будем вести в зависимости от «топологии» соединений (j,i) и (l,k).

1. $A\to B$ $C\to D$. 4 разных автомата, независимые соединения: $A\neq B$, $A\neq C$, $A\neq D$, $B\neq C$, $B\neq D$, $C\neq D$. Поскольку участвующие в этих двух соединениях пары автоматов $\{A,B\}$ и $\{C,D\}$ не пересекаются, также не пересекаются их множества переходов. Поэтому утверждение очевидно: при любом порядке композиции получается множество переходов $\{a[j,i]b,c[l,k]d\}$.

В остальных случаях результатом композиции по двум соединениям будет не два перехода, а один переход. Пусть g –переход, являющийся результатом композиции в порядке (j,i), (l,k), а f_g – условие существования этого перехода. Также пусть h – переход, являющийся результатом композиции в порядке (l,k), (j,i), а f_h – условие существования этого перехода. Тогда нужно доказать, что $f_g=f_h$ и $g=h$. Сначала мы докажем, что условие существования композиционного перехода, а также пресостояние и стимул не зависят от порядка композиции, т.е. $f_g=f_h$, $s_g=s_h$, $x_g=x_h$. После этого докажем аналогичное утверждение для остальных компонентов перехода: $p_g=p_h$, $y_g=y_h$, $q_g=q_h$, $t_g=t_h$. Рассмотрим все оставшиеся «топологии».

2. $A \to B \to D$. 3 разных автомата, цепочка соединений: $C=B$, $A \neq B$, $A \neq D$, $B \neq D$. Имеем: $g=(a[j,i]b)[l,k]d$, $h=a[j,i](b[l,k]d)$. Из $C=B$ следует $c=b$ и $l \in J_B$.

2.1. $f_g = f(a[j,i]b,l,k,d)$
$= a[j,i]b \in T_{A[j,i]B}$ & $d \in T_D$ & $y_{a[j,i]b}(l)=x_d(k)$ & $(l \in q_{a[j,i]b} \sim k \in p_d)$
$= f(a,j,i,b)$ & $d \in T_D$ & $(y_a \cup y_b/\{j\})(l)=x_d(k)$ & $(l \in q_a \cup q_b \setminus \{j\} \sim k \in p_d)$
$= a \in T_A$ & $b \in T_B$ & $y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$
& $d \in T_D$ & $y_b(l)=x_d(k)$ & $(l \in q_b \sim k \in p_d)$
$= b \in T_B$ & $d \in T_D$ & $y_b(l)=x_d(k)$ & $(l \in q_b \sim k \in p_d)$
& $a \in T_A$ & $y_a(j)= x_b(i)$ & $(j \in q_a \sim i \in p_b)$
$= f(b,l,k,d)$ & $a \in T_A$ & $y_a(j)=(x_b \cup x_d/\{k\})(i)$ & $(j \in q_a \sim i \in p_b \cup p_d \setminus \{k\})$
$= b[l,k]d \in T_{B[l,k]D}$ & $a \in T_A$ & $y_a(j)=x_{b[l,k]d}(i)$ & $(j \in q_a \sim i \in p_{b[l,k]d})$
$= f(a,j,i,b[l,k]d)= f_h$.

2.2. $s_g = s_{a[j,i]b} \cup s_d = (s_a \cup s_b) \cup s_d = s_a \cup (s_b \cup s_d) = s_a \cup s_{b[l,k]d} = s_h$.

2.3. $x_g = x_{a[j,i]b} \cup x_d/\{k\} = (x_a \cup x_b/\{i\}) \cup x_d/\{k\} = x_a \cup x_b \cup x_d/\{i,k\}$
$= x_a \cup (x_b \cup x_d/\{k\})/\{i\} = x_a \cup x_{b[l,k]d}/\{i\} = x_h$.

3. $B \leftarrow A \to D$. 3 разных автомата, расходящиеся соединения: $C=A$, $A \neq B$, $A \neq D$, $B \neq D$. Имеем: $g=(a[j,i]b)[l,k]d$, $h=(a[l,k]d)[j,i]b$. Из $C=A$ следует $c=a$ и $l \in J_A$.

3.1. $f_g = f(a[j,i]b,l,k,d)$
$= a[j,i]b \in T_{A[j,i]B}$ & $d \in T_D$ & $y_{a[j,i]b}(l)=x_d(k)$ & $(l \in q_{a[j,i]b} \sim k \in p_d)$
$= f(a,j,i,b)$ & $d \in T_D$ & $(y_a \cup y_b/\{j\})(l)=x_d(k)$ & $(l \in q_a \cup q_b \setminus \{j\} \sim k \in p_d)$
$= a \in T_A$ & $b \in T_B$ & $y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$
& $d \in T_D$ & $y_a(l)=x_d(k)$ & $(l \in q_a \sim k \in p_d)$
$= a \in T_A$ & $d \in T_D$ & $y_a(l)=x_d(k)$ & $(l \in q_a \sim k \in p_d)$
& $b \in T_B$ & $y_a(j)= x_b(i)$ & $(j \in q_a \sim i \in p_b)$
$= f(a,l,k,d)$ & $b \in T_B$ & $(y_a \cup y_d/\{l\})(j)=x_b(i)$ & $(j \in q_a \cup q_d \setminus \{l\} \sim i \in p_b)$
$= a[l,k]d \in T_{A[l,k]D}$ & $b \in T_B$ & $y_{a[l,k]d}(j)=x_b(i)$ & $(j \in q_{a[l,k]d} \sim i \in p_b)$
$= f(a[l,k]d,j,i,b)= f_h$.

3.2. $s_g = s_{a[j,i]b} \cup s_d = (s_a \cup s_b) \cup s_d = (s_a \cup s_d) \cup s_b = s_{a[l,k]d} \cup s_b = s_h$.

3.3. $x_g = x_{a[j,i]b} \cup x_d/\{k\} = (x_a \cup x_b/\{i\}) \cup x_d/\{k\} = x_a \cup x_b \cup x_d/\{i,k\}$
$= (x_a \cup x_d/\{k\}) \cup x_b/\{i\} = x_{a[l,k]d} \cup x_b/\{i\} = x_h$.

4. $A \to B \leftarrow C$. 3 разных автомата, сходящиеся соединения: $D=B$, $A \neq B$, $A \neq C$, $B \neq C$. Имеем: $g=c[l,k](a[j,i]b)$, $h=a[j,i](c[l,k]b)$. Из $D=B$ следует $d=b$ и $k \in I_B$.

4.1. $f_g = f(c,l,k,a[j,i]b)$
$= a[j,i]b \in T_{A[j,i]B}$ & $c \in T_C$ & $y_c(l)=x_{a[j,i]b}(k)$ & $(l \in q_c \sim k \in p_{a[j,i]b})$
$= f(a,j,i,b)$ & $c \in T_C$ & $y_c(l)=(x_a \cup x_b/\{i\})(k)$ & $(l \in q_c \sim k \in p_a \cup p_b \setminus \{i\})$

$= a \in T_A$ & $b \in T_B$ & $y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$
& $c \in T_C$ & $y_c(l)=x_b(k)$ & $(l \in q_c \sim k \in p_b)$
$= c \in T_C$ & $b \in T_B$ & $y_c(l)=x_b(k)$ & $(l \in q_c \sim k \in p_b)$
& $a \in T_A$ & $y_a(j)= x_b(i)$ & $(j \in q_a \sim i \in p_b)$
$= f(c,l,k,b)$ & $a \in T_A$ & $y_a(j)=(x_c \cup x_b/\{k\})(i)$ & $(j \in q_a \sim i \in p_c \cup p_b \setminus \{k\})$
$= c[l,k]b \in T_{C[l,k]B}$ & $a \in T_A$ & $y_a(j)=x_{c[l,k]b}(i)$ & $(j \in q_a \sim i \in p_{c[l,k]b})$
$= f(a,j,i,c[l,k]b)= f_h$.

4.2. $s_g = s_c \cup s_{a[j,i]b} = s_c \cup (s_a \cup s_b) = s_a \cup (s_c \cup s_b) = s_a \cup s_{c[l,k]b} = s_h$.

4.3. $x_g = x_c \cup x_{a[j,i]b}/\{k\} = x_c \cup (x_a \cup x_b/\{i\})/\{k\} = x_c \cup x_a \cup x_b/\{i,k\}$
$= x_a \cup (x_c \cup x_b/\{k\})/\{i\} = x_a \cup x_{c[l,k]b}/\{i\} = x_h$.

5. $A \overset{\curvearrowleft}{\rightarrow} B$. 2 разных автомата, петля в начале: $C=D=A$, $A \neq B$. Имеем: $g=(a[j,i]b)[l,k](a[j,i]b)$, $h=(a[l,k]a)[j,i]b$. Из $C=D=A$ следует $c=d=a$ и $l \in J_A$, $k \in I_A$.

5.1. $f_g = f(a[j,i]b,l,k,a[j,i]b)$
$= a[j,i]b \in T_{A[j,i]B}$ & $y_{a[j,i]b}(l)=x_{a[j,i]b}(k)$ & $(l \in q_{a[j,i]b} \sim k \in p_{a[j,i]b})$
$= f(a,j,i,b)$ & $(y_a \cup y_b/\{j\})(l)=(x_a \cup x_b/\{i\})(k)$ & $(l \in q_a \cup q_b \setminus \{i\} \sim k \in p_a \cup p_b \setminus \{i\})$
$= a \in T_A$ & $b \in T_B$ & $y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$ & $y_a(l)=x_a(k)$ & $(l \in q_a \sim k \in p_a)$
$= a \in T_A$ & $y_a(l)=x_a(k)$ & $(l \in q_a \sim k \in p_a)$ & $b \in T_B$ & $y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$
$= f(a,l,k,a)$ & $b \in T_B$ & $(y_a/\{l\})(j)=x_b(i)$ & $(j \in q_a \setminus \{l\} \sim i \in p_b)$
$= a[l,k]a \in T_{A[l,k]A}$ & $b \in T_B$ & $y_{a[l,k]a}(j)=x_b(i)$ & $(j \in q_{a[l,k]a} \sim i \in p_b)$
$= f(a[l,k]a,j,i,b)= f_h$.

5.2. $s_g = s_{a[j,i]b} = s_a \cup s_b = s_{a[l,k]a} \cup s_b = s_h$.

5.3. $x_g = x_{a[j,i]b}/\{k\} = (x_a \cup x_b/\{i\})/\{k\} = x_a \cup x_b/\{i,k\}$
$= (x_a/\{k\} \cup x_b/\{i\} = x_{a[l,k]a} \cup x_b/\{i\} = x_h$.

6. $A \overset{\curvearrowright}{\rightarrow} B$. 2 разных автомата, петля в конце: $C=D=B$, $A \neq B$. Имеем: $g=(a[j,i]b)[l,k](a[j,i]b)$, $h=a[j,i](b[l,k]b)$. Из $C=D=B$ следует $c=d=b$ и $l \in J_B$, $k \in I_B$.

6.1. $f_g = f(a[j,i]b,l,k,a[j,i]b)$
$= a[j,i]b \in T_{A[j,i]B}$ & $y_{a[j,i]b}(l)=x_{a[j,i]b}(k)$ & $(l \in q_{a[j,i]b} \sim k \in p_{a[j,i]b})$
$= f(a,j,i,b)$ & $(y_a \cup y_b/\{j\})(l)=(x_a \cup x_b/\{i\})(k)$ & $(l \in q_a \cup q_b \setminus \{i\} \sim k \in p_a \cup p_b \setminus \{i\})$
$= a \in T_A$ & $b \in T_B$ & $y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$ & $y_b(l)=x_b(k)$ & $(l \in q_b \sim k \in p_b)$
$= b \in T_B$ & $y_b(l)=x_b(k)$ & $(l \in q_b \sim k \in p_b)$ & $a \in T_A$ & $y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$
$= f(b,l,k,b)$ & $a \in T_A$ & $y_a(j)=(x_b/\{k\})(i)$ & $(j \in q_a \sim i \in p_b \setminus \{k\})$
$= b[l,k]b \in T_{B[l,k]B}$ & $a \in T_A$ & $y_a(j)=x_{b[l,k]b}(i)$ & $(j \in q_a \sim i \in p_{b[l,k]b})$
$= f(a,j,i,b[l,k]b)= f_h$.

6.2. $s_g = s_{a[j,i]b} = s_a \cup s_b = s_a \cup s_{b[l,k]b} = s_h$.

6.3. $x_g = x_{a[j,i]b}/\{k\} = (x_a \cup x_b/\{i\})/\{k\} = x_a \cup x_b/\{i,k\}$
$= x_a \cup (x_b/\{k\})/\{i\} = x_a \cup x_{b[l,k]b}/\{i\} = x_h$.

7. $A \leftrightarrows B$. 2 разных автомата, цикл соединений: $D=A$, $C=B$, $A \neq B$. Имеем: $g=(a[j,i]b)[l,k](a[j,i]b)$, $h=(b[l,k]a)[j,i](b[l,k]a)$. Из $D=A$ и $C=B$ следует $d=a$, $c=b$, и $k \in I_A$, $l \in J_B$.

7.1. $f_g = f(a[j,i]b,l,k,a[j,i]b)$
$= a[j,i]b \in T_{A[j,i]B}$ & $y_{a[j,i]b}(l) = x_{a[j,i]b}(k)$ & $(l \in q_{a[j,i]b} \sim k \in p_{a[j,i]b})$
$= f(a,j,i,b)$ & $(y_a \cup y_b/\{j\})(l) = (x_a \cup x_b/\{i\})(k)$ & $(l \in q_a \cup q_b \setminus \{j\} \sim k \in p_a \cup p_b \setminus \{i\})$
$= a \in T_A$ & $b \in T_B$ & $y_a(j) = x_b(i)$ & $(j \in q_a \sim i \in p_b)$ & $y_b(l) = x_a(k)$ & $(l \in q_b \sim k \in p_a)$
$= b \in T_B$ & $a \in T_A$ & $y_b(l) = x_a(k)$ & $(l \in q_b \sim k \in p_a)$ & $y_a(j) = x_b(i)$ & $(j \in q_a \sim i \in p_b)$
$= f(b,l,k,a)$ & $(y_b \cup y_a/\{l\})(j) = (x_b \cup x_a/\{k\})(i)$ & $(j \in q_b \cup q_a \setminus \{l\} \sim i \in p_b \cup p_a \setminus \{k\})$
$= b[l,k]a \in T_{B[l,k]A}$ & $y_{b[l,k]a}(j) = x_{b[l,k]a}(i)$ & $(j \in q_{b[l,k]a} \sim i \in p_{b[l,k]a})$
$= f(b[l,k]a,j,i,b[l,k]a) = f_h$.

7.2. $s_g = s_{a[j,i]b} = s_a \cup s_b = s_{b[l,k]a} = s_h$.

7.3. $x_g = x_{a[j,i]b}/\{k\} = (x_a \cup x_b/\{i\})/\{k\} = x_a \cup x_b/\{i,k\}$
$= (x_b \cup x_a/\{k\})/\{i\} = x_{b[l,k]a}/\{i\} = x_h$.

8. $A \rightleftarrows B$. 2 разных автомата, кратные соединения: $C=A$, $D=B$, $A \neq B$. Имеем: $g=(a[j,i]b)[l,k](a[j,i]b)$, $h=(a[l,k]b)[j,i](a[l,k]b)$. Из $C=A$, $D=B$ следует $c=a$, $d=b$, и $k \in I_B$, $l \in J_A$.

8.1. $f_g = f(a[j,i]b,l,k,a[j,i]b)$
$= a[j,i]b \in T_{A[j,i]B}$ & $y_{a[j,i]b}(l) = x_{a[j,i]b}(k)$ & $(l \in q_{a[j,i]b} \sim k \in p_{a[j,i]b})$
$= f(a,j,i,b)$ & $(y_a \cup y_b/\{j\})(l) = (x_a \cup x_b/\{i\})(k)$ & $(l \in q_a \cup q_b \setminus \{j\} \sim k \in p_a \cup p_b \setminus \{i\})$
$= a \in T_A$ & $b \in T_B$ & $y_a(j) = x_b(i)$ & $(j \in q_a \sim i \in p_b)$ & $y_a(l) = x_b(k)$ & $(l \in q_a \sim k \in p_b)$
$= a \in T_A$ & $b \in T_B$ & $y_a(l) = x_b(k)$ & $(l \in q_a \sim k \in p_b)$ & $y_a(j) = x_b(i)$ & $(j \in q_a \sim i \in p_b)$
$= f(a,l,k,b)$ & $(y_a \cup y_b/\{l\})(j) = (x_a \cup x_b/\{k\})(i)$ & $(j \in q_a \cup q_b \setminus \{l\} \sim i \in p_a \cup p_b \setminus \{k\})$
$= a[l,k]b \in T_{A[l,k]B}$ & $y_{a[l,k]b}(j) = x_{a[l,k]b}(i)$ & $(j \in q_{a[l,k]b} \sim i \in p_{a[l,k]b})$
$= f(a[l,k]b,j,i,a[l,k]b) = f_h$.

8.2. $s_g = s_{a[j,i]b} = s_a \cup s_b = s_{a[l,k]b} = s_h$.

8.3. $x_g = x_{a[j,i]b}/\{k\} = (x_a \cup x_b/\{i\})/\{k\} = x_a \cup x_b/\{i,k\}$
$= (x_a \cup x_b/\{k\})/\{i\} = x_{a[l,k]b}/\{i\} = x_h$.

9. A. 1 автомат, две петли: $B=C=D=A$. Имеем: $g=(a[j,i]a)[l,k](a[j,i]a)$, $h=(a[l,k]a)[j,i](a[l,k]a)$. Из $B=C=D=A$ следует $b=c=d=a$, и $i\in I_A$, $l\in J_A$, $k\in I_A$.

9.1. $f_g = f(a[j,i]a,l,k,a[j,i]a)$

$= a[j,i]a \in T_{A[j,i]A}$ & $y_{a[j,i]a}(l)=x_{a[j,i]a}(k)$ & $(l \in q_{a[j,i]a} \sim k \in p_{a[j,i]a})$

$= f(a,j,i,a)$ & $(y_a/\{j\})(l)=(x_a/\{i\})(k)$ & $(l \in q_a\backslash\{j\} \sim k \in p_a\backslash\{i\})$

$= a \in T_A$ & $y_a(j)=x_a(i)$ & $(j \in q_a \sim i \in p_a)$ & $y_a(l)=x_a(k)$ & $(l \in q_a \sim k \in p_a)$

$= a \in T_A$ & $y_a(l)=x_a(k)$ & $(l \in q_a \sim k \in p_a)$ & $y_a(j)=x_a(i)$ & $(j \in q_a \sim i \in p_a)$

$= f(a,l,k,a)$ & $(y_a/\{l\})(j)=(x_a/\{k\})(i)$ & $(j \in q_a\backslash\{l\}) \sim i \in p_a\backslash\{k\})$

$= a[l,k]a \in T_{A[l,k]A}$ & $y_{a[l,k]a}(j)=x_{a[l,k]a}(i)$ & $(j \in q_{a[l,k]a} \sim i \in p_{a[l,k]a})$

$= f(a[l,k]a,j,i,a[l,k]a) = f_h$.

9.2. $s_g = s_{a[j,i]a} = s_a = s_{a[l,k]a} = s_h$.

9.3. $x_g = x_{a[j,i]a}/\{k\} = (x_a/\{i\})/\{k\} = x_a/\{i,k\} = (x_a/\{k\})/\{i\} = x_{a[l,k]a}/\{i\} = x_h$.

Теперь докажем аналогичное утверждение для остальных компонентов перехода: $p_g=p_h$, $y_g=y_h$, $q_g=q_h$, $t_g=t_h$.

Аналогично п.п. 2.2-9.2 для пресостояния s заменой $s\to t$ доказываем утверждение для постсостояния: $t_g=t_h$.

Аналогично п.п. 2.3-9.3 для стимула x:

- заменой $x\to p$ и операций $/\to\backslash$ доказываем $p_g=p_h$,
- заменой (не в индексах) $x\to y$, $i\to j$, $k\to l$ доказываем $y_g=y_h$,
- заменой (не в индексах) $x\to q$, $i\to j$, $k\to l$, и операций $/\to\backslash$ доказываем $q_g=q_h$.

Лемма 1 доказана.

Лемма 2 об ассоциативности композиции автоматов:

Автоматы, которые получаются из автоматов A, B, C, D при композиции по соединениям (j,i) и (l,k), не зависят от того, в каком порядке производится композиция.

Доказательство. Если все 4 автомата разные: $A\neq B$, $A\neq C$, $A\neq D$, $B\neq C$, $B\neq D$, $C\neq D$, то утверждение очевидно: при любом порядке композиции получается множество автоматов $\{A[j,i]B,C[l,k]D\}$.

В остальных случаях результатом композиции по двум соединениям будет не два автомата, а один автомат. Пусть G – автомат, являющийся результатом композиции в порядке (j,i), (l,k), а H – автомат, являющийся результатом композиции в порядке (l,k), (j,i). Тогда нужно доказать, что $G=H$. Поскольку алфавит сообщений при композиции не меняется (остаётся равным M), достаточно доказать, что $I_G=I_H$, $J_G=J_H$, $S_G=S_H$, $T_G=T_H$, $s_{0G}=s_{0H}$.

Аналогично доказательству леммы 1, п.п. 2.2-9.2 для пресостояния s, заменой $s\to s_0$, а также строчных a,b,c,d,g,h на прописные буквы A,B,C,D,G,H,

соответственно, доказываем утверждение для начальных состояний автоматов: $s_{0G}=s_{0H}$.

Аналогично доказательству леммы 1, п.п. 2.3-9.3 для стимула *x*:

- заменой $x \to I$ и операций $/\to\backslash$, а также строчных *a,b,c,d,g,h* на прописные буквы *A,B,C,D,G,H*, соответственно, доказываем $I_G=I_H$,
- заменой (не в индексах) $x \to J$, $i \to j$, $k \to l$, и операций $/\to\backslash$, а также строчных *a,b,c,d,g,h* на прописные буквы *A,B,C,D,G,H*, соответственно, доказываем $J_G=J_H$.

Переходы автоматов совпадают $T_G=T_H$ по доказанной ассоциативности композиции переходов (лемма 1). Поскольку также доказано совпадение начальных состояний автоматов $s_{0G}=s_{0H}$, множества их состояний, состоящие из начальных состояний и пре- и постсостояний переходов, также совпадают $S_G=S_H$.

Лемма 2 доказана.

Лемма 3 об ассоциативности композиции системы автоматов:

Система, которая получается из системы $\boldsymbol{R}$ при композиции по соединениям *(j,i)* и *(l,k)*, не зависит от того, в каком порядке производится композиция: $\boldsymbol{R}[j,i][l,k] = \boldsymbol{R}[l,k][j,i]$.

Доказательство.

1. Алфавит сообщений при композиции не меняется.
2. Докажем, что $E_{\boldsymbol{R}[j,i][l,k]}=E_{\boldsymbol{R}[l,k][j,i]}$.
   Действительно, по определению композиции $E_{\boldsymbol{R}[j,i][l,k]} = (E_{\boldsymbol{R}}\backslash\{(j,i)\})\backslash\{(l,k)\} = (E_{\boldsymbol{R}}\backslash\{(l,k)\})\backslash\{(j,i)\} = E_{\boldsymbol{R}[l,k][j,i]}$.
3. Докажем, что $V_{\boldsymbol{R}[j,i][l,k]}=V_{\boldsymbol{R}[l,k][j,i]}$.
   Доказательство будем вести в зависимости от «топологии» соединений *(j,i)* и *(l,k)*, учитывая доказанную ассоциативность композиции автоматов (лемма 2).

3.1. $A \to B\ C \to D.\ V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}}\backslash\{A,B\}\cup\{A[j,i]B\})\backslash\{C,D\}\cup\{C[l,k]D\}$
$= (V_{\boldsymbol{R}}\backslash\{C,D\}\cup\{C[l,k]D\})\backslash\{A,B\}\cup\{A[j,i]B\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.2. $A \to B \to D.\ V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}}\backslash\{A,B\}\cup\{A[j,i]B\})\backslash\{A[j,i]B,D\}\cup\{(A[j,i]B)[l,k]D\}$
$= (V_{\boldsymbol{R}}\backslash\{B,D\}\cup\{B[l,k]D\})\backslash\{A,B[l,k]D\}\cup\{A[j,i](B[l,k]D)\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.3. $B \leftarrow A \to D.\ V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}}\backslash\{A,B\}\cup\{A[j,i]B\})\backslash\{A[j,i]B,D\}\cup\{(A[j,i]B)[l,k]D\}$
$= (V_{\boldsymbol{R}}\backslash\{A,D\}\cup\{A[l,k]D\})\backslash\{A[l,k]D,B\}\cup\{(A[l,k]D)[j,i]B\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.4. $A \to B \leftarrow C.\ V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}}\backslash\{A,B\}\cup\{A[j,i]B\})\backslash\{C,A[j,i]B\}\cup\{C[l,k](A[j,i]B)\}$
$= (V_{\boldsymbol{R}}\backslash\{C,B\}\cup\{C[l,k]B\})\backslash\{A,C[l,k]B\}\cup\{A[j,i](C[l,k]B)\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.5. $\overset{\curvearrowleft}{A} \to B.\ V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}}\backslash\{A,B\}\cup\{A[j,i]B\})\backslash\{A[j,i]B\}\cup\{(A[j,i]B)[l,k](A[j,i]B)\}$
$= (V_{\boldsymbol{R}}\backslash\{A\}\cup\{A[l,k]A\})\backslash\{A[l,k]A,B\}\cup\{(A[l,k]A)[j,i]B\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.6. $A \overset{\curvearrowright}{\rightarrow} B$. $V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}} \setminus \{A,B\} \cup \{A[j,i]B\}) \setminus \{A[j,i]B\} \cup \{(A[j,i]B)[l,k](A[j,i]B)\}$
$= (V_{\boldsymbol{R}} \setminus \{B\} \cup \{B[l,k]B\}) \setminus \{A,B[l,k]B\} \cup \{A[j,i](B[l,k]B)\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.7. $A \leftrightarrows B$. $V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}} \setminus \{A,B\} \cup \{A[j,i]B\}) \setminus \{A[j,i]B\} \cup \{(A[j,i]B)[l,k](A[j,i]B)\}$
$= (V_{\boldsymbol{R}} \setminus \{B,A\} \cup \{B[l,k]A\}) \setminus \{B[l,k]A\} \cup \{(B[l,k]A)[j,j](B[l,k]A)\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.8. $A \rightleftarrows B$. $V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}} \setminus \{A,B\} \cup \{A[j,i]B\}) \setminus \{A[j,i]B\} \cup \{(A[j,i]B)[l,k](A[j,i]B)\}$
$= (V_{\boldsymbol{R}} \setminus \{A,B\} \cup \{A[l,k]B\}) \setminus \{A[l,k]B\} \cup \{(A[l,k]B)[j,j](A[l,k]B)\} = V_{\boldsymbol{R}[l,k][j,i]}$.

3.9. $\overset{\curvearrowleft}{A}$. $V_{\boldsymbol{R}[j,i][l,k]} = (V_{\boldsymbol{R}} \setminus \{A\} \cup \{A[j,i]A\}) \setminus \{A[j,i]A\} \cup \{(A[j,i]A)[l,k](A[j,i]A)\}$
$= (V_{\boldsymbol{R}} \setminus \{A\} \cup \{A[l,k]A\}) \setminus \{A[l,k]A\} \cup \{(A[l,k]A)[j,j](A[l,k]A)\} = V_{\boldsymbol{R}[l,k][j,i]}$.

Лемма 3 доказана.

Теорема 3 об ассоциативности композиции системы по всем соединениям:

Композиция системы $\boldsymbol{R}$ по всем её соединениям не зависит от порядка композиции: для любых двух последовательностей всех соединений $Z^\wedge$ и $Z^{\wedge`}$ имеет место $\boldsymbol{R}[Z^\wedge] = \boldsymbol{R}[Z^{\wedge`}]$.

Доказательство. Утверждение теоремы непосредственно следует из леммы 3.

Мы доказали, что композиция системы $\boldsymbol{R}$ по всем её соединениям не зависит от порядка $Z^\wedge$ соединений, в котором композиция выполняется. Результатом является система $\boldsymbol{R}[Z^\wedge]$, в которой нет соединений, т.е. $E_{\boldsymbol{R}[Z^\wedge]} = \varnothing$, а все входы и выходы являются внешними. Если в системе $\boldsymbol{R}$ были автоматы, которые не связаны друг с другом, то в $\boldsymbol{R}[Z^\wedge]$ окажется несколько автоматов, т.е. $|V_{\boldsymbol{R}[Z^\wedge]}| > 1$. Очевидно, эти автоматы также не связаны друг с другом.

5. Система как компонент другой системы

Для того чтобы система автоматов $\boldsymbol{R}$ могла использоваться как компонент более сложной системы, композиция системы $\boldsymbol{R}[Z^\wedge]$ по всем её соединениям должна состоять из одного автомата. Однако если в системе есть несвязанные автоматы, то $\boldsymbol{R}[Z^\wedge]$ содержит несколько несвязанных автоматов. Для того чтобы обойти это неудобство, достаточно определить композицию несвязанных автоматов. Семантика такой композиции достаточно прозрачна: если два автомата не связаны, то работа системы из этих автоматов эквивалентна работе одного автомата, в котором объединяются входы и выходы обоих автоматов, а любые два перехода из разных автоматов объединяются в один переход. Определим такую композицию формально.

Композиция переходов без соединения. Определим композицию $a[]b$ двух переходов a и b без соединения:

$a[]b \triangleq (s_a \cup s_b,\ x_a \cup x_b,\ p_a \cup p_b,\ y_a \cup y_b,\ q_a \cup q_b,\ t_a \cup t_b)$.

В этом определении неявно предполагается, что как домены стимулов, так и домены реакций в компонуемых переходах не пересекаются:

$\boldsymbol{Dom}(x_a) \cap \boldsymbol{Dom}(x_b) = \boldsymbol{Dom}(y_a) \cap \boldsymbol{Dom}(y_b) = \varnothing$. В противном случае стимул и/или реакция в композиционном переходе могли бы стать многозначными функциями. Мы не записали это требование как условие композиции двух переходов, поскольку такая композиция будет использоваться только для определения композиции несвязанных и, следовательно, разных автоматов, а в этом случае это требование автоматически выполнено.

Композиция переходов без соединения естественным образом распространяется на композицию без соединения множеств переходов G и H, определяемую как множество попарных композиций переходов:

$G[]H \triangleq \{a[]b | a \in G \ \&\ b \in H\}$.

Композиция несвязанных автоматов.

Пусть в системе $\boldsymbol{R} = (M,V,E)$ есть два несвязанных между собой автомата A и B. Тогда их композиция определяется следующим образом:

$A[]B \triangleq (M, I_A \cup I_B, J_A \cup J_B, \{s_{0A} \cup s_{0B}\} \cup States(T_A[]T_B), T_A[]T_B, s_{0A} \cup s_{0B})$.

Очевидно, что такая композиция автоматов удовлетворяет условиям, налагаемым на автомат (аналог теоремы 1).

Композиция системы, состоящей из несвязанных автоматов.

При композиции двух несвязанных автоматов A и B в системе $\boldsymbol{R} = (M,V,E)$ получается новая система: $\boldsymbol{R}[A,B] \triangleq (M, V \setminus \{A,B\} \cup \{A[]B\}, E)$.

Очевидно, что такая композиция системы автоматов удовлетворяет условиям, налагаемым на систему автоматов (аналог теоремы 2).

Легко показать, что композиция переходов без соединения, композиция несвязанных автоматов и композиция системы, состоящей из несвязанных автоматов, ассоциативны.

Каждая композиция уменьшает на 1 число автоматов в системе. Поэтому через конечное число композиций система $\boldsymbol{R}[Z\hat{}]$ превратится в систему без соединений, состоящую из одного автомата.

Нетрудно показать, что композицию несвязанных автоматов можно чередовать с композицией по соединениям. Можно только отметить, что композиция несвязанных автоматов уменьшает на 1 число компонентов связности в графе $\boldsymbol{Ng(R)}$: автомат $A[]B$ оказывается связанным с каждым автоматом (отличном от A и B), с которым был связан автомат A или B.

6. Заключение

Наша предыдущая статья [1] была посвящена проблеме тестирования системы автоматов. В отличие от данной статьи сообщения между автоматами, расположенными в вершинах графа связей, буферизовались соединяющими автоматы дугами этого графа. Каждая дуга представляла собой очередь длины 1 и моделировалась специальным автоматом с одним входом и одним выходом, который взаимодействовал с автоматами вершин, инцидентных этой

дуге, уже синхронно. В данной статье автомат дуги обобщён таким образом, что он может моделировать очередь уже произвольной, в том числе, неограниченной длины, или стек, или очередь с приоритетами и т.д. Более того, автомат дуги может иметь уже несколько входов и несколько выходов. Тем самым, стирается само различие между автоматом дуги и автоматом вершины. Композиция определяется по соединению, такое соединение *(j,i)* связывает выход *j* автомата $A=\varphi(j)$ со входом *i* автомата $B=\varphi(i)$ и моделирует не буферизующий автомат дуги, а синхронное взаимодействие.

Однако главная цель, которая была заявлена в статье [1], это тестирование системы автоматов в предположении, что выполнена *гипотеза о связях*: граф связей, т.е. отображаемая им структура связей, не содержит ошибок. В этом случае целью тестирования становится покрытие переходов автоматов компонентов, которые достижимы при работе этих автоматов в системе. Предполагается, что при тестировании возможно наблюдение изменения состояний автоматов в вершинах графа и сообщений на дугах графа. В [1] на примере показано, что гипотеза о связях позволяет существенно сократить время такого тестирования. Полное тестирование системы автоматов без учёта гипотезы о связях может потребовать число тестовых воздействий порядка произведения чисел состояний автоматов компонентов, а с учётом гипотезы о связях – порядка суммы этих чисел. При равном числе состояний всех автоматов это даёт экспоненциальное уменьшение числа тестовых воздействий.

В [1] был предложен алгоритм генерации тестов, основанный на фильтрации тестов, генерируемых для покрытия всех переходов композиции: тест отбрасывается, если он покрывает только такие переходы в компонентах системы, которые покрываются остающимися тестами. В дальнейшем предполагается разработать модификацию этого алгоритма для общей системы автоматов, определённой в данной статье.

Этот алгоритм существенно опирается на детерминированность системы автоматов. Конечно, одним из источников недетерминизма системы может служить недетерминизм её компонентов. Однако проблема в том, что система может вести себя недетерминированно даже в том случае, когда каждый её компонент представляет собой детерминированный автомат. Поэтому требуется найти такие, по возможности минимальные, ограничения на устройство системы детерминированных автоматов, при которых система будет детерминированной.

Список литературы

[1]. И.Б.Бурдонов, А.С.Косачев. Тестирование системы автоматов. Труды ИСП РАН, том 28(1), 2016 г.

[2]. Hoare C.A.R. Communicating Sequential Processes. Englewood Cliffs, NJ: Prentice Hall International, 1985.

[3]. Milner R. Communication and Concurrency. Prentice-Hall, 1989.

[4]. Langerak R. A testing theory for LOTOS using deadlock detection. In E.Brinksma, G.Scollo, and C.A.Vissers, editors, Protocol Specification, Testing, and Verification IX, pages 87–98. North-Holland, 1990.
[5]. Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3, 1996.
[6]. van der Bijl M., Rensink A., Tretmans J. Compositional testing with ioco. In Formal Approaches to Software Testing: Third International Workshop, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003. Editors: Alexandre Petrenko, Andreas Ulrich ISBN: 3-540-20894-1. LNCS volume 2931, Springer, pp. 86-100.
[7]. Petrenko A., Yevtushenko N., Huo J.L. Testing Transition Systems with Input and Output Testers. Proc. IFIP TC6/WG6.1 15th Int. Conf. Testing of Communicating Systems, TestCom'2003, pp. 129-145. Sophia Antipolis, France, May 26-29, 2003.
[8]. И.Б.Бурдонов, А.С.Косачев. Системы с приоритетами: конформность, тестирование, композиция. "Программирование", 2009, №4, стр. 24-40.
[9]. И.Б.Бурдонов, А.С.Косачев. Согласование конформности и композиции. Программирование, №6, 2013, стр. 3-15.

Automata system: composition according to graph of links

I.B. Burdonov <igor@ispras.ru>
A.S. Kossatchev <kos@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The problem of modeling and composing of aggregate systems is considered. The system components are described with finite automata with multiple entries and exits. The communication between automata is described with message passing over simplex communication channels. The system is described with a directed graph of links. Each node of the graph corresponds to automaton of a component and an arc corresponds to a communication channel connecting exit of one automaton with entry of another automaton. Automaton of the graph node in each state can accept multiple messages from its entries (at most one message from each entry) and send multiple messages to its exits (at most one message to each exit). Entries (exits) of the automata not connected to exits (entries) of automata are considered to be external and used for communication between the system and its environment. The automata of the system operate synchronously: on each cycle each automaton performs one transition. A transition of an automaton imposes requirements on states of all its entries and exits (messages in them are specified) and explicitly specifies subset of entries and exits through which the messages are received or sent, respectively. Synchronous communication between automata means that for each link the requirements of the automata connected with this link must conform to each other. It makes possible to describe a wider spectrum of automata behavior. For example, a priority of message receiving: if there are multiple message in the automata entries, it can receive messages with the highest priority and discard the rest of the messages. It also makes possible for the automaton to receive messages regardless of ability to simultaneously send some message to some exit. A composition of the automata of the system according to the graph of links is defined and its associativity is proved. In conclusion, the directions of future research are described.

Keywords: directed graphs, communicating automata, automata composition, distributed systems, networks.

DOI: 10.15514/ISPRAS-2016-28(1)-8

For citation: Burdonov I.B., Kossatchev A.S. Automata system: composition according to graph of links. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 131-150 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-8

References

[1]. I. B. Burdonov, A. S. Kossatchev. Testirovanie sistemy avtomatov [Testing of automata system]. Trudy ISP RAN [The proceeding of ISP RAS], Vol. 28(1), 2016 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-7

[2]. Hoare C.A.R. Communicating Sequential Processes. Englewood Cliffs, NJ: Prentice Hall International, 1985.

[3]. Milner R. Communication and Concurrency. Prentice-Hall, 1989.

[4]. Langerak R. A testing theory for LOTOS using deadlock detection. In E.Brinksma, G.Scollo, and C.A.Vissers, editors, Protocol Specification, Testing, and Verification IX, pages 87–98. North-Holland, 1990.

[5]. Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3, 1996.

[6]. van der Bijl M., Rensink A., Tretmans J. Compositional testing with ioco. In Formal Approaches to Software Testing: Third International Workshop, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003. Editors: Alexandre Petrenko, Andreas Ulrich ISBN: 3-540-20894-1. LNCS volume 2931, Springer, pp. 86-100.

[7]. Petrenko A., Yevtushenko N., Huo J.L. Testing Transition Systems with Input and Output Testers. Proc. IFIP TC6/WG6.1 15th Int. Conf. Testing of Communicating Systems, TestCom'2003, pp. 129-145. Sophia Antipolis, France, May 26-29, 2003.

[8]. I. B. Burdonov, A. S. Kossatchev. Systems with Priorities: Conformance, Testing, and Composition. Programming and Computer Software,Vol. 35, No. 4, 2009, pp.198-211. DOI: 10.1134/S0361768809040045

[9]. I. B. Burdonov, A. S. Kossatchev. Agreement between Conformance and Composition. Programming and Computer Software,Vol. 39, No. 6, 2013, pp. 269–278. *DOI*: 10.1134/S0361768813060029

Система автоматов: условия детерминизма и тестирование

И.Б. Бурдонов <igor@ispras.ru>
А,С. Косачев <kos@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. Статья посвящена проблеме тестирования составных систем, компоненты которых моделируются конечными автоматами с несколькими входами и выходами, а взаимодействие между ними – обменом сообщениями по симплексным каналам связи. Система описывается ориентированным графом связей, вершина которого соответствует автомату компонента, а дуга – каналу связи, соединяющему выход одного автомата со входом другого автомата. Предполагается выполненной следующая гипотеза о связях: граф связей статический, а отображаемая им структура связей не содержит ошибок. Автомат, находящийся в вершине графа, в каждом состоянии может принимать несколько сообщений по своим входам (не более одного по каждому входу) и посылать несколько сообщений по своим выходам (не более одного по каждому выходу). Определяется ассоциативная композиция автоматов системы. Показывается, при каких ограничениях на автоматы и граф связей их композиция, т.е. сама система, детерминирована. Целью тестирования является покрытие переходов автоматов компонентов, которые достижимы при работе этих автоматов в системе. Предполагается, что при тестировании возможно наблюдение изменения состояний автоматов системы и передаваемых между ними сообщений. Полное тестирование системы автоматов без учёта гипотезы о связях может потребовать число тестовых воздействий порядка произведения чисел состояний автоматов компонентов, а с учётом гипотезы о связях – порядка суммы этих чисел. При равном числе состояний всех автоматов это даёт экспоненциальное уменьшение числа тестовых воздействий. При условии выполнения гипотезы о связях для детерминированной системы предлагается алгоритм генерации тестов, основанный на фильтрации тестов, генерируемых для покрытия всех переходов композиции. Тест отбрасывается, если он покрывает только такие переходы в компонентах системы, которые покрываются остающимися тестами. В заключение определяются направления дальнейших исследований.

Ключевые слова: ориентированные графы, покрытие графа, взаимодействующие автоматы, композиция автоматов, детерминизм, распределенные системы, тестирование, сети.

DOI: 10.15514/ISPRAS-2016-28(1)-9

Для цитирования: Бурдонов И.Б., Косачев А.С. Система автоматов: условия детерминизма и тестирование. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 151-184. DOI: 10.15514/ISPRAS-2016-28(1)-9

1. Введение

Большинство сложных, особенно распределённых, систем представляет собой набор взаимодействующих компонентов. Данная статья продолжает исследование системы автоматов, начатое в нашей предыдущей статье [2]. Компоненты моделируются конечными автоматами, а взаимодействие – обменом сообщениями между автоматами. Автомат может иметь несколько входов для приёма сообщений и может принимать сразу несколько сообщений; также автомат может иметь несколько выходов для посылки сообщений и может посылать сразу несколько сообщений. Структура связей между компонентами моделируется ориентированным графом (будем называть его *графом связей*), вершинам которого соответствуют автоматы, а дуги называются *соединениями* и соответствуют симплексным каналам передачи сообщений. Соединение, ведущее из автомата *A* в автомат *B*, помечается выходом *j* автомата *A* и входом *i* автомата *B*. При этом каждый вход (выход) каждого автомата соединён не более чем с одним выходом (входом) другого (или того же самого) автомата. Входы (выходы) автоматов, которые не соединяются с выходами (входами) автоматов, являются *внешними*, через них осуществляется связь системы с её *окружением*. При тестировании тест подменяет собой окружение: он передаёт сообщения в систему на её внешние входы (набор таких сообщений можно назвать внешним *стимулом*) и принимает от системы сообщения с её внешних выходов (набор таких сообщений можно назвать внешней *реакцией*).

Считается, что граф связей статический, то есть не меняющийся в процессе работы системы. В этом случае система (также как её компоненты) может моделироваться конечным автоматом, получающимся из автоматов-компонентов с помощью подходящего оператора композиции, учитывающего граф связей.

Система работает правильно, если структура её связей правильна, и каждый автомат-компонент работает правильно. Обратное, вообще говоря, не верно, если требования к системе не однозначно определяют её структуру. Например, функциональные требования, определяющие внешнюю реакцию системы как требуемую функцию от последовательности внешних стимулов. В данной статье целью тестирования является покрытие всех переходов автоматов компонентов, которые достижимы при работе этих автоматов в системе. Поэтому, если в структуре связей нет ошибок, т.е. множество соединений совпадает с заданным (будем называть это *гипотезой о связях*), то такое тестирование системы сводится к проверке правильности переходов каждого автомата. Проблема в том, что каждый автомат может тестироваться только как часть системы. Это означает, что тест не имеет непосредственного

доступа к автомату-компоненту, и вынужден осуществлять тестовые воздействия с помощью сообщений, посылаемых по внешним входам, которые принадлежат, быть может, другим автоматам. Тестирование компонента такой системы похоже на тестирование в контексте ([3], [4], [5], [6]), когда этот компонент рассматривается как тестируемая система, а остальные – как контекст. Существенное отличие, однако, в том, что в таком контексте тоже могут быть ошибки, хотя, если верна гипотеза о связях, то только в компонентах, а не в структуре связей между ними. С другой стороны, один тест может проверять работу сразу нескольких компонентов, через которые проходят сообщения.

Поскольку компонент тестируется как часть системы, не все переходы автомата компонента, которые можно проверить при автономном тестировании с прямым доступом к компоненту, можно проверить при тестировании системы. Поэтому речь должна идти только о *достижимых переходах* автоматов, то есть таких переходах, которые могут быть выполнены при работе автомата как части системы.

Как мы показали в [1] на примере гипотеза о связях позволяет получить существенный выигрыш во времени тестирования, вплоть до экспоненциального его уменьшения. Если гипотеза о связях верна, то целью тестирования системы автоматов становится покрытие всех достижимых переходов автоматов компонентов системы.

В модели, предложенной в [2], автоматы системы работают синхронно: на каждом такте каждый автомат выполняет один переход. Также на каждом такте окружение системы посылает сообщения на некоторые внешние входы системы и принимает сообщения с некоторых внешних выходов системы. Для получения замкнутой формальной системы окружение моделируется автоматом, выходы которого соединяются с внешними входами системы, а внешние выходы системы соединяются со входами окружения. При тестировании роль окружения играет тест.

В данной статье предлагается алгоритм построения набора тестов, аналогичный алгоритму, предложенному в [1] для систем частного вида. Этот набор тестов является полным (проверяет все достижимые переходы автоматов компонентов) при выполнении двух условий: 1) верна гипотеза о связях, 2) система детерминирована. Дополнительно этот алгоритм определяет недостижимые переходы автоматов компонентов. Предполагается, во-первых, что нам известно, каким должен быть автомат каждого компонента (задан граф переходов автомата с точностью до изоморфизма) и именно это должно проверяться при тестировании. Во-вторых, тестовая система может не только выполнять тест, посылая сообщения на внешние входы системы и принимая сообщения с внешних выходов системы, но и наблюдать как состояния автоматов системы, так и сообщения, передаваемые по соединениям между автоматами системы. Такие предположения могут быть оправданы, например,

при имитационном тестировании аппаратуры (simulation-based verification) (см. например, [7]).

Сначала, в разделе 2, повторяются необходимые определения и утверждения (без доказательств) из [2]. Формально определяются модель автомата и модель системы, а также композиция переходов, композиция автоматов и композиция системы по соединению. Композиция автоматов удовлетворяет требованиям, предъявляемым к автомату, а композиция системы удовлетворяет требованиям, предъявляемым к системе автоматов. Композиция системы по всем её соединениям представляет собой автомат или набор автоматов, не связанных между собой соединениями. Композиция переходов, автоматов и системы обладает свойством ассоциативности. Ассоциативность важна для того, чтобы работа системы зависела только от множества её автоматов и соединений и не зависела от какого-либо упорядочивания этих множеств.

В разделе 3 даётся определение вполне определённого и детерминированного автомата, формулируются условия, при которых система таких автоматов тоже вполне определённая и детерминированная и доказывается соответствующая теорема.

В разделе 4 рассматриваются системы автоматов класса «вершины-дуги», которые являются обобщением системы автоматов, изучавшейся в [1].

Раздел 5 содержит описание алгоритма генерации полного набора тестов системы при выполнении гипотезы о связях.

В заключение намечаются направления дальнейших исследований.

2. Модель системы и композиция

В этом разделе мы повторим основные определения и утверждения из нашей предыдущей статьи [2].

Пусть задано некоторое конечное *множество сообщений* M, которое мы будем называть *алфавитом* (сообщений). Пустое множество $\varnothing$ будем называть *пустым сообщением*, считать, что $\varnothing \notin M$, и обозначать $M_\varnothing \triangleq M \cup \{\varnothing\}$.

Автомат в алфавите M – это набор $A=(M,I,J,S,T,s_0)$, где

I – конечное множество *входов* автомата,

J – конечное множество *выходов* автомата,

S – конечное множество *состояний* автомата,

$T \subseteq S \times X \times P \times Y \times Q \times S$ – множество *переходов* автомата, где

$X = \{ x \mid x: I \to M_\varnothing \}$ – множество *стимулов,*

$Y = \{ y \mid y: J \to M_\varnothing \}$ – множество *реакций,*

$P = 2^I$ – семейство подмножеств входов (по ним принимаются сообщения),

$Q = 2^J$ – семейство подмножеств выходов (по ним посылаются сообщения),

$s_0 \in S$ – *начальное состояние,*

причём выполнены следующие условия:

1. входы и выходы не пересекаются: $I \cap J = \varnothing$,
2. приниматься по входу и передаваться по выходу может только непустое сообщение: $\forall (s, x, p, y, q, t) \in T\ (\forall i \in p\ x(i) \neq \varnothing\ \&\ \forall j \in q\ y(j) \neq \varnothing)$, что эквивалентно $\forall (s, x, p, y, q, t) \in T\ (p \subseteq x^{-1}(M)\ \&\ q \subseteq y^{-1}(M))$.

Очевидно, что из конечности множеств M, I, J и S следует конечность множеств X, Y, P, Q, T.

Для перехода $a = (s, x, p, y, q, t)$ состояние s будем называть *пресостоянием* перехода, а состояние t – *постсостоянием*, и будем обозначать $s_a=s$, $x_a=x$, $p_a=p$, $y_a=y$, $q_a=q$, $t_a=t$. Для автомата $A=(M,I,J,S,T,s_0)$ будем обозначать $I_A=I$, $J_A=J$, $S_A=S$, $T_A=T$, $s_{0A}=s_0$, а также $X_A=X$, $Y_A=Y$, $P_A=P$, $Q_A=Q$.

Пусть V – конечное множество автоматов в алфавите M. Без ограничения общности можно считать, что входы, выходы и состояния автоматов разные: $\forall A,B \in V\ I_A \cap I_B = \varnothing\ \&\ I_A \cap J_B = \varnothing\ \&\ J_A \cap I_B = \varnothing\ \&\ J_A \cap J_B = \varnothing\ \&\ S_A \cap S_B = \varnothing$. Этого всегда можно добиться с помощью систематического переименования состояний, входов и выходов автоматов, в результате которого получаются автоматы изоморфные исходным.

Система автоматов – это набор $\boldsymbol{R} = (M,V,E)$, где $E : E_{Dom} \to E_{Im}$ – биекция, определяющая *соединения*, где $E_{Dom} \subseteq J_{\boldsymbol{R}}$, $E_{Im} \subseteq I_{\boldsymbol{R}}$, $J_{\boldsymbol{R}} = \cup\{J_A | A \in V\}$ – множество всех выходов всех автоматов, $I_{\boldsymbol{R}} = \cup\{I_A | A \in V\}$ – множество всех входов всех автоматов. Для системы $\boldsymbol{R} = (M,V,E)$ обозначим: $V_{\boldsymbol{R}}=V$, $E_{\boldsymbol{R}}=E$.

Определим функцию $\varphi : (I_{\boldsymbol{R}} \cup J_{\boldsymbol{R}}) \to V$, которая каждому входу или выходу системы $\boldsymbol{R}$ ставит в соответствие автомат, которому этот вход или выход принадлежит, т.е. $\forall A \in V\ \forall z \in I_A \cup J_A\ \varphi(z)=A$. Будем говорить, что соединение (j,i) ведёт из автомата $\varphi(j)$ в автомат $\varphi(i)$. Функция φ зависит от системы автоматов, однако для сокращения записи везде, где подразумевается система $\boldsymbol{R}$, мы её не указываем в параметрах функции φ. Если функция φ применяется для другой системы $\boldsymbol{R}` \neq \boldsymbol{R}$, мы будем писать $\varphi_{\boldsymbol{R}`}$.

Вход $i \in I_{\boldsymbol{R}}$, для которого не определено соединение, т.е. $i \notin E_{Im}$, будем называть *внешним входом системы*, и выход $j \in J_{\boldsymbol{R}}$, для которого не определено соединение, т.е. $j \notin E_{Dom}$, будем называть *внешним выходом системы*.

Системе $\boldsymbol{R} = (M,V,E)$ соответствует ориентированный граф с помеченными дугами, вершинами которого являются автоматы из V, а дуга $A \to B$ существует и помечена соединением (j,i) тогда и только тогда, когда $(j,i) \in E$, $\varphi(j)=A$ и $\varphi(i)=B$. Такой граф обозначим через $\boldsymbol{Og(R)}$. Если снять ориентацию дуг, то соответствующий неориентированный граф обозначим через $\boldsymbol{Ng(R)}$. Будем говорить, что автоматы A и B *связаны* в системе, если в графе $\boldsymbol{Ng(R)}$ существует (неориентированный) путь из A в B (возможно, пустой, если $A=B$). В противном случае разные автоматы A и B *не связаны*. Очевидно, что отношение связанности автоматов рефлексивно, симметрично и транзитивно, т.е. является отношением эквивалентности, которое разбивает множество автоматов V на классы эквивалентности, которые вместе с соединениями

между автоматами одного класса мы будем называть *компонентами связности*.

Соединение $(j,i)\in E$ будем называть *петлёй*, если это дуга-петля в графе $\boldsymbol{Og(R)}$, то есть $\varphi(j)=\varphi(i)$.

Для удобства будем считать, что каждое состояние каждого автомата в системе $\boldsymbol{R}=(M,V,E)$ является множеством, причём состояния разных автоматов в системе являются непересекающимися множествами: $\forall A,B\in V\ \forall s_A\in S_A\ \forall s_B\in S_B\ (A\neq B \Rightarrow s_A\cap s_B=\varnothing)$. Если уже задана система, в которой это условие не выполнено, то можно преобразовать каждый автомат, заменив в нём каждое состояние s на синглетон $\{s\}$. Поскольку в системе автоматов состояния разных автоматов разные, то таким преобразованием будут получены автоматы изоморфные исходным, а для системы автоматов будет выполнено условие попарного непересечения множеств состояний автоматов.

Введём способ сокращённой записи для преобразования функции с помощью уменьшения её домена: для произвольной функции f и произвольного множества N обозначим $f/N \triangleq f \setminus \{(z,f(z)) \mid z\in N\cap \boldsymbol{Dom}(f)\}$. Очевидно, что для произвольного множества $N`$ имеет место $f/(N\cup N`) = (f/N)/N`$ и для произвольной функции g такой, что $\boldsymbol{Dom}(f)\cap\boldsymbol{Dom}(g)=\varnothing$ и $N\cap\boldsymbol{Dom}(g)=\varnothing$, имеет место $(f\cup g)/N = (f/N)\cup g$. Будем считать, что операция "/" имеет тот же приоритет, что разность "\" и объединение "$\cup$" множеств. Как следствие, в выражениях над множествами, где используются только операции "/", "\" и "$\cup$", мы будем использовать бесскобочную запись и предполагать, что операции выполняются слева направо.

Логическую эквивалентность будем обозначать символом «~»: $a\sim b \triangleq (a\&b)\vee(\neg a\&\neg b)$.

Композиция определяется по соединению $(j,i)\in E$. Обозначим через A автомат, которому принадлежит выход j, т.е. $A=\varphi(j)$, а через B автомат, которому принадлежит вход i, т.е. $B=\varphi(i)$.

Композиция переходов по соединению. Для соединения (j,i) определим условие $f(a,j,i,b)$ композиции двух переходов a и b и результат композиции $a[j,i]b$:

$f(a,j,i,b) \triangleq a\in T_A\ \&\ b\in T_B\ \&\ (A=B \Rightarrow a=b)\ \&\ y_a(j)=x_b(i)\ \&\ (j\in q_a \sim i\in p_b)$,

$f(a,j,i,b)$: $a[j,i]b \triangleq (s_a\cup s_b,\ x_a\cup x_b/\{i\},\ p_a\cup p_b\setminus\{i\},\ y_a\cup y_b/\{j\},\ q_a\cup q_b\setminus\{j\},\ t_a\cup t_b)$.

Заметим, что если $A=B$, то

$f(a,j,i,a) \triangleq a\in T_A\ \&\ y_a(j)=x_a(i)\ \&\ (j\in q_a \sim i\in p_a)$,

$f(a,j,i,a)$: $a[j,i]a \triangleq (s_a,\ x_a/\{i\},\ p_a\setminus\{i\},\ y_a/\{j\},\ q_a\setminus\{j\},\ t_a)$.

Композиция переходов естественным образом распространяется на композицию множеств переходов, определяемую как множество попарных

композиций переходов. Для соединения *(j,i)* определим композицию множеств переходов *G* и *H*: $G[j,i]H \triangleq \{a[j,i]b \mid a \in G \ \&\ b \in H \ \&\ f(a,j,i,b)\}$.

Композиция автоматов по соединению.

Для множества переходов *T* обозначим множество пре- и постсостояний этих переходов: $States(T) \triangleq \{s_a \mid a \in T\} \cup \{t_a \mid a \in T\}$. Для соединения *(j,i)* определим композицию автоматов *A* и *B*:

$A[j,i]B \triangleq (M, I_A \cup I_B \setminus \{i\}, J_A \cup J_B \setminus \{j\}, \{s_{0A} \cup s_{0B}\} \cup States(T_A[j,i]T_B), T_A[j,i]T_B, s_{0A} \cup s_{0B})$.

Композиция системы по соединению. Для системы $\boldsymbol{R}$ и соединения *(j,i)* определим $\boldsymbol{R}[j,i] \triangleq (M, V \setminus \{A,B\} \cup \{A[j,i]B\}, E \setminus \{(j,i)\})$.

В [2] доказано, что композиция автоматов и системы определены корректно, т.е. композиция автоматов удовлетворяет условиям, налагаемым на автомат (теорема 1 в [2]), а композиция системы удовлетворяет условиям, налагаемым на систему (теорема 2 в [2]).

Последовательностью соединений для системы $\boldsymbol{R} = (M,V,E)$ назовём последовательность *Z* соединений из *E*, в которой каждое соединение встречается не более одного раза, т.е. $Z:[1..|Z|] \to E$ является инъекцией. Определим композицию системы $\boldsymbol{R}$ по последовательности *Z* как $\boldsymbol{R}[Z] \triangleq \boldsymbol{R}[Z(1)][Z(2)]...[Z(|Z|)]$, если *Z* не пусто, и $\boldsymbol{R}[Z] \triangleq \boldsymbol{R}$, если *Z* пусто.

Последовательность соединений, являющуюся биекцией, т.е. содержащую каждое соединение из *E*, будем называть *последовательность всех соединений* и обозначать *Z^*. Очевидно, что в системе $\boldsymbol{R}[Z^]$ нет соединений, т.е. $E_{\boldsymbol{R}[Z^]} = \varnothing$, т.е. все входы и выходы автоматов являются внешними. Если в системе $\boldsymbol{R}$ были автоматы, которые не связаны друг с другом, то в $\boldsymbol{R}[Z^]$ окажется несколько автоматов, соответствующих компонентам связности в $\boldsymbol{R}$, и все эти автоматы также не связаны друг с другом.

В [2] доказана ассоциативность композиции переходов (лемма 1 в [2]), автоматов (лемма 2 в [2]) и системы (лемма 3 в [2]), т.е. независимость результата композиции по двум соединениям $(j,i) \in E$ и $(l,k) \in E$ от порядка композиции: сначала по *(j,i)*, а потом по *(l,k)*, или, наоборот, сначала по *(l,k)*, а потом по *(j,i)*. Как следствие, композиция системы $\boldsymbol{R}$ по всем её соединениям не зависит от порядка композиции (теорема 3 в [2]): для любых двух последовательностей всех соединений *Z^* и *Z^`* имеет место $\boldsymbol{R}[Z^] = \boldsymbol{R}[Z^`]$. Ассоциативность важна для того, чтобы работа системы зависела только от множества её автоматов и соединений и не зависела от какого-либо упорядочивания этих множеств.

3. Вполне определённость и детерминизм

В композиции, которая определена выше, нет «асинхронных» переходов, все переходы «синхронные»: переход в каждом автомате выполняется одновременно с переходами в автоматах, связанных с ним по соединениям.

Это означает, что если какой-то автомат «стоит», т.е. в текущем своем состоянии не может выполнить никакого перехода, то «стоят» все автоматы из того же компонента связности, и это «стояние» продолжается бесконечно до рестарта системы. Это ситуация тупика (deadlock).

Тупик возникает из-за несогласованности требований автоматов, связанных соединением. Эти требования определяются условиями композиции переходов $a[j,i]b$: 1) $y_a(j)=x_b(i)$ и 2) $j \in q_a \sim i \in p_b$. Для того чтобы тупик был невозможен из-за нарушения условия 1, достаточно потребовать, чтобы либо А) в каждом автомате в каждом состоянии был определен переход по каждому стимулу x, либо В) в каждом автомате в каждом состоянии был определен переход по каждой реакции y. Для того чтобы тупик был невозможен из-за нарушения условия 2, достаточно потребовать, чтобы либо А) в каждом автомате в каждом состоянии для каждой реакции y был определен переход с каждым возможным параметром q, определяющим по какому выходу j сообщение $y(j)$ передаётся, а по какому нет, либо В) в каждом автомате в каждом состоянии для каждого стимула x был определен переход с каждым параметром p, определяющим по какому входу i сообщение $x(i)$ принимается, а по какому нет. Мы выбираем варианты А для обоих условий, что формализовано в определении вполне определенного автомата.

Автомат $A=(M,I,J,S,T,s_0)$ будем называть *определённым по всем стимулам*, если в каждом состоянии есть переход по каждому стимулу:

1. $\forall s \in S \ \forall x \in X \ \exists a \in T \ s_a=s \ \& \ x_a=x$,

Автомат $A=(M,I,J,S,T,s_0)$ будем называть *вполне определенным*, если он определенный по всем стимулам и есть переходы по всем возможным q:

2. $\forall a \in T \ \forall q` \subseteq y_a^{-1}(M) \ \exists a` \in T \ s_{a`}=s_a \ \& \ x_{a`}=x_a \ \& \ p_{a`}=p_a \ \& \ y_{a`}=y_a \ \& \ q_{a`}=q`$.

Вместо условий 1 и 2 можно использовать *правила умолчания*, если в определении автомата указывать подмножество T^* явных переходов, и дополнять его до множества T всех переходов согласно правилам умолчания:

1) Если T^* не содержит явного перехода в некотором состоянии по некоторому стимулу, то это означает, что такой переход определён неявно без приёма и выдачи сообщений и без изменения состояния.
2) Если в T^* есть переход (s, x, p, y, q, t), но для некоторого $q` \subseteq y^{-1}(M)$ нет перехода вида $(s, x, p, y, q`, t`)$, то это означает, что такой переход определён неявно без изменения состояния ($t`=t$).

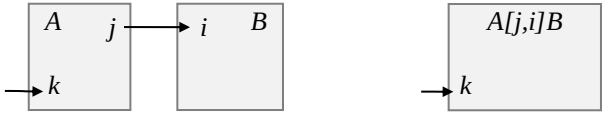
Формально это означает:

$$T \triangleq T^* \cup \{(s, x, \varnothing, J \times \{\Lambda\}, \varnothing, s) \mid s \in S \ \& \ x \in X \ \& \ \forall p,y,q,t \ (s,x,p,y,q,t) \notin T^*\}$$

$$\cup \{(s,x,p,y,q`,s) \mid \exists q,t \ (s,x,p,y,q,t) \in T^* \ \& \ q` \subseteq y^{-1}(M) \ \& \ \forall t` \ (s,x,p,y,q`,t`) \notin T^*\}.$$

В некотором смысле неявный переход вида $(s, x, \varnothing, J \times \{\Lambda\}, \varnothing, s)$ моделирует отсутствие перехода, поскольку состояние автомата не меняется, сообщения не принимаются и не передаются.

Заметим, что, если в определении автомата используются правила умолчания, то композиция автоматов должна определяться через композицию не только явных, но и неявных переходов. Это показано в примере на Рис. 1.



$A = (\{m\}, \{k\}, \{j\}, \{s_{A0}\}, T^*_A, \{s_{A0},s_{A1}\})$

$T_A=T^*_A=\{(s_{A0},\{(k,\Lambda)\},\varnothing,\{(j,\Lambda)\},\varnothing,s_{A0}), (s_{A0},\{(k,m)\},\{k\},\{(j,\Lambda)\},\varnothing,s_{A1})\}$

$B = (\{m\}, \{i\}, \varnothing, \{s_{B0}\}, T^*_B, s_{B0}\}$

$T^*_B=\varnothing$, $T_B=\{(s_{B0},\{(i,\Lambda)\},\varnothing,\varnothing,\varnothing,s_{B0}), (s_{B0},\{(i,m)\},\varnothing,\varnothing,\varnothing,s_{B0})\}$

$T_A[j,i]T_B = \{(s_{A0}\cup s_{B0},\{(k,\Lambda)\},\varnothing,\varnothing,\varnothing,s_{A0}\cup s_{B0}), (s_{A0}\cup s_{B0},\{(k,m)\},\{k\},\varnothing,\varnothing,s_{A1}\cup s_{B0})\} = T_{A[j,i]B}$

$T^*_A[j,i]T^*_B = \varnothing$, а правила умолчания добавляют множество переходов:

$\{(s_{A0}\cup s_{B0},\{(k,\Lambda)\},\varnothing,\varnothing,\varnothing,s_{A0}\cup s_{B0}), (s_{A0}\cup s_{B0},\{(k,m)\},\varnothing,\varnothing,\varnothing,s_{A0}\cup s_{B0})\} \neq T_{A[j,i]B}$

Рис. 1. Композиция и неявные переходы
Fig 1. Composition and implicit transitions

В этом примере автомат B не имеет явных переходов, поэтому все его переходы неявные и являются петлями в начальном состоянии s_{B0}. Из-за этого композиция $T^*_A[j,i]T^*_B$ множеств явных переходов пуста, а после её пополнения по правилам умолчания все добавляемые переходы будут петлями в начальном состоянии $s_{A0}\cup s_{B0}$. В то же время композиция $T_A[j,i]T_B$ множеств всех переходов (как явных, так и неявных) не требует пополнения по правилам умолчания, но содержит переход по стимулу $x=\{(k,m)\}$, который ведёт в состояние $s_{A1}\cup s_{B0}$ отличное от начального.

Для целей тестирования мы ограничимся исследованием детерминированных автоматов. Детерминированность будет означать, что каждый автомат в системе в своём текущем состоянии может выполнить не более одного перехода. При условии вполне определенности будет ровно один переход. Для детерминизма, прежде всего, нужно, чтобы в каждом состоянии автомата каждый стимул x однозначно определял параметр p, то есть по какому входу i сообщение $x(i)$ принимается, а по какому нет. Стимул определяется сообщениями на входах автомата, а они, в свою очередь, определяются автоматами, посылающими сообщения на входы данного автомата, то есть автоматами, выходы которых соединены со входами данного автомата. Кроме того, стимул x однозначно определяет реакцию y. При этом параметр q,

определяющий по какому выходу j сообщение $y(j)$ передаётся, а по какому нет, может выбираться любой, но зато однозначно определяется автоматами, принимающими сообщения с выходов данного автомата, то есть автоматами, входы которых соединены с выходами данного автомата. Наконец, постсостояние t должно быть единственным, если определены остальные параметры перехода: s, x, p, y и q. Эти требования формализованы ниже в определении детерминированного автомата.

Автомат $A=(M,I,J,S,T,s_0)$ будем называть *детерминированным*, если выполнены условия:

1. Состояние s и стимул x однозначно определяют приём сообщений p и реакцию y:
   $\forall a,b \in T_A\ (s_a=s_b \ \&\ x_a=x_b \Rightarrow p_a=p_b \ \&\ y_a=y_b)$.
2. Одинаково помеченные переходы, ведущие из одного состояния, совпадают, т.е. ведут в одно постсостояние:
   $\forall a,b \in T_A\ (s_a=s_b \ \&\ x_a=x_b \ \&\ p_a=p_b \ \&\ y_a=y_b \ \&\ q_a=q_b \Rightarrow t_a=t_b)$.

Заметим, что если выполнено условие 1, условие 2 можно переписать короче:

$\forall a,b \in T_A\ (s_a=s_b \ \&\ x_a=x_b \ \&\ q_a=q_b \Rightarrow t_a=t_b)$.

В то же время детерминированность каждого автомата системы не гарантирует детерминированность системы: без дополнительных условий автоматы в некотором цикле соединений могут выполнять либо один, либо другой набор переходов, выбор между которыми недетерминирован. Выяснению достаточных условий детерминизма системы как раз и посвящена основная часть данной статьи.

Пусть задана система автоматов $\boldsymbol{R}=(M,V,E)$.

Лемма 1 о композиции разных автоматов:

Композиция различных вполне определённых и детерминированных автоматов вполне определённая и детерминированная.

Доказательство: Пусть автоматы $A,B \in V$ вполне определённые, детерминированные и различные $A \neq B$, а соединение $(j,i) \in E$ ведёт из A в B. Надо доказать, что автомат $A[j,i]B$ вполне определённый и детерминированный.

1. Докажем, что автомат $A[j,i]B$ вполне определённый.

1.1. Докажем выполнение 1-го условия вполне определённости (определенность по всем стимулам):

$\forall s \in S_{A[j,i]B}\ \ \forall x \in X_{A[j,i]B}\ \exists c \in T_{A[j,i]B}\ s_c=s \ \&\ x_c=x$.

Пусть $s \in S_{A[j,i]B}$ и $x \in X_{A[j,i]B}$. Тогда найдутся такие состояния $s_A \in S_A$ и $s_B \in S_B$, что $s=s_A \cup s_B$. Поскольку $\boldsymbol{Dom}(x)=I_{A[j,i]B}=I_A \cup I_B \backslash \{i\}$ и $i \in I_B$, стимул x может быть представлен в виде объединения непересекающихся по доменам отображений $x=x_A \cup x_B$, где $\boldsymbol{Dom}(x_A)=I_A$ и $\boldsymbol{Dom}(x_B)=I_B \backslash \{i\}$. Поскольку автомат A вполне определённый, по условию 1 в нём есть переход по каждому стимулу в каждом состоянии. Следовательно, имеется переход a

в состоянии s_A по стимулу x_A, т.е. $s_a=s_A$ и $x_a=x_A$. Объединение $x`_B=x_B\cup\{(i,y_a(j))\}$ является стимулом вполне определённого автомата $B$, поэтому по условию 1 имеется переход $b$ в состоянии $s_B$ по стимулу $x`_B$, т.е. $s_b=s_B$ и $x_b=x`_B$. Поскольку автомат $A$ вполне определённый, по условию 2 в нём имеются переходы по всем $q$, следовательно, найдётся в состоянии $s_a$ такой переход $a`$, что $s_{a`}=s_a$, $x_{a`}=x_a$ и $j\in q_{a`} \sim i\in p_b$. Тогда $f(a`,j,i,b)=\mathbf{true}$. Следовательно, в автомате $A[j,i]B$ имеется в состоянии $s_{a`}\cup s_b$ переход $c=a`[j,i]b$ по стимулу $x_{a`}\cup x_b/\{i\}$. Поскольку $s=s_A\cup s_B$, $s_a=s_A$, $s_{a`}=s_a$, $s_b=s_B$, имеем $s_c= s_{a`}\cup s_b=s$. Поскольку $x=x_A\cup x_B$, $x`_B=x_B\cup\{(i,y_a(j))\}$, $x_b=x`_B$, $x_a=x_A$, $x_{a`}=x_a$, имеем $x_c= x_{a`}\cup x_b/\{i\}=x$. Тем самым, в автомате $A[j,i]B$ имеется в состоянии s переход c по стимулу x, что и требовалось доказать.

1.2. Докажем выполнение 2-го условия вполне определённости:

$\forall c\in T_{A[j,i]B}\ \forall q`\subseteq y_c^{-1}(M)\ \exists t`\ (s_c, x_c, p_c, y_c, q`, t`)\in T_{A[j,i]B}$.

Пусть $c\in T_{A[j,i]B}$ и $q`\subseteq y_c^{-1}(M)$. По определению композиции переход $c$ является композицией двух переходов $c=a[j,i]b$, где $a\in T_A$ и $b\in T_B$. Тогда $y_c=y_a\cup y_b\backslash\{j\}$. Поскольку $\mathbf{Dom}(y_a)=J_A$ и $\mathbf{Dom}(y_b)=J_B$, а $q`\subseteq y_c^{-1}(M)$, множество $q`$ можно представить в виде объединения двух множеств $q`=q_A\cup q_B$, где $q_A\subseteq y_a^{-1}(M)\backslash\{j\}$ и $q_B\subseteq y_b^{-1}(M)$. Обозначим $q`_A=q_A$, если $i\notin p_b$, и $q`_A=q_A\cup\{j\}$, если $i\in p_b$. Поскольку автоматы A и B вполне определённые по условию 2 найдутся такие состояния $t_A\in S_A$ и $t_B\in S_B$, что $a`=(s_a,x_a,p_a,y_a,q`_A,t_A)\in T_A$ и $b`=(s_b,x_b,p_b,y_b,q_B,t_B)\in T_B$. Имеем $y_a(j)=y_a(j)=x_b(i)=x_{b`}(i)$ и $j\in q_{a`} \sim j\in q`_A \sim i\in p_b \sim i\in p_{b`}$. Следовательно, $f(a`,j,i,b`)=\mathbf{true}$. Также имеем $q`=q_A\cup q_B=q`_A\cup q_B\backslash\{j\}$. Поэтому, обозначая $c`=a`[j,i]b`$, имеем $c`=(s_c,x_c,p_c,y_c,q`,t_A\cup t_B)\in T_{A[j,i]B}$, т.е. в композиции нашлось такое состояние $t`=t_A\cup t_B$, что $(s_c, x_c, p_c, y_c, q`, t`)\in T_{A[j,i]B}$, что и требовалось доказать.

2. Докажем, что автомат $A[j,i]B$ детерминированный.

По определению композиции автоматов переход в автомате $A[j,i]B$ получается в результате композиции двух переходов в автоматах A и B. Пусть $a,a`\in T_A$, $b,b`\in T_B$ и переход a компонуется с переходом b, а переход $a`$ – с переходом $b`$, т.е. $f(a,j,i,b)=f(a`,j,i,b`)=\mathbf{true}$. Докажем выполнение условий детерминизма для переходов $a[j,i]b$ и $a`[j,i]b`$. По определению композиции переходов для $a,a`\in T_A$, $b,b`\in T_B$ и $A\neq B$ имеем:

$f(a,j,i,b) \Rightarrow y_a(j)=x_b(i)\ \&\ (j\in q_a \sim i\in p_b)$,

$f(a`,j,i,b`) \Rightarrow y_{a`}(j)=x_{b`}(i)\ \&\ (j\in q_{a`} \sim i\in p_{b`})$,

$a[j,i]b = (s_a\cup s_b, x_a\cup x_b/\{i\}, p_a\cup p_b\backslash\{i\}, y_a\cup y_b/\{j\}, q_a\cup q_b\backslash\{j\}, t_a\cup t_b)$ и

$a`[j,i]b` = (s_{a`}\cup s_{b`}, x_{a`}\cup x_b/\{i\}, p_{a`}\cup p_{b`}\backslash\{i\}, y_{a`}\cup y_b/\{j\}, q_{a`}\cup q_{b`}\backslash\{j\}, t_{a`}\cup t_{b`})$.

Пусть $s_a\cup s_b=s_{a`}\cup s_{b`}$ и $x_a\cup x_b/\{i\}=x_{a`}\cup x_b/\{i\}$.

2.1. Докажем выполнение 1-го условия детерминизма:

$p_a \cup p_b \backslash \{i\} = p_{a'} \cup p_{b'} \backslash \{i\}$ и $y_a \cup y_b / \{j\} = y_{a'} \cup y_{b'} / \{j\}$.

$A \neq B$ влечёт $s_{a'} \cap s_b = s_a \cap s_{b'} = \varnothing$. Поэтому, $s_a \cup s_b = s_{a'} \cup s_{b'}$ влечёт $s_a = s_{a'}$ и $s_b = s_{b'}$. $A \neq B$ влечёт $I_A \cap I_B = \varnothing$, что влечёт $x_{a'} \cap x_b = x_a \cap x_{b'} = \varnothing$. Также $i \in I_B$. Поэтому $x_a \cup x_b / \{i\} = x_{a'} \cup x_{b'} / \{i\}$ влечёт $x_a = x_{a'}$ и $x_b / \{i\} = x_{b'} / \{i\}$. По 1-ому условию детерминизма для A из $s_a = s_{a'}$ и $x_a = x_{a'}$ следует $y_a = y_{a'}$ и $p_a = p_{a'}$. Поскольку $y_a(j) = x_b(i)$ и $y_{a'}(j) = x_{b'}(i)$, а $y_a = y_{a'}$ влечёт $y_a(j) = y_{a'}(j)$, имеем $x_b(i) = x_{b'}(i)$. А тогда, поскольку $x_b / \{i\} = x_{b'} / \{i\}$, имеем $x_b = x_{b'}$. По 1-ому условию детерминизма для B из $s_b = s_{b'}$ и $x_b = x_{b'}$ следует $y_b = y_{b'}$ и $p_b = p_{b'}$. Итак, $s_a = s_{a'}$, $s_b = s_{b'}$, $x_a = x_{a'}$, $x_b = x_{b'}$, $p_a = p_{a'}$, $p_b = p_{b'}$, $y_a = y_{a'}$, $y_b = y_{b'}$. А тогда $p_a \cup p_b \backslash \{i\} = p_{a'} \cup p_{b'} \backslash \{i\}$ и $y_a \cup y_b / \{j\} = y_{a'} \cup y_{b'} / \{j\}$, что и требовалось доказать.

2.2. Пусть также $q_a \cup q_b / \{j\} = q_{a'} \cup q_{b'} / \{j\}$. Докажем выполнение 2-го условия детерминизма: $t_a \cup t_b = t_{a'} \cup t_{b'}$.

По доказанному $s_a = s_{a'}$, $s_b = s_{b'}$, $x_a = x_{a'}$, $x_b = x_{b'}$, $p_a = p_{a'}$, $p_b = p_{b'}$, $y_a = y_{a'}$, $y_b = y_{b'}$. $A \neq B$ влечёт $J_A \cap J_B = \varnothing$, что влечёт $q_{a'} \cap q_b = q_a \cap q_{b'} = \varnothing$. Также $j \in J_A$. Поэтому $q_a \cup q_b / \{j\} = q_{a'} \cup q_{b'} / \{j\}$ влечёт $q_a \backslash \{j\} = q_{a'} \backslash \{j\}$ и $q_b = q_{b'}$. Поскольку $j \in q_a \sim i \in p_b$ и $j \in q_{a'} \sim i \in p_{b'}$, а $p_b = p_{b'}$, имеем $j \in q_a \sim j \in q_{a'}$. А поскольку $q_a \backslash \{j\} = q_{a'} \backslash \{j\}$, имеем $q_a = q_{a'}$. Итак, $s_a = s_{a'}$, $s_b = s_{b'}$, $x_a = x_{a'}$, $x_b = x_{b'}$, $p_a = p_{a'}$, $p_b = p_{b'}$, $y_a = y_{a'}$, $y_b = y_{b'}$, $q_a = q_{a'}$, $q_b = q_{b'}$. А тогда по 2-ому условию детерминизма для A и для B имеем $t_a = t_{a'}$ и $t_b = t_{b'}$. А тогда $t_a \cup t_b = t_{a'} \cup t_{b'}$, что и требовалось доказать.

Лемма 1 доказана.

Пару *(k,l)* будем называть *контактом* внутри автомата A, если k вход, а l выход автомата A: $l \in J_A \,\&\, k \in I_A$. Контакт *(k,l)* будем называть *свободным* и обозначать $k \# l$, если сообщение на выходе l не зависит от сообщения на входе k:

$$k \# l \triangleq \exists A \in V \; l \in J_A \,\&\, k \in I_A \,\&\, \forall a,b \in T_A \; (s_a = s_b \,\&\, x_a / \{k\} = x_b / \{k\} \Rightarrow y_a(l) = y_b(l)).$$

Очевидно, что свободность контакта внутри автомата является свойством самого этого автомата независимо от того, в какую систему он входит. Однако принадлежность выхода и входа тому или иному автомату зависит от системы и меняется при композиции системы, поскольку меняется множество автоматов системы. Если $l \in J_A$, $k \in I_A$ и выполняется композиция автоматов $A[j,i]B$ или $B[j,i]A$, то выход l остаётся в системе, только если $l \neq j$, вход k остаётся в системе, только если $k \neq i$. Кроме того, если $l \neq j$ и $k \neq i$, то после композиции выход l и вход k будут принадлежать уже другому, композиционному автомату $A[j,i]B$ или $B[j,i]A$. Поэтому, если речь идёт о системе $\boldsymbol{R}$` отличной от подразумеваемой системы $\boldsymbol{R}$, мы в обозначении будем указывать систему в индексе: $k \#_{\boldsymbol{R}'} l$.

Лемма 2 о композиции по петле:

Если автомат вполне определённый и детерминированный, внутри автомата контакт $i \# j$ и в системе есть соединение-петля *(j,i)*, то композиция этого автомата по этому соединению вполне определённая и детерминированная.

Доказательство: Пусть соединение $(j,i)\in E$ является петлёй, т.е. $\varphi(j)=\varphi(i)$, контакт $i \# j$, и автомат $A=\varphi(j)$ вполне определённый и детерминированный.

1. Докажем, что автомат $A[j,i]A$ вполне определённый.

1.1. Докажем выполнение 1-го условия вполне определённости (определенность по всем стимулам):

$\forall s\in S_{A[j,i]A}\ \forall x\in X_{A[j,i]A}\ \exists c\in T_{A[j,i]A}\ s_c=s\ \&\ x_c=x.$

Пусть $s\in S_{A[j,i]A}$ и $x\in X_{A[j,i]A}$. Тогда $s\in S_A$. Поскольку $\mathbf{Dom}(x)=I_{A[j,i]A}=I_A\backslash\{i\}$, для любого сообщения $m\in M_A$ объединение $x`=x\cup\{(i,m)\}$ является стимулом для автомата $A$. Поскольку автомат $A$ вполне определённый, по условию 1 в нём есть переход по каждому стимулу в каждом состоянии. Следовательно, имеется переход $a$ в состоянии $s$ по стимулу $x`$, т.е. $s_a=s$ и $x_a=x`$. Тогда $x``=x\cup\{(i,y_a(j))\}$ тоже является стимулом для автомата $A$. Поскольку автомат $A$ вполне определённый, по условию 1 в нём есть переход $a`$ в состоянии $s$ по стимулу $x``$, т.е. $s_{a`}=s$ и $x_{a`}=x``$. Очевидно, что $x_{a`}(i)=y_a(j)$ и $x_{a`}\backslash\{i\}=x_a\backslash\{i\}$. А тогда, поскольку $i\#j$, имеем $x_{a`}(i)=y_a(j)=y_{a`}(j)$, что влечёт $x_{a`}(i)=y_{a`}(j)$. Поскольку автомат A вполне определённый, по условию 2 в нём имеются переходы по всем q, следовательно, найдётся в состоянии s переход $a``=(s_{a`}, x_{a`}, p_{a`}, y_{a`}, q_{a``}, t_{a``})$, где $q_{a``}\backslash\{i\}=q_{a`}\backslash\{i\}$ и $j\in q_{a``} \sim i\in p_{a``}$. Тогда $f(a``,j,i,a``)=\mathbf{true}$. Следовательно, в автомате $A[j,i]A$ имеется в состоянии s переход $c=a``[j,i]a``$ по стимулу $x_{a`}\backslash\{i\}$. Поскольку $s_{a``}=s_{a`}=s$, имеем $s_c=s$. Поскольку $x_{a``}=x_{a`}=x``=x\cup\{(i,y_a(j))\}$, имеем $x_c=x_{a`}\backslash\{i\}=x$. Тем самым, в автомате $A[j,i]A$ имеется в состоянии s переход c по стимулу x, что и требовалось доказать.

1.2. Докажем выполнение 2-го условия вполне определённости:

$\forall c\in T_{A[j,i]A}\ \forall q`\subseteq y_c^{-1}(M)\ \exists t`\ (s_c, x_c, p_c, y_c, q`, t`)\in T_{A[j,i]A}.$

Пусть $c\in T_{A[j,i]A}$ и $q`\subseteq y_c^{-1}(M)$. По определению композиции переход c является композицией одного перехода $c=a[j,i]a$, где $a\in T_A$. Тогда $x_a(i)=y_a(j)$, $j\in q_a \sim i\in p_a$ и $y_c=y_a\backslash\{j\}$, $s_c=s_a$. Обозначим $q``=q`$, если $i\notin p_a$, и $q``=q`\cup\{j\}$, если $i\in p_a$. Поскольку автомат $A$ вполне определённый, по условию 2 найдется переход $a`=(s_c, x_a, p_a, y_a, q``, t_{a`})\in T_A$. Имеем $y_{a`}(j)=y_a(j)=x_a(i)=x_{a`}(i)$ и $j\in q_{a`} \sim j\in q`` \sim i\in p_a \sim i\in p_{a`}$. Следовательно, $f(a`,j,i,a`)=\mathbf{true}$. Также имеем $q`=q``\backslash\{j\}=q_{a`}\backslash\{j\}$. Поэтому, обозначая $c`=a`[j,i]a`$, имеем $c`=(s_c,x_c,p_c,y_c,q`,t_{a`})\in T_{A[j,i]A}$, т.е. в композиции нашлось такое состояние $t`=t_{a`}$, что $(s_c, x_c, p_c, y_c, q`, t`)\in T_{A[j,i]B}$, что и требовалось доказать.

2. Докажем, что автомат $A[j,i]A$ детерминированный.

По определению композиции автоматов переход в автомате $A[j,i]A$ получается в результате композиции одного перехода в автомате A. Пусть $a,b\in T_A$ и каждый из переходов a и b компонуется сам с собой, т.е. $f(a,j,i,a)=f(b,j,i,b)=\mathbf{true}$. Надо доказать выполнение обоих условий

детерминизма для переходов $a[j,i]a$ и $b[j,i]b$. По определению композиции переходов для $a,b \in T_A$ имеем:

$f(a,j,i,a) \Rightarrow y_a(j)=x_a(i)$ & $(j \in q_a \sim i \in p_a)$,

$f(b,j,i,b) \Rightarrow y_b(j)=x_b(i)$ & $(j \in q_b \sim i \in p_b)$,

$a[j,i]a = (s_a, x_a/\{i\}, p_a\backslash\{i\}, y_a/\{j\}, q_a\backslash\{j\}, t_a)$ и

$b[j,i]b = (s_b, x_b/\{i\}, p_b\backslash\{i\}, y_b/\{j\}, q_b\backslash\{j\}, t_b)$.

Пусть $s_a=s_b$ и $x_a/\{i\}=x_b/\{i\}$.

2.1. Докажем выполнение 1-го условия детерминизма:

$p_a\backslash\{i\}=p_b\backslash\{i\}$ и $y_a/\{j\}=y_b/\{j\}$.

Поскольку $i \# j$, из $s_a=s_b$ и $x_a/\{i\}=x_b/\{i\}$ следует $y_a(j)=y_b(j)$. А тогда, поскольку $y_a(j)=x_a(i)$ и $y_b(j)=x_b(i)$, имеем $x_a(i)=x_b(i)$. Отсюда, учитывая, что $x_a/\{i\}=x_b/\{i\}$, имеем $x_a=x_b$. Из $s_a=s_b$ и $x_a=x_b$ по 1-ому условию детерминизма для автомата A имеем $y_a=y_b$ и $p_a=p_b$. Итак, $s_a=s_b$, $x_a=x_b$, $p_a=p_b$, $y_a=y_b$. А тогда $p_a\backslash\{i\}=p_b\backslash\{i\}$ и $y_a/\{j\}=y_b/\{j\}$, что и требовалось доказать.

2.2. Пусть также $q_a\backslash\{j\}=q_b\backslash\{j\}$. Докажем выполнение 2-го условия детерминизма: $t_a=t_b$.

По доказанному имеем $s_a=s_b$, $x_a=x_b$, $p_a=p_b$, $y_a=y_b$. Поскольку $j \in q_a \sim i \in p_a$ и $j \in q_b \sim i \in p_b$, а $p_a=p_b$, имеем $j \in q_a \sim j \in q_b$. А поскольку $q_a\backslash\{j\}=q_b\backslash\{j\}$, имеем $q_a=q_b$. Итак, $s_a=s_b$, $x_a=x_b$, $p_a=p_b$, $y_a=y_b$, $q_a=q_b$. А тогда по 2-ому условию детерминизма для автомата A имеем $t_a=t_b$, что и требовалось доказать.

Лемма 2 доказана.

Лемма 3 о контакте внутри «постороннего» автомата: При композиции системы $\boldsymbol{R}$ по соединению $(j,i) \in E$ сохраняется свободность контактов внутри любого автомата C, который не участвует в композиции, т.е. $C \neq \varphi(j)$ и $C \neq \varphi(i)$.

Доказательство: Пусть $(j,i) \in E$, $C \in V$, $C \neq \varphi(j)$, $C \neq \varphi(i)$, $l \in J_C$, $k \in I_C$ и $k \# l$. После композиции $C \in V_{\boldsymbol{R}[j,i]}$, $l \in J_C$, $k \in I_C$ и, поскольку свободность контактов внутри автомата является свойством самого этого автомата независимо от системы, в системе $\boldsymbol{R}[j,i]$ сохраняется свободность контакта $k \#_{\boldsymbol{R}[j,i]} l$.

Лемма 3 доказана.

Лемма 4 о контакте в композиции разных автоматов: При композиции различных детерминированных автоматов по соединению $(j,i) \in E$ сохраняется свободность контакта (k,l), где $l \neq j$ и $k \neq i$.

Доказательство: Пусть $A=\varphi(j)$, $B=\varphi(i)$ и $A \neq B$. Пусть также автомат $C \in V$, $l \in J_C$, $k \in I_C$, $l \neq j$, $k \neq i$ и $k \# l$. Надо доказать, что $k \#_{\boldsymbol{R}[j,i]} l$. Если $C \neq A$ и $C \neq B$, то по лемме 3 $k \#_{\boldsymbol{R}[j,i]} l$. Рассмотрим остальные случаи (Рис. 2): 1) $C=A \neq B$, 2) $C=B \neq A$. Для этих случаев рассмотрим два перехода в композиции, которые получаются как композиции двух пар переходов a,b и $a`,b`$, где $a,a` \in T_A$ и $b,b` \in T_B$:

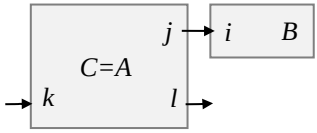
$f(a,j,i,b) \Rightarrow y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$, $f(a`,j,i,b`) \Rightarrow y_{a`}(j)=x_{b`}(i)$ & $(j \in q_{a`} \sim i \in p_{b`})$,

$a[j,i]b = (s_a \cup s_b, x_a \cup x_b/\{i\}, p_a \cup p_b\backslash\{i\}, y_a \cup y_b/\{j\}, q_a \cup q_b\backslash\{j\}, t_a \cup t_b)$ и

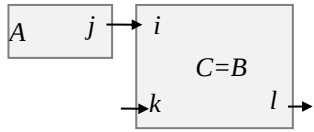
$a`[j,i]b` = (s_{a`} \cup s_{b`}, x_{a`} \cup x_b/\{i\}, p_{a`} \cup p_{b`} \setminus \{i\}, y_{a`} \cup y_b/\{j\}, q_{a`} \cup q_{b`} \setminus \{j\}, t_{a`} \cup t_{b`})$
такие, что $s_a \cup s_b = s_{a`} \cup s_{b`}$ и $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_b/\{i\}/\{k\}$.

Надо доказать, что $(y_a \cup y_b/\{j\})(l) = (y_{a`} \cup y_b/\{j\})(l)$.

$A \neq B$ влечёт $s_{a`} \cap s_b = s_a \cap s_{b`} = \varnothing$. Поэтому $s_a \cup s_b = s_{a`} \cup s_{b`}$ влечёт $s_a = s_{a`}$ и $s_b = s_{b`}$.



1) $C=A \neq B$



2) $C=B \neq A$

Рис. 2. Свободность контакта при композиции разных автоматов
Fig. 2. Free contact with a composition of different machines

1. $C=A \neq B$.

   Поскольку $I_A \cap I_B = \varnothing$, $k \in I_A$, $i \in I_B$, имеем $x_a \cup x_b/\{i\}/\{k\} = (x_a/\{k\}) \cup (x_b/\{i\})$.

   Аналогично, $x_{a`} \cup x_b/\{i\}/\{k\} = (x_{a`}/\{k\}) \cup (x_b/\{i\})$.

   Следовательно, поскольку $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_b/\{i\}/\{k\}$, имеем $(x_a/\{k\}) \cup (x_b/\{i\}) = (x_{a`}/\{k\}) \cup (x_b/\{i\})$.

   Отсюда, поскольку $I_A \cap I_B = \varnothing$, имеем $x_a/\{k\} = x_{a`}/\{k\}$.

   Поскольку $J_A \cap J_B = \varnothing$, $l \in J_A$, $j \in J_A$, $l \neq j$, имеем $(y_a \cup y_b/\{j\})(l) = (y_a/\{j\})(l) = y_a(l)$.

   Аналогично, $(y_{a`} \cup y_b/\{j\})(l) = y_{a`}(l)$.

   Поскольку $k \# l$, из $s_a = s_{a`}$ и $x_a/\{k\} = x_{a`}/\{k\}$ следует $y_a(l) = y_{a`}(l)$.

   Следовательно, $(y_a \cup y_b/\{j\})(l) = (y_{a`} \cup y_b/\{j\})(l)$, что и требовалось доказать.

2. $C=B \neq A$.

   Поскольку $I_A \cap I_B = \varnothing$, $k \in I_B$ и $i \in I_B$, имеем $x_a \cup x_b/\{i\}/\{k\} = x_a \cup (x_b/\{i\}/\{k\})$.

   Аналогично $x_{a`} \cup x_b/\{i\}/\{k\} = x_{a`} \cup (x_b/\{i\}/\{k\})$.

   Следовательно, поскольку $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_b/\{i\}/\{k\}$, имеем $x_a \cup (x_b/\{i\}/\{k\}) = x_{a`} \cup (x_b/\{i\}/\{k\})$.

   Отсюда, поскольку $I_A \cap I_B = \varnothing$, имеем $x_a = x_{a`}$ и $x_b/\{i\}/\{k\} = x_b/\{i\}/\{k\}$.

   Поскольку автомат A детерминированный, из $s_a = s_{a`}$ и $x_a = x_{a`}$ следует $y_a = y_{a`}$.

   А тогда, поскольку $y_a(j) = x_b(i)$ и $y_{a`}(j) = x_{b`}(i)$, имеем $x_b(i) = x_{b`}(i)$.

   Отсюда, поскольку $x_b/\{i\}/\{k\} = x_{b`}/\{i\}/\{k\}$, имеем $x_b/\{k\} = x_{b`}/\{k\}$.

   Поскольку $J_A \cap J_B = \varnothing$, $l \in J_B$ и $j \in J_A$, имеем $(y_a \cup y_b/\{j\})(l) = y_b(l)$.

   Аналогично $(y_{a`} \cup y_b/\{j\})(l) = y_{b`}(l)$.

   Поскольку $k \# l$, из $s_b = s_{b`}$ и $x_b/\{k\} = x_{b`}/\{k\}$ следует $y_b(l) = y_{b`}(l)$.

   Следовательно, $(y_a \cup y_b/\{j\})(l) = (y_{a`} \cup y_b/\{j\})(l)$, что и требовалось доказать.

Лемма 4 доказана.

Будем говорить, что соединение (j,i) *смежно* с соединением $(j`,i`)$, если в графе $\boldsymbol{Og(R)}$, дуга (j,i) смежна с дугой $(j`,i`)$, т.е. вход первого соединения и выход второго соединения образуют контакт (принадлежат одному и тому же автомату): $\varphi(i)=\varphi(j`)$. *Маршрутом* длины m в системе $\boldsymbol{R}$ назовём ориентированный маршрут длины m в графе $\boldsymbol{Og(R)}$, т.е. последовательность соединений $Z=(j_0,i_1),(j_1,i_2),...(j_{m-1},i_m)$, в которой каждое соединение, кроме последнего, смежно со следующим соединением: $\forall r=1..m-1\ \varphi(i_r)=\varphi(j_r)$. Контакт (i_r,j_r) для $r=1..m-1$ будем называть *контактом в маршруте* Z. Маршрут будем называть *путём*, если в нём все соединения разные: $\forall r,r`=0..m-1\ (r\neq r` \Rightarrow (j_r,i_{r+1})\neq(j_{r`},i_{r`+1}))$. Путь будем называть *циклом*, если $\varphi(i_m)=\varphi(j_0)$, т.е. последнее соединение смежно с первым. В этом случае будем обозначать $i_0=i_m$ и пару $(i_0,j_0)=(i_m,j_0)$ также будем называть контактом в цикле Z. Цикл Z будем называть A-*простым*, если $A=\varphi(j_0)$ и $A\neq\varphi(j_r)$ для $r=1..m-1$. Если в цикле Z имеет место $\varphi(j_r)=\varphi(i_{r`})$, то через $Z[j_r,i_{r`}]$ обозначим цикл $(j_r,i_{r+1}),...(j_{r`-1},i_{r`})$, если $r < r`$, и цикл $(j_r,i_{r+1}),...(j_{m-1},i_m)\ (j_0,i_1),...(j_{r`-1},i_{r`})$, если $r \geq r`$. Цикл будем называть *свободным*, если в цикле есть свободный контакт, т.е. $i_r \# j_r$ для некоторого $r=0..m-1$.

<u>Лемма 5 о сохранении свободности циклов при композиции</u>:

Если в системе $\boldsymbol{R}$ все автоматы вполне определённые и детерминированные, все циклы свободные, а $(j,i)\in E$, то в системе $\boldsymbol{R}[j,i]$ все циклы свободные.

<u>Доказательство</u>: Обозначим компонуемые автоматы $A=\varphi(j)$ и $B=\varphi(i)$. Рассмотрим в $\boldsymbol{R}[j,i]$ цикл $Z=(j_0,i_1),...(j_{m-1},i_m)$. По определению композиции системы $E_{\boldsymbol{R}[j,i]}=E\backslash\{(j,i)\}$, поэтому все соединения из Z были и в системе $\boldsymbol{R}$. Также по определению композиции системы $V_{\boldsymbol{R}[j,i]}=V\backslash\{A,B\}\cup\{A[j,i]B\}$. Смежность соединений из Z может отсутствовать в системе $\boldsymbol{R}$ только в том случае, когда в $\boldsymbol{R}[j,i]$ в цикле Z есть контакт (k,l) внутри автомата $A[j,i]B$, но в системе $\boldsymbol{R}$ вход k и выход l относятся к разным автоматам: 1) $A\neq B$, $k\in I_B$ и $l\in J_A$, или 2) $A\neq B$, $k\in I_A$ и $l\in J_B$. За исключением этого случая Z является циклом в $\boldsymbol{R}$.

1. Z является циклом в $\boldsymbol{R}$.

   По условию леммы цикл Z в $\boldsymbol{R}$ свободный. Тогда в $\boldsymbol{R}$ в цикле Z существует хотя бы один свободный контакт. Покажем, что хотя бы для одного такого контакта $k \# l$ имеет место $k \#_{\boldsymbol{R}[j,i]} l$.

1.1. $A\neq B$.

   Пусть $k \# l$ контакт в $\boldsymbol{R}$ в цикле Z. Поскольку Z является циклом в $\boldsymbol{R}[j,i]$, а $(j,i)\notin E_{\boldsymbol{R}[j,i]}$, имеем $l\neq j$ и $k\neq i$. А тогда по лемме 4 $k \#_{\boldsymbol{R}[j,i]} l$.

1.2. $A=B$ и в $\boldsymbol{R}$ в цикле Z есть контакт $k \# l$ внутри автомата, отличного от A, т.е. $k\notin I_A$ и, следовательно, $l\notin J_A$.

   По лемме 3 $k \#_{\boldsymbol{R}[j,i]} l$.

1.3. $A=B$ и в $\boldsymbol{R}$ в цикле Z любой свободный контакт является контактом внутри автомата A.

Сначала докажем вспомогательное утверждение: В системе ***R*** существует контакт *a # b* внутри автомата *A* в цикле *Z*, для которого *i # b* или *a # j*.

Так как все циклы в ***R*** свободные, а в *Z* свободный контакт может быть только внутри автомата *A*, существует контакт *a # b* внутри автомата *A* в цикле *Z*.

Если $Z=(j_0,i_1),...(j_{m-1},i_m)$, то $a=i_r$ и $b=j_r$ для некоторого $r=0..m-1$. Последовательность $Z`=(b=j_r,i_{r+1}),...(j_{m-1},i_m)(j_0,i_1),...(j_{r-1},i_r=a)$, очевидно, тоже является циклом, в котором все соединения и контакты такие же, как в цикле *Z*.

Цикл *Z`* можно представить в виде конкатенации *A*-простых циклов:

$W = Z`[b=l_1,k_2]\cdot Z`[l_2,k_3]\cdot...\cdot Z`[l_{w-2},k_{w-1}]\cdot Z`[l_{w-1},k_w=a]$ (Рис. 3).

Выберем в *W* максимальный начальный отрезок $W` = Z`[b=l_1,k_2]\cdot...\cdot Z`[l_{w`-1},k_{w`}=c]$ без свободных контактов. В цикле *Z* для некоторого выхода *d* существует контакт *(c,d)*, который по условию максимальности *W`* свободный.

Рассмотрим цикл $W`` = Z`[b=l_1,k_2]\cdot...\cdot Z`[l_{w`-1},k_{w`}=c]\cdot(j,i)$, который получается из *W`* добавлением соединения-петли *(j,i)*. В нём нет контактов *(a,b)* и *(c,d)*, но есть контакты *(i,b)* и *(c,j)*. В цикле *W``* все контакты не внутри автомата *A* не свободные, поскольку эти контакты есть также в цикле *Z`* и, следовательно, в *Z*, а в *Z* все контакты не внутри *A* не свободные. В цикле *W``* внутри *A* все контакты, кроме, быть может, *(i,b)* и *(c,j)* тоже не свободные по построению *W`* и *W``*. Если бы вспомогательное утверждение было не верно, то, поскольку контакты *a # b* и *c # d*, контакты *(i,b)* и *(c,j)* были бы не свободные. Но тогда в цикле *W``* вообще не было бы свободных контактов, что противоречит тому, что в ***R*** все циклы свободные. Следовательно, вспомогательное утверждение верно.

Теперь докажем основное утверждение пункта 1.3.

Поскольку *A=B*, соединение *(j,i)* является петлёй в ***R***, и в ней есть единственный контакт *(i,j)*. По условию леммы петля (как цикл) свободная, следовательно, *i # j*.

Поскольку *k # l*, по вспомогательному утверждению *i # l* или *k # j*.

Итак, мы имеем: *k # l & i # j & (i # l ∨ k # j)*. Докажем, что $k \#_{\boldsymbol{R}[j,i]} l$.

По определению композиции автоматов переход в автомате *A[j,i]A* получается в результате композиции одного перехода в автомате *A*. Пусть $a,b\in T_A$ и каждый из переходов *a* и *b* компонуется сам с собой, т.е. *f(a,j,i,a)=f(b,j,i,b)=**true***. По определению композиции переходов для $a,b\in T_A$ имеем:

$f(a,j,i,a) \Rightarrow y_a(j)=x_a(i) \;\&\; (j\in q_a \sim i\in p_a)$,

$f(b,j,i,b) \Rightarrow y_b(j)=x_b(i) \;\&\; (j\in q_b \sim i\in p_b)$,

$a[j,i]a = (s_a, x_a/\{i\}, p_a\backslash\{i\}, y_a/\{j\}, q_a\backslash\{j\}, t_a)$ и

$b[j,i]b = (s_b, x_b/\{i\}, p_b\backslash\{i\}, y_b/\{j\}, q_b\backslash\{j\}, t_b)$.

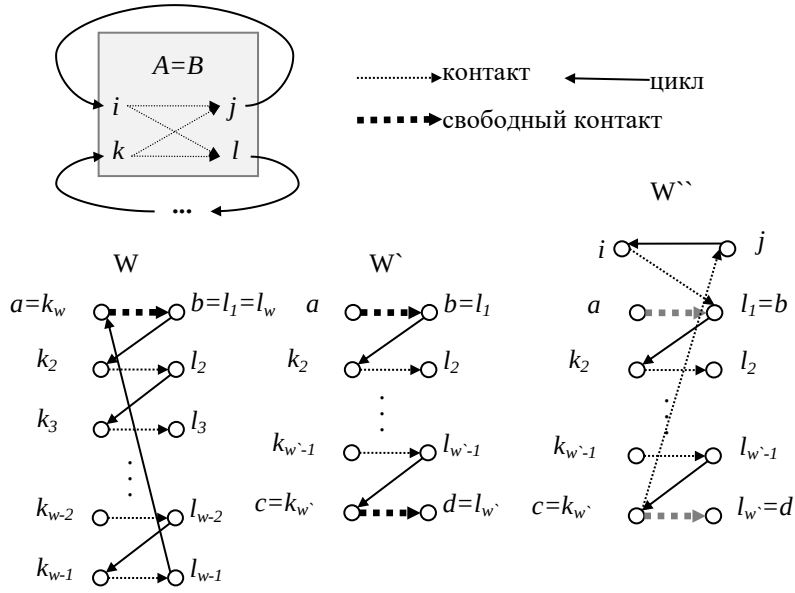


Рис. 3. Свободность цикла при композиции по петле
Fig. 3. Free cycle when the loop composition

Пусть $s_a=s_b$ и $x_a/\{i\}/\{k\}=x_b/\{i\}/\{k\}$.

Обозначим $s=s_a$, $x=x_a/\{i\}/\{k\}$. Тогда $s=s_b$, $x=x_b/\{i\}/\{k\}$.

Надо доказать, что $(y_a/\{i\})(l)=(y_b/\{i\})(l)$.

Используя обозначения, имеем

$a = (s, x\cup\{(i,x_a(i))\}\cup\{(k,x_a(k))\}, p_a, y_a, q_a, t_a)$ и

$b = (s, x\cup\{(i,x_b(i))\}\cup\{(k,x_b(k))\}, p_b, y_b, q_b, t_b)$.

Поскольку в автомате определены переходы по всем стимулам, должны быть ещё два перехода:

$c = (s, x\cup\{(i,x_a(i))\}\cup\{(k,x_b(k))\}, p_c, y_c, q_c, t_c)$ и

$d = (s, x\cup\{(i,x_b(i))\}\cup\{(k,x_a(k))\}, p_d, y_d, q_d, t_d)$.

В переходах a и c стимулы отличаются только по входу k. А тогда, поскольку $k \# l$, реакции на выходе l одинаковы для этих переходов: $y_a(l)=y_c(l)$. В переходах b и d стимулы тоже отличаются только по входу k. Поэтому, если $k \# j$, то $y_b(j)=y_d(j)$.

В переходах a и d стимулы отличаются только по входу i. А тогда, поскольку $i \# j$, реакции на выходе j одинаковы для этих переходов: $y_a(j)=y_d(j)$. В переходах b и c стимулы тоже отличаются только по входу i. Поэтому, если $i \# l$, то $y_b(l)=y_c(l)$.

Теперь, если $k \# j$, то $y_a(j)=y_d(j)$ и $y_b(j)=y_d(j)$. Тогда $y_a(j)=y_b(j)$. А поскольку $y_a(j)=x_a(i)$ и $y_b(j)=x_b(i)$, имеем $x_a(i)=x_b(i)$. Но тогда в переходах a и b

стимулы отличаются только по входу k. А поскольку $k \# l$, имеем $y_a(l)=y_b(l)$. А это влечёт $(y_a/\{i\})(l)=(y_b/\{i\})(l)$.

Если же $i \# l$, то $y_a(l)=y_c(l)$ и $y_b(l)=y_c(l)$. Тогда $y_a(l)=y_b(l)$. А это влечёт $(y_a/\{i\})(l)=(y_b/\{i\})(l)$.

Следовательно, $k \#_{\boldsymbol{R}[j,i]} l$.

Итак во всех трёх случаях мы доказали, что в $\boldsymbol{R}$ в цикле Z существует хотя бы один свободный контакт (k,l), который остаётся свободным после композиции $k \#_{\boldsymbol{R}[j,i]} l$. А поскольку контакт (k,l) имеется в цикле Z в $\boldsymbol{R}[j,i]$, цикл Z свободный в $\boldsymbol{R}[j,i]$.

2. Z не является циклом в $\boldsymbol{R}$, поскольку $A \neq B$, $k \in I_B$ и $l \in J_A$.

   Покажем, что любой контакт (k,l) внутри автомата $A[j,i]B$ такой, что $k \in I_B$ и $l \in J_A$, свободный: $k \#_{\boldsymbol{R}[j,i]} l$ (Рис. 4). Очевидно, $k \neq i$ и $l \neq j$.

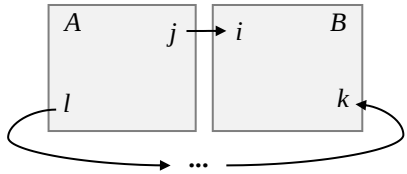


Рис. 4. Свободность контакта (k,l) в $\boldsymbol{R}[j,i]$
Fig. 4. Free contact (k, l) in $\boldsymbol{R}[j,i]$

   Рассмотрим два перехода в композиции:

   $f(a,j,i,b) \Rightarrow y_a(j)=x_b(i)$ & $(j \in q_a \sim i \in p_b)$,

   $f(a`,j,i,b`) \Rightarrow y_{a`}(j)=x_{b`}(i)$ & $(j \in q_{a`} \sim i \in p_{b`})$,

   $a[j,i]b = (s_a \cup s_b,\ x_a \cup x_b/\{i\},\ p_a \cup p_b \setminus \{i\},\ y_a \cup y_b/\{j\},\ q_a \cup q_b \setminus \{j\},\ t_a \cup t_b)$ и

   $a`[j,i]b` = (s_{a`} \cup s_{b`},\ x_{a`} \cup x_{b`}/\{i\},\ p_{a`} \cup p_{b`} \setminus \{i\},\ y_{a`} \cup y_{b`}/\{j\},\ q_{a`} \cup q_{b`} \setminus \{j\},\ t_{a`} \cup t_{b`})$

   такие, что $s_a \cup s_b = s_{a`} \cup s_{b`}$ и $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_{b`}/\{i\}/\{k\}$.

   Надо доказать, что $(y_a \cup y_b/\{j\})(l)=(y_{a`} \cup y_{b`}/\{j\})(l)$.

   Поскольку $A \neq B$, $k \in I_B$ и $i \in I_B$, из $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_{b`}/\{i\}/\{k\}$ следует $x_a = x_{a`}$. А тогда, поскольку автомат $A$ детерминированный, $y_a(l)=y_{a`}(l)$, что вместе с $j \in J_A$ и $l \in J_A$, влечёт $(y_a \cup y_b/\{j\})(l)=(y_{a`} \cup y_{b`}/\{j\})(l)$.

   Следовательно, $k \#_{\boldsymbol{R}[j,i]} l$, а поскольку контакт (k,l) имеется в цикле Z в $\boldsymbol{R}[j,i]$, цикл Z свободный в $\boldsymbol{R}[j,i]$.

3. Z не является циклом в $\boldsymbol{R}$, поскольку $A \neq B$, $k \in I_A$ и $l \in J_B$.

   Очевидно, $k \neq i$ и $l \neq j$. Контакт (a,b) внутри автомата $A[j,i]B$ будем называть *AB-разрывом*, если $a \in I_A$ и $b \in J_B$, и *BA-разрывом*, если $a \in I_B$ и $b \in J_A$.

   Цикл Z можно представить как конкатенацию путей без разрывов в каждом пути и с разрывами между путями: $Z = Z_1 \cdot Z_2 \cdot \ldots \cdot Z_w$. Для $v=1..w$ через k_v обозначим вход последнего соединения в Z_v, а через l_v обозначим выход первого соединения в Z_v.

Тогда для $v=1..w-1$ должно быть $\varphi_{\boldsymbol{R}[j,i]}(k_v)=\varphi_{\boldsymbol{R}[j,i]}(l_{v+1})=A[j,i]B$ и контакт (k_v,l_{v+1}) является разрывом, и $\varphi_{\boldsymbol{R}[j,i]}(k_w)=\varphi_{\boldsymbol{R}[j,i]}(l_1)=A[j,i]B$ и контакт (k_w,l_1) является разрывом. Так как в каждом пути Z_v нет разрывов, он является путём в $\boldsymbol{R}$.

Если в $\boldsymbol{R}[j,i]$ в цикле Z есть BA-разрыв (a,b), то по доказанному в п.2 $a \#_{\boldsymbol{R}[j,i]} b$, следовательно, цикл Z свободный в $\boldsymbol{R}[j,i]$.

Если в каком-то из путей Z_v, где $v=1..w$, есть контакт $a \# b$, то, поскольку путь Z_v является также путём в $\boldsymbol{R}[j,i]$, должно быть $a \neq i$ и $b \neq j$. А тогда по лемме 4 $a \#_{\boldsymbol{R}[j,i]} b$, следовательно, цикл Z свободный в $\boldsymbol{R}[j,i]$.

Далее будем считать, что в $\boldsymbol{R}[j,i]$ в цикле Z есть только AB-разрывы, и в каждом из путей Z_v, где $v=1..w$, нет свободных контактов (Рис. 5).

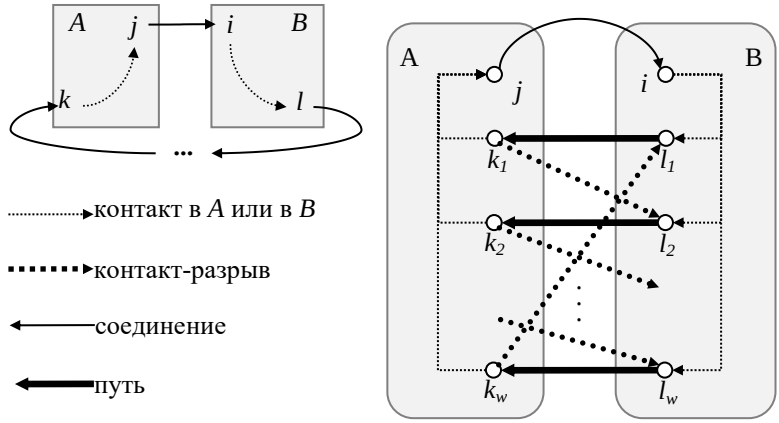


Рис. 5. Цикл как конкатенация путей с разрывами между путями
Fig. 5. The cycle as a concatenation of paths with breaks between paths

Для каждого $v=1..w$ путь Z_v вместе с соединением (j,i) образует в $\boldsymbol{R}$ цикл $Z`_v = Z_v \cdot (j,i)$. В цикле $Z`_v$ есть два контакта внутри автоматов A и B: контакт (k_v,j) в автомате A и контакт (i,l_v) в автомате B. Поскольку остальные контакты цикла $Z`_v$ являются контактами пути $Z_v$, они не свободные. Поскольку все циклы в $\boldsymbol{R}$ свободные, цикл $Z`_v$ должен быть свободным, а тогда должно быть $k_v \# j$ или $i \# l_v$.

Нам надо показать, что для некоторого v имеет место $k_v \#_{\boldsymbol{R}[j,i]} l_{v+1}$, если $v=1..w-1$, или $k_w \#_{\boldsymbol{R}[j,i]} l_1$, если $v=w$.

Допустим для некоторого $v=1..w$ имеет место $k_v \# j$.

Обозначим $k=k_v$ и $l=l_{v+1}$, если $v=1..w-1$, или $l=l_1$, если $v=w$. Имеем $k \# j$. Надо показать, что $k \#_{\boldsymbol{R}[j,i]} l$.

Рассмотрим два перехода в композиции:

$f(a,j,i,b) \Rightarrow y_a(j)=x_b(i) \;\&\; (j \in q_a \sim i \in p_b)$,

$f(a`,j,i,b`) \Rightarrow y_{a`}(j)=x_{b`}(i) \;\&\; (j \in q_{a`} \sim i \in p_{b`})$,

$a[j,i]b = (s_a \cup s_b, x_a \cup x_b/\{i\}, p_a \cup p_b\backslash\{i\}, y_a \cup y_b/\{j\}, q_a \cup q_b\backslash\{j\}, t_a \cup t_b)$ и

$a`[j,i]b` = (s_{a`} \cup s_{b`}, x_{a`} \cup x_{b`}/\{i\}, p_{a`} \cup p_{b`}\backslash\{i\}, y_{a`} \cup y_{b`}/\{j\}, q_{a`} \cup q_{b`}\backslash\{j\}, t_{a`} \cup t_{b`})$

такие, что $s_a \cup s_b = s_{a`} \cup s_{b`}$ и $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_{b`}/\{i\}/\{k\}$.

Достаточно показать, что $(y_a \cup y_b/\{j\})(l) = (y_{a`} \cup y_{b`}/\{j\})(l)$.

Поскольку $A \neq B$, $k \in I_A$ и $i \in I_B$, из $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_{b`}/\{i\}/\{k\}$ следует $x_a/\{k\} = x_{a`}/\{k\}$ и $x_b/\{i\} = x_{b`}/\{i\}$.

Поскольку $k \# j$, из $x_a/\{k\} = x_{a`}/\{k\}$ следует $y_a(j) = y_{a`}(j)$, что влечёт $x_b(i) = x_{b`}(i)$. А это вместе с $x_b/\{i\} = x_{b`}/\{i\}$ влечёт $x_b = x_{b`}$. Но тогда, поскольку автомат $B$ детерминированный, $y_b = y_{b`}$, что влечёт $(y_a \cup y_b/\{j\})(l) = (y_{a`} \cup y_{b`}/\{j\})(l)$, поскольку $A \neq B$, $j \in J_A$ и $l \in J_B$.

Теперь предположим, что для каждого $v=1..w$ не верно, что $k_v \# j$.

Тогда для каждого $v=1..w$ имеет место $i \# l_v$.

Выберем любое $v=1..w$ и обозначим $l=l_v$ и $k=k_{v-1}$, если $v=2..w$, или $k=k_w$, если $v=1$. Имеем $i \# l$. Надо показать, что $k \#_{\boldsymbol{R}[j,i]} l$.

Рассмотрим два перехода в композиции:

$f(a,j,i,b) \Rightarrow y_a(j) = x_b(i) \;\&\; (j \in q_a \sim i \in p_b)$,

$f(a`,j,i,b`) \Rightarrow y_{a`}(j) = x_{b`}(i) \;\&\; (j \in q_{a`} \sim i \in p_{b`})$,

$a[j,i]b = (s_a \cup s_b, x_a \cup x_b/\{i\}, p_a \cup p_b\backslash\{i\}, y_a \cup y_b/\{j\}, q_a \cup q_b\backslash\{j\}, t_a \cup t_b)$ и

$a`[j,i]b` = (s_{a`} \cup s_{b`}, x_{a`} \cup x_{b`}/\{i\}, p_{a`} \cup p_{b`}\backslash\{i\}, y_{a`} \cup y_{b`}/\{j\}, q_{a`} \cup q_{b`}\backslash\{j\}, t_{a`} \cup t_{b`})$

такие, что $s_a \cup s_b = s_{a`} \cup s_{b`}$ и $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_{b`}/\{i\}/\{k\}$.

Достаточно показать, что $(y_a \cup y_b/\{j\})(l) = (y_{a`} \cup y_{b`}/\{j\})(l)$.

Поскольку $A \neq B$, $k \in I_A$ и $i \in I_B$, из $x_a \cup x_b/\{i\}/\{k\} = x_{a`} \cup x_{b`}/\{i\}/\{k\}$ следует $x_b/\{i\} = x_{b`}/\{i\}$.

Поскольку $i \# l$, из $x_b/\{i\} = x_{b`}/\{i\}$ следует $y_b(l) = y_{b`}(l)$, что влечёт $(y_a \cup y_b/\{j\})(l) = (y_{a`} \cup y_{b`}/\{j\})(l)$, поскольку $A \neq B$, $j \in J_A$ и $l \in J_B$.

Лемма 5 доказана.

Теорема 1 о вполне определенности и детерминированности системы автоматов:

Если в системе $\boldsymbol{R}=(M,V,E)$ все автоматы вполне определенные и детерминированные, и все циклы соединений свободные, то система $\boldsymbol{R}[Z\hat{}]$, где $Z\hat{}$ – последовательность всех соединений, состоит из вполне определенных детерминированных автоматов, не связанных друг с другом. При этом система $\boldsymbol{R}[Z\hat{}]$ не зависит от выбора последовательности $Z\hat{}$ для одного и того же множества E.

Доказательство: Так как в системе $\boldsymbol{R}=(M,V,E)$ все автоматы вполне определенные и детерминированные и все соединения-петли (как циклы соединений) свободные, по леммам 1 и 2 в композиции $\boldsymbol{R}[j,i]$ для любого соединения $(j,i) \in E$ все автоматы тоже вполне определенные и детерминированные. Так как в системе $\boldsymbol{R}$ все автоматы вполне определенные и детерминированные, и все циклы свободные, по лемме 5 в композиции $\boldsymbol{R}[j,i]$

все циклы тоже свободные. Таким образом, в системе ***R[j,i]***, как и в исходной системе ***R***, все автоматы вполне определенные и детерминированные, и все циклы соединений свободные. Будем выполнять композицию до тех пор, пока в системе есть соединения. Так как число соединений в системе конечно, а при композиции оно уменьшается на 1, через конечное число композиций (равное числу соединений в ***R***) получим систему ***R[Z^]***, в которой нет соединений, и которая состоит только из вполне определенных и детерминированных автоматов. По ассоциативности композиции (теорема 3 в [2]) система ***R[Z^]*** не зависит от выбора последовательности *Z^* для одного и того же множества *E*.

Теорема 1 доказана.

4. Системы автоматов класса «вершины-дуги»

В нашей статье [1] система автоматов описывалась как граф, в вершинах которого находились автоматы, а дуги соответствовали каналам связи, через которые автоматы обменивались сообщениями. При этом дуга представляла собой очередь сообщений длины 1. Она моделировалась автоматом с одним входом и одним выходом. Переход в автомате (вершины или дуги) помечался только парой *(x,y)*, где x – стимул, а *y* – реакция, в отличие от более общих *(x,p,y,q)*-переходов в модели, используемой в данной статье. Автоматы вершин и дуг были детерминированными, но, кроме того, в автомате дуги реакция (сообщение, выбираемое из начала очереди) зависела только от состояния автомата, т.е. не зависела от стимула (сообщения, помещаемого в конец очереди).

Для того чтобы интерпретировать автоматы из [1] в терминах данной статьи, выполним следующие преобразования:

1. Нам нужно заменить стимул *x* и реакцию *y* как частичные отображения на полные отображения *x`* и *y`*, а также добавить параметры *p* и *q*. В каждом автомате каждый переход *(s,x,y,t)* заменяется на множество кратных переходов вида *(s,x`,p,y`,q,t)*, где стимулы *x* и *x`* совпадают на домене *x*, т.е. $\forall i \in \mathbf{Dom}(x)\ x(i)=x`(i)$, $p= \mathbf{Dom}(x)$, реакции *y* и *y`* совпадают на домене *y*, а на выходах из его дополнения сообщения пустые, т.е. $\forall j \in \mathbf{Dom}(y)\ y(j)=y`(j)$ и $\forall j \notin \mathbf{Dom}(y)\ y`(j)=\Lambda$, $q= \mathbf{Dom}(y)$.
2. Если после этого в автомате в некотором состоянии *s* нет перехода по некоторому стимулу *x`*, то добавляется переход-петля по этому стимулу без приёма и выдачи сообщений: $(s,x`,\varnothing,J\times\{\Lambda\},\varnothing,s)$.
3. Кроме этого, в автомате дуги (очередь длины 1) с входом *i* и выходом *j* добавляются переходы для любого стимула без приёма сообщений для случая, когда на дуге находится сообщение, которое не может быть передано: $\forall m \in M\ \forall m` \in M_\Lambda\ (m,(i,m`),\varnothing,(j,m),\varnothing,m)$.

Легко видеть, что после такого преобразования автомат вершины определен по всем стимулам, а автомат дуги вполне определен. Если автоматы

детерминированы в смысле [1], то после преобразования они детерминированы в терминах данной статьи. Также в автомате дуги реакция зависит только от состояния (не зависит от стимула).

Однако для моделирования системы автоматов из [1] в терминах данной статьи мы не можем применять композицию, которая определена в данной статье (и в [2]). Дело в том, что в [1] предполагалась *семантика с асинхронными переходами*, когда один автомат «стоит» (не выполняет никакого перехода), а связанный с ним автомат может работать (выполнять переход). Это происходит тогда, когда автоматы некоторых дуг, выходящих из вершины, не принимают сообщения, посылаемые на эти дуги автоматом этой вершины. В этом случае автомат вершины «стоит», а автоматы выходящих дуг могут выполнять переходы, передавая находящиеся на них сообщения в автоматы вершины, в которые эти дуги входят. В данной статье (и в [2]), как указано в начале раздела 3, используется *семантика без асинхронных переходов*: переход в каждом автомате выполняется одновременно с переходами в автоматах, связанных с ним по соединениям, поэтому, когда один автомат «стоит», все связанные с ним автоматы тоже «стоят».

Для моделирования семантики с асинхронными переходами мы предложим специальную асинхронную композицию автомата вершины с автоматами всех выходящих дуг. Почему автомат вершины следует единовременно компоновать с автоматами сразу всех выходящих дуг? Причина этого в том, что в семантике статьи [1] автомат вершины выполняет переход, принимая сообщения со своих входов, только в том случае, когда удается передать по его выходам все непустые сообщения, описываемые реакцией перехода. Иными словами, в терминах *(x,p,y,q)*-переходов параметр *q* всегда был равен множеству тех выходов *j*, по которым передавалось непустое сообщение $y(j) \neq \Lambda$. Поэтому то, «стоит» автомат вершины или выполняет переход, зависит от состояния автоматов всех выходящих дуг.

Мы также обобщим понятие дуги так, чтобы автомат дуги мог моделировать не только очередь длины 1, но также очередь любой длины, очередь с приоритетами сообщений, стек и т.п. В самом общем виде дуга будет рассматриваться как автомат-посредник между автоматами вершин, но с одним важным ограничением: у автомата дуги должен быть только один вход. Заметим, что выходов в автомате дуги может быть несколько.

Будем говорить, что система автоматов $\boldsymbol{R} = (M,V,E)$ относится к классу «вершины-дуги», если 1) множество V автоматов разбито на два подмножества: автоматы вершин и автоматы дуг; 2) автоматы вершин и дуг чередуются: каждое соединение ведёт либо из автомата вершины в автомат дуги, либо из автомата дуги в автомат вершины; 3) автоматы вершин определены по всем стимулам, а автоматы дуг вполне определены; 4) все автоматы детерминированные; 5) в автомате дуги реакция зависит только от состояния, т.е. не зависит от стимула, 6) автомат дуги имеет только один вход.

Асинхронная композиция автомата вершины с автоматами всех выходящих дуг.

Пусть в системе $\boldsymbol{R} = (M,V,E)$ класса «вершины-дуги» имеется автомат вершины A и автоматы выходящих дуг $B1,...,Bn$ так, что каждое соединение (j,i), где $j \in J_A$, ведёт в некоторый из автоматов дуг, т.е. $\exists m=1..n$ такое, что $i \in I_{Bm}$, и для каждого $m=1..n$ найдётся такое соединение (l,k), что $l \in J_A$ и $k \in I_{Bm}$.

Для m-ого ($m=1..n$) автомата дуги множество его входов, используемых в соединениях с автоматом вершины, равно $I_m = \{i \in I_{Bm} | \exists j \in J_A\ (j,i) \in E\}$. Множество I входов композиции определим как множество всех входов автоматов-операндов, кроме тех, что используются в соединениях автомата вершины с автоматами дуг: $I = I_A \cup (I_{B1} \setminus I_1) \cup ... \cup (I_{Bn} \setminus I_n)$. Множество J выходов композиции определим как множество всех выходов автоматов дуг: $J = J_{B1} \cup ... \cup J_{Bn}$. Состояние композиции – это набор состояний автоматов-операндов $s=(s_A,s_1,...,s_n) \in S_A \times S_{B1} \times ... \times S_{Bn}$. Стимул композиции $x: I \to M_\Lambda$.

Для данного композиционного состояния s и композиционного стимула x в автомате вершины выделяется его состояние s_A и его стимул $x_A = x \cap (I_A \times M_\Lambda)$. Поскольку автомат вершины определен по всем стимулам и детерминирован, пара (s_A,x_A) однозначно определяет множество переходов вида $a=(s_A,x_A,p_A,y_A,q_A,t_A) \in T_A$, отличающихся друг от друга только параметром q_A и постсостоянием t_A, однозначно определяемым остальными параметрами перехода. Реакция y_A вместе с композиционным стимулом x однозначно определяют стимул для каждого m-го автомата дуги $x_m = (x \cap (I_{Bm} \times M_\Lambda)) \cup \{(i,y_A(j)) | i \in I_m \ \&\ (j,i) \in E\}$. Поскольку автомат дуги вполне определен и детерминирован, пара (s_m,x_m) однозначно определяет в автомате дуги множество переходов вида $a_m=(s_m,x_m,p_m,y_m,q_m,t_m) \in T_{Bm}$, отличающихся друг от друга только параметром q_m, который пробегает все возможные значения, и постсостоянием t_m, однозначно определяемым остальными параметрами перехода. Поскольку для каждого $(j,i) \in E$, где $i \in I_m$, имеет место $x_m(i)=y_A(j)$, переход a выполняется, если для каждого $m=1..n$ выполняется остальная часть условия композиции перехода a с переходом a_m по соединению (j,i): $j \in q_A \sim i \in p_m$.

Если эти условия выполнены для некоторого перехода a, т.е. для некоторого q_A, то они, очевидно, не выполнены для остальных q_A. Тогда определяется композиционный переход (s,x,p,y,q,t), где $p = p_A \cup (p_1 \setminus I_1) \cup ... \cup (p_n \setminus I_n)$, $y = y_1 \cup ... \cup y_n$, $q = q_1 \cup ... \cup q_n$, $t = (t_A,t_1,...,t_n)$.

Если эти условия не выполнены ни для какого перехода a, т.е. ни для какого q_A, то автомат вершины не меняет состояние, не принимает и не передает сообщения, поэтому на входах автоматов дуг, соединенных с выходами автомата вершины, находятся пустые сообщения. Эти пустые сообщения вместе с композиционным стимулом x однозначно определяют стимул для каждого m-го автомата дуги $x`_m = (x \cap (I_{Bm} \times M_\Lambda)) \cup \{(i,\Lambda) | i \in I_m\}$. Поскольку автомат дуги вполне определен и детерминирован, пара $(s_m,x`_m)$ однозначно

определяет в автомате дуги множество переходов вида $a`_m=(s_m,x`_m,p`_m,y`_m,q`_m,t`_m)$, отличающихся друг от друга только параметром $q`_m$, который пробегает все возможные значения, и постсостоянием $t`_m$, однозначно определяемым остальными параметрами перехода. В этом случае определяется композиционный переход $(s,x,p`,y`,q`,t`)$, где $p` = (p`_1\backslash I_1) \cup ... \cup (p`_n\backslash I_n)$, $y` = y`_1 \cup ... \cup y`_n$, $q` = q`_1 \cup ... \cup q`_n$, $t` = (s_A,t`_1,...,t`_n)$. Этот композиционный переход как раз и моделирует асинхронные переходы в автоматах дуг, когда автомат вершины «стоит».

Теорема 2 об асинхронной композиции автомата вершины с автоматами всех выходящих дуг:

В системе автоматов класса «вершины-дуги» асинхронная композиция автомата вершины с автоматами всех выходящих дуг вполне определена и детерминирована, а все её контакты свободные.

Доказательство: Определенность композиции по всем стимулам следует из определения композиции. Определенность композиции по всем q следует, кроме того, из вполне определенности автоматов дуг. Детерминированность композиции следует из определения композиции и детерминированности автоматов вершин и дуг. Поскольку реакция в автомате дуги зависит только от состояния дуги, а в композиции реакция состоит из реакций автоматов дуг, композиционная реакция зависит только от композиционного состояния, содержащего все состояния автоматов дуг. Тем самым, все контакты в композиции свободные.

Теорема 2 доказана.

Поскольку в системе класса «вершины-дуги» автоматы вершин и дуг строго чередуются, и каждая дуга имеет только один вход, асинхронную композицию можно проводить до тех пор, пока остаются автоматы вершин. В конечном счете останутся только вполне определенные и детерминированные автоматы, полученные с помощью асинхронной композиции, все контакты которых свободные, а также автоматы внешних входных дуг, которые как и любые автоматы дуг, тоже вполне определенные и детерминированные, а все их контакты свободные. После этого мы уже можем использовать семантику без асинхронных переходов, применяя композицию, определенную в данной статье (и в [2]). После завершения такой композиции системы по теореме 1 гарантируется детерминизм системы класса «вершины-дуги».

Возникает вопрос: а что делать, если дуга имеет несколько входов? Возможно ли соответствующее обобщение системы автоматов класса «вершины-дуги» в семантике с асинхронными переходами? Как было сказано выше, аксинхронная композиция должна компоновать автомат вершины с автоматами сразу всех выходящих из вершины дуг. Поэтому, если дуга имеет несколько входов, то её следует компоновать сразу со всеми вершинами, выходы которых соединены со входами этой дуги. В целом в асинхронной композиции должен участвовать двудольный подграф системы: все соединения в нем ведут из вершин в дуги, и подграф замкнут по выходящим

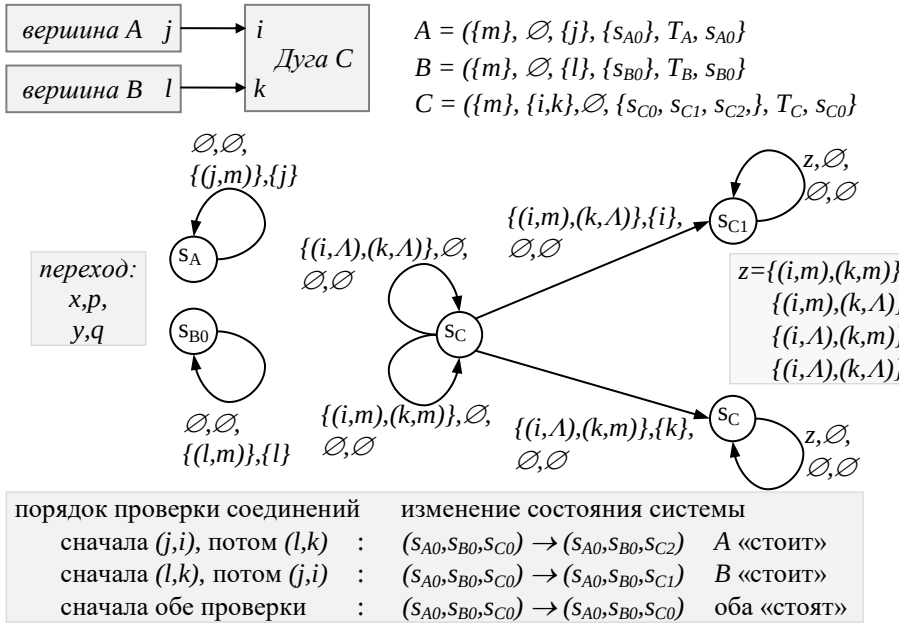
из вершин соединениям. Тем самым в асинхронной композиции может участвовать сразу несколько автоматов вершин, и некоторые из них могут «стоять».

В целом получается следующая процедура. Поскольку в автомате дуги реакция зависит только от её состояния, состояния автоматов всех дуг однозначно определяют сообщения на выходах автоматов всех дуг. Поскольку автоматы вершин и дуг чередуются, тем самым однозначно определены сообщения на входах автоматов всех вершин. Поскольку автомат вершины определен по всем стимулам и детерминирован, в каждом автомате вершины однозначно определяется параметр p и реакция y подмножества выполнимых переходов. Эти реакции однозначно определяют сообщения на выходах автоматов всех вершин. Поскольку автоматы вершин и дуг чередуются, тем самым однозначно определены сообщения на входах автоматов всех дуг. Поскольку автомат дуги определен по всем стимулам и детерминирован, в каждом автомате дуги однозначно определяется параметр p подмножества выполнимых переходов.

После этого по каждому соединению (j,i), ведущему из автомата вершины $\varphi(j)$ в автомат дуги $\varphi(i)$, проверяется согласованность параметров p и q, т.е. проверяется, что $j \in q \sim i \in p$. Если это не так, автомат вершины $\varphi(j)$ должен «стоять», но тогда меняются сообщения на его выходах и, соответственно, сообщения на входах дуг, с которыми эти выходы соединены: эти сообщения становятся пустыми. В этом случае требуется повторно «пересчитать» подмножества выполнимых переходов в автоматах дуг.

Если проверка согласованности параметров p и q по соединениям и «пересчет» подмножеств выполнимых переходов в автоматах дуг будут чередоваться, то это может привести к недетерминизму, т.е. результирующее состояние системы (как набор состояний всех автоматов) будет зависеть от того, в каком порядке проверяются соединения (пример на Рис. 6). Поэтому нужно сначала выполнить проверку всех соединений, определяя, какие автоматы вершин будут «стоять», а уже потом заново «пересчитывать» подмножества выполнимых переходов в автоматах дуг. Этот процесс может потребовать несколько итераций, но он гарантированно «сходится», поскольку на каждом шаге, кроме последнего, число «стоящих» автоматов вершин увеличивается, а общее число вершин конечно.

Получившаяся процедура асинхронной композиции достаточно сложная, а её семантика недостаточно «прозрачна». Это наводит на мысль попробовать разработать новую модель взаимодействия автоматов, которая совмещала бы в себе достоинства семантики с асинхронными переходами и семантики без асинхронных переходов.



порядок проверки соединений		изменение состояния системы	
сначала *(j,i)*, потом *(l,k)*	:	$(s_{A0},s_{B0},s_{C0}) \to (s_{A0},s_{B0},s_{C2})$	*A* «стоит»
сначала *(l,k)*, потом *(j,i)*	:	$(s_{A0},s_{B0},s_{C0}) \to (s_{A0},s_{B0},s_{C1})$	*B* «стоит»
сначала обе проверки	:	$(s_{A0},s_{B0},s_{C0}) \to (s_{A0},s_{B0},s_{C0})$	оба «стоят»

Рис. 6. Пример недетерминизма
Fig. 6. An example of nondeterminism

5. Тестирование

Если для системы $\boldsymbol{R}=(M,V,E)$ не делается никаких предположений о том, правильно ли работают автоматы системы из множества V и правильно ли (не меняющееся) множество соединений E, то для тестирования системы тесты генерируются по композиции $\boldsymbol{R}[Z^\wedge]$. Если композиция состоит из нескольких автоматов, то, поскольку эти автоматы не связаны между собой, то тесты можно генерировать для каждого автомата из $V_{\boldsymbol{R}[Z^\wedge]}$ независимо: полный набор тестов системы совпадает с объединением полных наборов тестов для всех автоматов из $V_{\boldsymbol{R}[Z^\wedge]}$. Иными словами, каждый компонент связности системы $\boldsymbol{R}$ тестируется отдельно и независимо от других компонентов связности, что естественно.

Как указывалось в нашей предыдущей статье [1], если в соединениях (множестве E) нет ошибок (будем называть это *гипотезой о связях*), то есть множество соединений совпадает с заданным, то такое тестирование системы сводится к проверке правильности переходов каждого автомата из $V_{\boldsymbol{R}}$. Поскольку компонент тестируется как часть системы, не все переходы автомата компонента, которые можно проверить при автономном тестировании с прямым доступом к компоненту, можно проверить при тестировании системы. Поэтому речь идёт только о *достижимых переходах*

автоматов, то есть таких переходах, которые могут быть выполнены при работе автомата как части системы. С другой стороны, если нас интересует правильность работы системы в целом, то достаточно «правильности» только достижимых переходов.

Тест моделируется автоматом, выходы которого соединяются с внешними входами системы, а внешние выходы системы соединяются со входами теста. В результате получается замкнутая система, т.е. система без внешних входов и выходов. Автоматы такой системы, в том числе тест, работают синхронно: на каждом такте каждый автомат выполняет один переход.

В данном разделе предлагается алгоритм построения набора тестов, который является полным (проверяет все переходы автоматов компонентов, достижимые при работе этих компонентов в системе) при выполнении двух условий: 1) верна гипотеза о связях, 2) система детерминирована. Дополнительно этот алгоритм определяет недостижимые переходы автоматов компонентов. Предполагается, во-первых, что нам известно, каким должен быть автомат каждого компонента (задан граф переходов автомата с точностью до изоморфизма) и именно это должно проверяться при тестировании. Во-вторых, тестовая система может не только выполнять тест, посылая сообщения на внешние входы системы и принимая сообщения с внешних выходов системы, но и наблюдать как состояния автоматов системы, так и сообщения, передаваемые по соединениям между автоматами системы.

Поскольку мы не налагаем никаких ограничений на связность графов переходов автоматов, вообще говоря, полный набор тестов содержит более одного теста. Это означает, что требуется рестарт системы при переходе от одного теста к другому: при рестарте происходит «подмена» одного теста другим.

Для решения этой задачи мы предлагаем использовать любой алгоритм генерации полного набора тестов для одного детерминированного автомата. Таких алгоритмов предложено довольно много, они базируются на построении набора маршрутов, покрывающих граф переходов автомата, достижимых из его начального состояния (смотри, например, [8]). В качестве такого автомата для наших целей выбирается каждый из автоматов композиционной системы $\boldsymbol{R}[Z\hat{}]$. Понятно, что покрывая все достижимые переходы автомата $A\hat{}\in V_{\boldsymbol{R}[Z\hat{}]}$, мы покрываем все достижимые переходы всех автоматов из соответствующего компонента связности системы $\boldsymbol{R}$, который обозначим $V(A\hat{})$. Однако такой набор тестов может оказаться сильно избыточным для решения нашей задачи: покрытие всех достижимых переходов всех автоматов из $V(A\hat{})$ не обязательно требует покрытия всех достижимых переходов автомата $A\hat{}$.

Поэтому предлагается в процессе генерации полного набора тестов для автомата $A\hat{}$ применять процедуру фильтрации, которая будет отбрасывать «лишние» тесты. Эта процедура работает следующим образом. С самого начала создаётся пустое множество T генерируемого набора тестов и

множество P непокрытых переходов автоматов из $V(A^\wedge)$, которое сначала равно множеству всех переходов всех автоматов из $V(A^\wedge)$. Когда генерируется очередной i-ый тест T_i для автомата $A^\wedge$, вычисляется множество P_i переходов автоматов из $V(A^\wedge)$, покрываемое этим тестом. Тесту соответствует маршрут $Z^\wedge$ в автомате $A^\wedge$. Каждому переходу маршрута $Z^\wedge$ соответствует множество переходов всех автоматов из $V(A^\wedge)$: по одному переходу на каждый автомат из $V(A^\wedge)$. Объединение этих множеств для всех переходов из $Z^\wedge$ и есть множество P_i. Далее алгоритм фильтрации проверяет, покрывает ли i-ый тест какой-либо новый, ещё не покрытый переход какого-либо автомата компонента. Если $P_i \cap P = \varnothing$, то никаких новых переходов i-ый тест не покрывает, и он отбрасывается. В противном случае тест добавляется к набору тестов $T := T \cup \{T_i\}$, а из множества непокрытых переходов удаляются новые переходы $P := P \setminus P_i$. После того как все тесты сгенерированы и отфильтрованы, получившееся множество T является искомым полным набором тестов, а множество P – множеством недостижимых переходов автоматов компонентов.

6. Заключение

В статье [1] в заключении были перечислены направления дальнейших исследований из семи пунктов. После статьи [2] и данной статьи остаются четыре пункта, которые мы здесь и перечислим.

1) Предложенный алгоритм фильтрации не обязательно даёт оптимальный полный набор тестов. Оптимальность может пониматься как минимальное число или минимальная суммарная длина тестовых последовательностей. Соответственно, возникает задача оптимизации, т.е. поиска оптимального набора, которая, вообще говоря, сводима к задаче о поиске минимального покрытия ([9], [10]).

2) При тестировании, как обычно, возникает задача «огрубления», когда проверяется не «всё подряд», а проверка делается выборочно. Это называют факторизацией тестирования [11]. Например, для проверки правильности реализации числовой функции домены её аргументов разбиваются на поддомены, и из каждого поддомена выбирается хотя бы одно число для проверки. В терминах рассматриваемой здесь задачи факторизация означает, что на переходах автомата компонента определяется отношение эквивалентности, и считается, что достаточно проверить хотя бы один переход из класса эквивалентности. Задача заключается в том, чтобы уметь находить «правильные» отношения эквивалентности, т.е. отношения, удовлетворяющие некоторым практическим потребностям.

6) В данной статье предлагается решение проблемы тестирования компонентов только детерминированной системы автоматов. Одним из направлений дальнейших исследований могло бы стать исследование проблемы недетерминизма. Более точно, ставится задача определить такие ограничения на недетерминизм системы и/или составляющих её автоматов,

которые позволяли бы выполнять полное тестирование за конечное время. Для автономного тестирования, когда автомат находится под непосредственным управлением теста (не в контексте окружающей его части системы), предложенные неплохие решения этой задачи ([12], [13], [14], [15]).

7) В данной статье предполагается, что известно, каким должен быть автомат каждого компонента: задан граф переходов автомата с точностью до изоморфизма. В более общем случае между автоматом компонента в реализации и автоматом компонента в спецификации задаётся то или иное отношение конформности, которое слабее изоморфизма: квази-редукция, симуляция и т.п. Это требует более сложного алгоритма тестирования. Если тестирование компонента автономное, т.е. не в контексте других компонентов системы и связующих их дуг, то возможно полное тестирование за конечное время конформности типа редукции или слабой симуляции ([4], [5], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]). Для составной системы возникает проблема декомпозиции системных требований, называемая также проблемой несохранения конформности. Она заключается в том, что композиция реализаций компонентов, конформных спецификациям этих компонентов, в общем случае неконформна спецификации системы, в частности, композиции спецификаций компонентов. Этой проблеме посвящён ряд работ ([4], [6], [23]), но остаётся задача переосмысления предложенных решений для рассматриваемой в этой статье задачи тестирования компонентов системы при условии, что верна гипотеза о связях.

Список литературы

[1]. И.Б.Бурдонов, А.С.Косачев. Тестирование системы автоматов. Труды ИСП РАН, том 28(1), 2016 г. DOI: 10.15514/ISPRAS-2016-28(1)-7

[2]. И.Б.Бурдонов, А.С.Косачев. Система автоматов: композиция по графу связей. Труды ИСП РАН, том 28(1), 2016 г. DOI: 10.15514/ISPRAS-2016-28(1)-8

[3]. Revised Working Draft on “Framework: Formal Methods in Conformance Testing”. JTC1/SC21/WG1/Project 54/1, ISO Interim Meeting, ITU-T on, Paris, 1995 г.

[4]. И.Б.Бурдонов, Косачев А.С., В.В.Кулямин. Теория соответствия для систем с блокировками и разрушением. «Физ-мат лит» Наука, Москва, 2008 г., 412 стр.

[5]. И.Б.Бурдонов. Теория конформности (функциональное тестирование программных систем на основе формальных моделей). LAP Lambert Academic Publishing, 2011 г., 428 стр.

[6]. И.Б.Бурдонов, А.С.Косачев. Пополнение спецификации для ioco. Программирование, 2011 г., №1, стр. 3-18.

[7]. А. Камкин, М. Чупилко. Обзор современных технологий имитационной верификации аппаратуры. Программирование, 2011 г., №3, стр. 42-49.

[8]. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай. Программирование, 2003 г., №5, стр. 59-69.

[9]. Ананий В. Левитин. Алгоритмы: введение в разработку и анализ. М.: «Вильямс», 2006 г., стр. 160-163, ISBN 0-201-74395-7.

[10]. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ. 2-ое издание. М.: «Вильямс», 2006 г., стр. 456-458, ISBN 0-07-013151-1.
[11]. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ. Программирование. 2000 г., № 2, стр.12-28.
[12]. И.Б. Бурдонов, А.С. Косачев. Полное тестирование с открытым состоянием ограниченно недетерминированных систем. Программирование, 2009 г., №6, стр. 3-18.
[13]. И.Б.Бурдонов, А.С.Косачев. Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования. Вестник Томского Государственного Университета, № 2(15), 2011 г., стр. 89-98.
[14]. И.Б. Бурдонов, А.С. Косачев. Тестирование конформности на основе соответствия состояний. Труды ИСП РАН, № 18, 2010 г., стр. 183-220.
[15]. И.Б.Бурдонов, А.С.Косачев, Безопасное тестирование симуляции систем с отказами и разрушением. Моделирование и анализ информационных систем, том 17(4), 2010 г., стр. 27-40.
[16]. I.B.Bourdonov, A.S.Kossatchev, V.V.Kuliamin. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proceedings of the Workshop on Model Based Testing (MBT 2004), Elsevier, 2006.
[17]. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Формализация тестового эксперимента. Программирование, 2007 г., №5, стр. 3-32.
[18]. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Безопасность, верификация и теория конформности. Материалы второй международной научной конференции по проблемам безопасности и противодействия терроризму. МГУ 2006, М., МЦНМО, 2007 г., стр. 135-158.
[19]. И.Б.Бурдонов, Косачев А.С. Системы с приоритетами: конформность, тестирование, композиция. Программирование, 2009 г., № 4, стр.24-40.
[20]. И.Б. Бурдонов, А.С. Косачев. Тестирование с преобразованием семантик. Труды ИСП РАН, том 17, 2009 г., стр.193-208.
[21]. A.Kossachev, I.Burdonov. Formal Conformance Verifcation, Short Papers of the 22nd IFIP ICTSS, Alexandre Petrenko, Adenilso Simao, Jose Carlos Maldonado (eds.), Nov. 08-10, 2010, Natal, Brazil, pp.1-6.
[22]. А.С. Косачев, И.Б.Бурдонов. Семантики взаимодействия с отказами, дивергенцией и разрушением. Программирование, 2010 г., №5, стр. 3-23.
[23]. И.Б.Бурдонов, А.С.Косачев. Согласование конформности и композиции. Программирование, 2013 г., №6, стр. 3-15.

Automata system: determinism conditions and testing

Igor Burdonov <igor@ispras.ru>
Alexander Kossatchev kos@ispras.ru
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. The problem of testing of aggregate systems is considered. The system components are described with finite automata with multiple entries and exits. The communication between automata is described with message passing over simplex

communication channels. The system is described with an oriented graph of links. Each node of the graph corresponds to automaton of a component and an arc corresponds to a communication channel connecting exit of one automaton with entry of another automaton. The hypothesis of the links is assumed: the graph of links is static and the link structure is error-free. Automaton of the graph node in each state can accept multiple messages from its entries (at most one message from each entry) and send multiple messages to its exits (at most one message to each exit). An associative composition of the automata is defined. The restrictions on the automata and the graph of links are determined, for which their composition, i.e. the system itself, is deterministic. The goal of testing is to cover transitions of the automata reachable during the system work. It assumed that during testing it is possible to observe the states of automata and the messages on the arcs. The complete testing of the automata system without considering the hypothesis of links may require the number of testing actions of order of multiplication of numbers of states of the component automata, while with the consideration of the hypothesis of links – of order of the sum of these numbers. If the numbers of states of all automata are equal, it gives exponential reduction of the number of test actions. If the hypothesis of links is true, an algorithm of test generation for a deterministic system is proposed basing on filtration of tests generated for covering all transitions of the composition. Test is rejected if it covers only such transitions of the components that are covered by the remaining tests. In conclusion, the directions of future research are described.

Keywords: directed graphs, graph coverage, communicating automata, automata composition, distributed systems, testing, networks.

DOI: 10.15514/ISPRAS-2016-28(1)-9

For citation: Burdonov I.B., Kossatchev A.S. Automata system: determinism conditions and testing. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 151-184 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-9

References

[1]. I. B. Burdonov, A. S. Kossatchev. Testirovanie sistemy avtomatov [Testing of automata system]. Trudy ISP RAN [The proceeding of ISP RAS], Vol. 28(1), 2016. (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-7

[2]. I. B. Burdonov, A. S. Kossatchev. Sistema avtomatov: kompoziciia po grafu sviazei [Automata system: composition on connection graph]. Trudy ISP RAN [The proceeding of ISP RAS], Vol. 28(1), 2016 г.. (in Russian)

[3]. Revised Working Draft on “Framework: Formal Methods in Conformance Testing”. JTC1/SC21/WG1/Project 54/1, ISO Interim Meeting, ITU-T on, Paris, 1995.

[4]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Teoriya sootvetstviya dlya system s blokirovkami I razrusheniem [Conformance theory of the systems with Refused Inputs and Forbidden Actions]. Moscow, «Nauka», 2008, 412 p. (in Russian)

[5]. Bourdonov I. Teoriya konformnosti (funkcional'noe testirovanie prorammny'kh system na osnove formal'ny'kh modelej [Conformance theory (functional testing on formal model base)]. LAP LAMBERT Academic Publishing, Saarbrucken, Germany, 2011, ISBN 978-3-8454-1747-9, 428 p. (in Russian)

[6]. Bourdonov I.B., Kossatchev A.S. Specification Completion for IOCO. Programming and Computer Software, vol. 37(1), 2011, pp. 1-14. *DOI*: 10.1134/S0361768811010014

[7]. A. S. Kamkin, M. M. Chupilko. Survey of modern technologies of simulation-based verification of hardware. Programming and Computer Software, vol. 37 (3), 2011, pp. 147-152. *DOI*: 10.1134/S0361768811030017

[8]. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Deterministic Case. Programming and Computer Software, vol. 29(5), 2003, pp. 245-258. *DOI*: 10.1023/A:1025733107700

[9]. A. Levitin. Algoritmy: vvedenie v razrabotku i analiz [Introduction to The Design and Analysis of Algorithms]. M.: «Viliams», 2006, pp. 160-163, ISBN 0-201-74395-7. (in Russian)

[10]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms. 2-nd Edition. MIT Press Cambridge, MA, USA, 2001, ISBN 0-262-03293-7.

[11]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Application of Finite Automatons for Program Testing. Programming and Computer Software, vol. 26(2), 2000, pp. 61-73. *DOI*: 10.1007/BF02759192

[12]. Bourdonov I.B., Kossatchev A.S. Complete Open-State Testing of Limitedly Nondeterministic Systems. Programming and Computer Software, vol. 35(6), 2009, pp.301-313. DOI: 10.1134/S0361768809060012

[13]. Bourdonov I.B., Kossatchev A.S. Semantiki vzaimodejstviya s otkazami, divergentsiej i razrusheniem. Chast' 2. Usloviya konechnogo polnogo testirovaniya. [Semantics of Interaction with Refused Inputs, Divergence and Forbidden Actions. Part 2. The condition of finite complete testing]. Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika. [Tomsk State University. Journal of Control and Computer Science], 2011, №2, pp. 89-98. (in Russian)

[14]. Bourdonov I.B., Kossatchev A.S. Testirovanie konformnosti na osnove sootvetstviya sostoyanij [Conformance testing based on a state relation]. Trudy ISP RAN [The proceeding of ISP RAS], vol. 18, 2010, pp. 183-320. (in Russian)

[15]. 84. Bourdonov I.B., Kossatchev A.S.. Safe simulation testing of systems with refusals and destructions. Automatic Control and Computer Sciences, vol. 45(7), 2011, pp. 380-389. *DOI*: 10.3103/S0146411611070042

[16]. I.B.Bourdonov, A.S.Kossatchev, V.V.Kuliamin. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proceedings of the Workshop on Model Based Testing (MBT 2004), Elsevier, 2006.

[17]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Formalization of Test Experiments. Programming and Computer Software, vol. 33(5), 2007, pp. 239-260. *DOI*: 10.1134/S0361768807050015

[18]. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Bezopasnost', verifikatsiya i teoriya konformnosti [Safety, Verification and Conformance Theory]. Materialy Vtoroj mezhdunarodnoj nauchnoj konferentsii po problemam bezopasnosti i protivodejstviya terrorizmu [The proceeding of the Second international conference on the problems of safety and counteraction against terrorizm], Moscow, MNCMO, 2007, pp. 135-158. (in Russian)

[19]. Bourdonov I.B., Kossatchev A.S. Systems with Priorities: Conformance, Testing, and Composition. Programming and Computer Software, 2009, Vol/35(4), pp.198-211. *DOI*: 10.1134/S0361768809040045

[20]. Bourdonov I.B., Kossatchev A.S. Testirovanie s preobrazovaniem semantic [Testing with Semantics Conversion] Trudy ISP RAN [The proceeding of ISP RAS], vol. 17, 2009, pp. 193-208. (in Russian)

[21]. A.Kossachev, I.Burdonov. Formal Conformance Verifcation, Short Papers of the 22nd IFIP ICTSS, Alexandre Petrenko, Adenilso Simao, Jose Carlos Maldonado (eds.), Nov. 08-10, 2010, Natal, Brazil, pp.1-6.

[22]. Bourdonov I.B., Kossatchev A.S. Interaction Semantics with Refusals, Divergence, and Destruction. Programming and Computer Software, vol. 36(5), 2010, pp. 247-263. *DOI*: 10.1134/S0361768810050014

[23]. Bourdonov I.B., Kossatchev A.S. Agreement between Conformance and Composition. Programming and Computer Software, vol. 39(6), 2013, pp. 269–278. *DOI*: 10.1134/S0361768813060029

Численное моделирование течения в канале с неглубокими лунками с использованием Code Saturne

[1]А.А. Цынаева <a.tsinaeva@rambler.ru >
[2]М.Н. Никитин <nikitin.pro@gmail.com>
[1]ФГБОУ ВПО СГАСУ, 443001, Россия, г. Самара, ул. Молодогвардейская,дом 194
[2]ФГБОУ ВПО СамГТУ, 443010, Россия, г. Самара, ул. Молодогвардейская,дом 221.

Аннотация. Работа посвящена построению модели и численному исследованию течения в прямоугольном канале с неглубокими лунками. Математическое моделирование выполняется на базе дифференциальных уравнений движения, энергии и состояния. Численное решение получено с использованием метода конечных объемов на базе программного пакета с открытым программным кодом Code Saturne. Построение сетки, учитывающей особенности течения вблизи лунок, выполнено в программном пакете Salome со свободной лицензией. В работе рассмотрены вопросы создания сетки, показана зависимость получаемого решения от размерности сетки, проведен анализ влияния типа сетки на время ее генерирования и качество получаемого численного решения для течения в прямоугольном канале с неглубокими лунками.

Ключевые слова: свободное программное обеспечение, численное моделирование, аэродинамика, течение, лунки.

DOI: 10.15514/ISPRAS-2016-28(1)-10

Для цитирования: Цынаева А.А., Никитин М.Н. Численное моделирование течения в канале с неглубокими лунками с использованием Code Saturne. Труды ИСП РАН, том 28, вып. 1, 2016 г..с. 185-196DOI: 10.15514/ISPRAS-2016-28(1)-10

1. Введение

Численное моделирование является перспективным методом исследования при разработке и проектировании новых образцов техники: теплообменных аппаратов, энергетических установок и другого оборудования. Однако профессиональный теплофизик не всегда является профессиональным программистом. Программные комплексы позволяют пользователю, имеющему узкоспециальный нематематический профиль подготовки, решать

поставленные перед ним технические задачи [1]. При этом коммерческие программные пакеты [2,3] для моделирования течения и теплообмена являются достаточно дорогостоящими и имеют существенные недостатки, обусловленные невозможностью изменения разработанного программного кода для решения конкретной задачи, неизменности используемых решателей. В этой связи, использование программных пакетов с открытым кодом предоставляет возможность для решения достаточно сложных научно-технических задач путем численного моделирования [4,5]. К таким задачам относится исследование течения в канале вблизи неглубоких лунок. В настоящее время известно, что использование неглубоких лунок для интенсификации теплообмена не приводит к значительному изменению гидравлического сопротивления поверхности [6,7,8]. Однако для оптимизации геометрии лунки с целью минимизации гидравлических потерь и интенсификации отвода (подвода) теплоты требуется проведение большого числа экспериментов по исследованию характеристик течения и теплообмена. Это требование может быть выполнено за счет проведения множества численных экспериментов с помощью программных пакетов с открытым кодом. Программные комплексы и интегрируемые платформы, обладающие графическим интерфейсом, например, Salome [9] и Code Saturne [10] могут быть более приемлемыми для решения пользователями инженерно-технических задач, нежели пакеты без графического интерфейса.

2. Формулировка задачи, расчетные сетки

В работе выполнено решение трехмерной задачи течения в прямоугольном канале с неглубокими лунками.

2.1 Математическая модель

Для численного исследования течения в канале с неглубокими лунками в качестве инструмента исследования выбран программный пакет Code Saturne с открытым кодом [10], в котором моделирование течения базируется на использовании осредненных по Рейнольдсу уравнений Навье-Стокса. Эти уравнения для ламинарного потока имеют вид:

$$\begin{cases} \dfrac{d\rho}{dt} + \operatorname{div}\left(\rho \underline{u}\right) = \Gamma \\ \dfrac{d\left(\rho \underline{u}\right)}{dt} + \underline{\operatorname{div}}\left(\underline{u} \otimes \rho \underline{u}\right) = -\underline{\nabla} P + \underline{\operatorname{div}}\left(\mu\left[\underline{\underline{\nabla u}} + \underline{\underline{\nabla u}}^{T} - \frac{2}{3} tr\left(\underline{\underline{\nabla u}}\right)\underline{\underline{Id}}\right]\right) + \rho \underline{g} + \underline{\underline{ST}}_{\underline{u}} - \underline{\underline{Ku}} + \Gamma \underline{u}^{in} \end{cases}, \quad (1)$$

где ρ – плотность; $\underline{u}$ – скорость; Γ – член уравнения, соответствующий источнику массы; ∇ – оператор Гамильтона; μ – динамическая вязкость; P – давление; $tr\left(\underline{\underline{.}}\right)$ – след тензора; $\underline{g}$ – ускорение свободного падения; $\underline{\underline{ST}}_{\underline{u}}$, $\underline{\underline{Ku}}$ –

явный и неявный источник, при этом $\underline{\underline{K}}$ симметричный положительный тензор.

Для турбулентного потока при $\underline{u} = \overline{\underline{u}} + \underline{u'}$ уравнения Навье-Стокса записываются в виде:

$$\begin{cases} \dfrac{d\rho}{dt} + \mathrm{div}\ \left(\rho\overline{\underline{u}}\right) = \Gamma \\ \rho\dfrac{d\overline{\underline{u}}}{dt} + \underline{\underline{\nabla}}\overline{\underline{u}}\cdot\left(\rho\overline{\underline{u}}\right) = -\underline{\nabla}P + \underline{\mathrm{div}}\left(\mu\left[\underline{\underline{\nabla}}\overline{\underline{u}} + \underline{\underline{\nabla}}\overline{\underline{u}}^{T} - \dfrac{2}{3}tr\left(\underline{\underline{\nabla}}\overline{\underline{u}}\right)\underline{\underline{Id}}\right]\right) + \rho\underline{g} - \underline{\mathrm{div}}\left(\rho\underline{\underline{R}}\right) + \\ \qquad + \underline{ST}_{\underline{u}} - \underline{\underline{K}}\underline{u} + \Gamma\left(\overline{\underline{u}}^{in} - \overline{\underline{u}}\right) \end{cases}, \quad (2)$$

здесь дополнительный член уравнения $\underline{\underline{R}} \equiv \overline{\underline{u'}} \otimes \underline{u'}$; $\mu = \mu_l + \mu_T$, μ_T – динамическая турбулентная вязкость; $\rho\underline{\underline{R}} = \dfrac{2}{3}\rho k\underline{\underline{1}} - 2\mu_T\overline{\underline{\underline{S}}}^{D}$, k – кинетическая энергия турбулентности $k \equiv \dfrac{1}{2}tr\left(\underline{\underline{R}}\right)$.

При моделировании течения из-за наличия особенностей поверхности (неглубоких лунок) применялась *k-ω sst* модель турбулентности [11]. Численное исследование получено в пакете программ Code Saturne [10] методом конечных объемов, в рамках которой уравнения интегрируются по каждой ячейке сетки (контрольный объем Ω_i).

2.2 Условия моделирования, характерные масштабы задачи

В качестве начальных условий задается поле скоростей на входе в канал ($\overline{\underline{u}}_{00} = 10$ м/с), свободное истечение потока на выходе из канала. Стенки канала заданы гладкими.

Канал длиной *L*=120 мм имеет прямоугольную форму с размерами сечения axb=70x20 мм. На одной из поверхностей канала выполнены неглубокие лунки (глубиной $h = 1{,}2$ мм), расположенные в три ряда по шесть лунок в каждом.

Численное исследование течения в канале с неглубокими лунками характеризуется следующими размерными параметрами: рабочее тело – газ, (при давлении 101325 Па имеет $\rho = 1{,}205$ кг/м^3, $\mu = 1{,}72\cdot10^{-5}$ м^2/с; $c_P = 1005$ Дж/кг), ускорение свободного падения $\underline{g} = 9{,}81$ м/с^2. При моделировании физические свойства приняты постоянными, так как рассматривалась задача газодинамики потока. Вышеперечисленные параметры формируют характерные масштабы времени, скорости, вязкости, длины, отношение которых дает характерные критерии задачи, например, критерий Рейнольдса $\mathrm{Re}_l = \overline{\underline{u}}\cdot l/\upsilon$, ($l$ – гидравлический диаметр

прямоугольного канала, м; υ – кинематическая вязкость, $м^2/с$). Для рассматриваемого прямоугольного канала, имеющего лунки на одной из внутренних поверхностей, l=0,042 м, $\mathrm{Re}_l = 31626{,}5$.

2.3 Дискретизация области исследования

Для дискретизации расчетной области использовалась открытая интегрируемая платформа Salome [9]. Так как, вблизи стенки канала с неглубокими лунками формируются вихревые структуры, обусловленные рельефом поверхности, то с целью повышения качества решения сетка выполнялась структурированной по поверхности канала с формированием элементов для моделирования пограничного слоя. Минимальный размер ячейки определялся из условия размещения нескольких параллельных рядов ячеек расчетной сетки по глубине лунки для наилучшего воспроизведения особенностей течения в ней.

Рис. 1. показывает зависимость времени t_{gen} (в секундах) дискретизации расчетной области от количества ячеек N для неструктурированной и структурированной сеток.

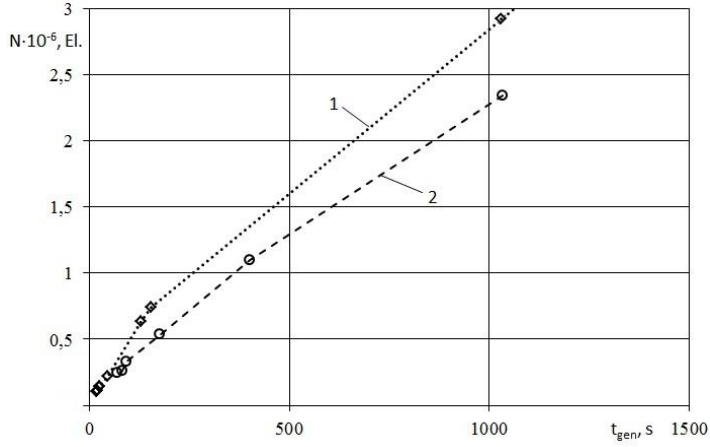


Рис. 1. Характеристика дискретизации расчетной области: 1 – неструктурированная сетка; 2 – структурированная сетка
Fig. 1. Discretisation characteristics of computational region: 1 - unstructured grid; 2 - structured grid

Для исследуемой расчетной области, согласно (рис. 1), время построения структурированной сетки при том же количестве ячеек несколько выше, чем неструктурированной сетки. При этом структурирование сетки позволяет выполнять более качественную послойную дискретизацию расчетной области вблизи поверхности стенок канала. Толщина структурного слоя выбиралась

соразмерной глубине (h=1,2 мм) лунок и составляла 2 мм при разбиении этого слоя на 6 подслоев.

Для дискретизации граничных условий применялся инструментарий пакета Code Saturne [10].

Code Saturne представляет собой программный комплекс, имеющий графический интерфейс, с инструментарием для численного решения задач гидрогазодинимики и тепломассообмена. Следует отметить наличие возможности реализации параллельных вычислений с определением способа разделения расчетного домена между процессорами. В качестве граничных условий в Code Saturne могут быть заданы условия Дирихле и Неймана. В Code Saturne для разрешения поля скорости использована схема второго порядка (SOLU), для энергии турбулентных пульсаций – 1-го порядка (Upwind), для разрешения поля давления выбран решатель Multigrid, принятое максимальное число итераций по каждому циклу составляло 10000, точность решателя (Solver Precision) выбрана равной 10^{-8}. Для получения решения применен итерационный решатель с шагом по времени 0,0001 сек. с количеством итераций 2000.

3. Проверка адекватности

Проверка адекватности полученного решения выполнялась посредством сравнения результатов численного расчета, выполненного в программном комплексе Code Saturne [10] с экспериментальными данными, представленными в работах Института теплофизики СО РАН и Института гидромеханики НАН Украины [12,13,14]. Экспериментальные исследования течения в прямоугольном канале с лункой [12,13,14] были проведены независимо друг от друга для сходной геометрии канала и лунки, отличие заключалось в используемых средствах измерения. Прямоугольный канал шириной 0,2 м и высотой 0,015 м с длиной 1,34 м выполнен с одиночной лункой с острыми кромками диаметром D=0,046 м, расположенной на оси канала на расстоянии равном l_1~11D. Рабочей средой являлась несжимаемая жидкость – вода, скорость жидкости на входе в канал составляла 0,43 м/с. Течение в канале турбулентное, значение $\mathrm{Re}_D = \bar{\underline{u}} \cdot D/\upsilon = 20000$. Численное моделирование было выполнено с условием симметрии по оси лунки для сокращения времени расчета. Такое допущение оказалось возможным на основе анализа экспериментальных данных. Для моделирования использовалась система уравнений (2), для замыкания задачи турбулентного течения применена *k-ω sst* модель турбулентности [11]. Дискретизация расчетной области осуществлена в Salome [10], сетка выполнена структурированной, вблизи стенок канала сгенерированы 6 параллельных слоев, толщиной 0,5 мм. Для дискретизации трехмерных элементов выбрана гипотеза Tetrahedron(Netgen), для двумерных элементов алгоритм Netgen 1d-2d с минимальным размером ячейки 0,7 мм и максимальным размером 2,5 мм. В результате дискретизации получена сетка из 878977 объемных элементов.

Дискретизация начальных и граничных условий выполнялась средствами Code Saturne [10]. Для расчета давления, скорости и энергии турбулентных пульсаций применены описанные выше схемы и решатели. Задача решалась в стационарной постановке. Результаты численного расчета представлены для коэффициента давления вблизи поверхности, рассчитанного в соответствии с работой Терехова В.И.и др. [12] по формуле:

$$C_p = \left(p_i - p_0\right) / \left(\rho \bar{u}_S^2 / 2\right) \quad (3)$$

где p_i – текущее значение давления; p_0 – давление у верхней по потоку кромки лунки; ρ – плотность воды; $\bar{u}_S$ – средняя скорость потока воды для тестового эксперимента. Результаты сравнения численного и физического эксперимента с целью проверки адекватности представлены на рис. 2.

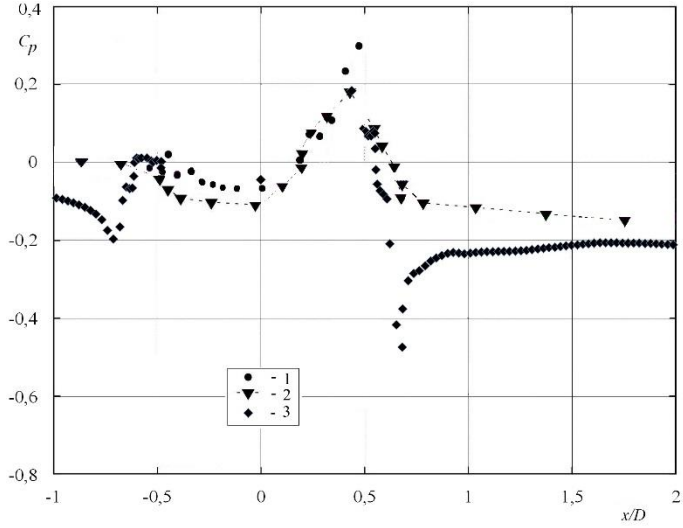


Рис. 2. Распределение давления на обтекаемой поверхности в окрестности сферической лунки: 1 – эксперимент Терехова В.И. и др. [12]; 2 – эксперимент Воскобойника В.А. и др. [13,14]; 3 – расчет авторов в Code Saturne
Fig. 2. Pressure distribution at streamlined surface in a vicinity of the dimples: 1 - experiment of V.I. Terekhov and others. [12], 2 - experiment of V.A. Voskoboynik and others. [13,14]; 3 - calculation of the authors with Code Saturne

Анализируя, представленные на рис. 2 данные для параметров вблизи поверхности стенки, следует отметить качественное совпадение экспериментальных [12,13,14] и расчетных данных. Имеющиеся отличия незначительны, но наблюдаются в узкой области непосредственно за одиночной лункой. На рис. 3 представлена картина распределения статического давления в одиночной лунке.

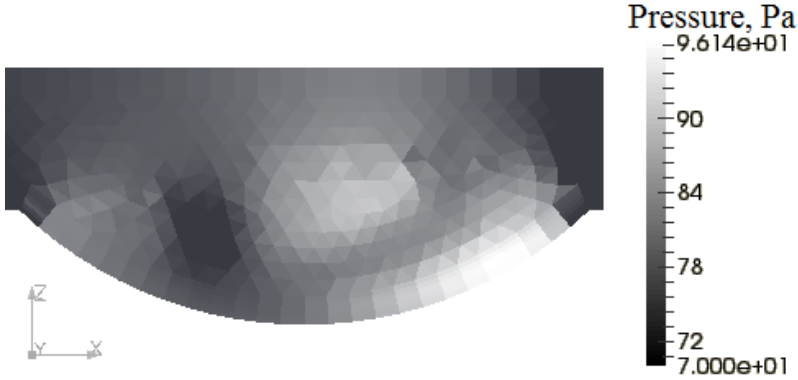


Рис. 3. Распределение статического давления в лунке по расчету в Code Saturne
Fig. 3. Distribution of static pressure in the dimple on the calculation with the Code Saturne

Анализ картины течения показывает наличие вихревых структур в лунке, что показано в работах [6-8,12-15]. В связи вышесказанного, был сделан вывод об адекватности выбранного метода и инструмента численного исследования.

4. Результаты численного решения и обсуждение

Численное исследование проводилось для двух основных типов расчетной области: прямоугольного канала с тремя рядами сферических лунок и того же канала с гантелеобразными лунками. Целью численного моделирования являлся анализ параметров течения в канале с гантелееобразными трехмерными лунками. Для проведения сравнения параметров течения для каналов с различными лунками необходимо выполнить следующие условия: одинаковая глубина (h=1,2 мм) лунок исследуемой конфигурации и сферических лунок, равные площади «пятна» исследуемых лунок (S_p=59,76 мм^2).

Первым этапом исследования явилось определение влияния качества расчетной сетки на продолжительность расчета и качество получаемого решение. Анализ проводился для канала со сферическими лунками. Согласно полученным данным (рис. 4), продолжительность расчета t_c определяется не только количеством элементов сетки, но и типом поверхностного элемента разбиения. На рис. 5 представлены результаты расчета скорости в зависимости от качества дискретизации расчетной области; профиль скорости построен для течения на расстоянии l_1=6,25D; линии 1,3 соответствуют оси канала; 2,4 – построены для пристеночной области, секущая плоскость на расстоянии 1 мм от стенки с лунками. Анализ результатов (рис. 5) показал значительные флуктуации скорости потока в пристеночной области при малом N_{el}, которые носят случайный характер. В связи с чем, для проведения дальнейших исследований выбрана сетка с N_{el}=766796 объемными элементами.

На рис. 6 представлены результаты численного исследования для канала со сферическими и гантелеобразными лунками.

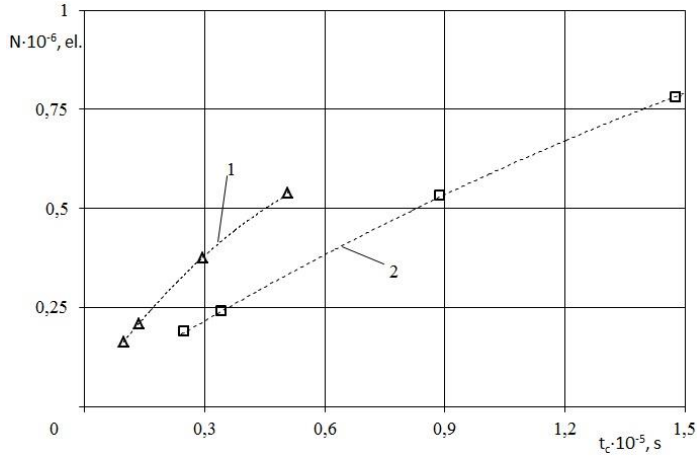


Рис. 4. Зависимость продолжительности расчета от параметров дискретизации расчетной области: 1 – сетка по поверхностям выполнена в виде треугольных элементов; 2 – сетка по поверхностям выполнена в виде четырехугольных элементов
Fig. 4. Dependency of calculation duration on discretisation parameters of calculation region: 1 – grid on a surfaces is formed as triangular elements; 2 – grid on surfaces is formed as a quadrangle elements

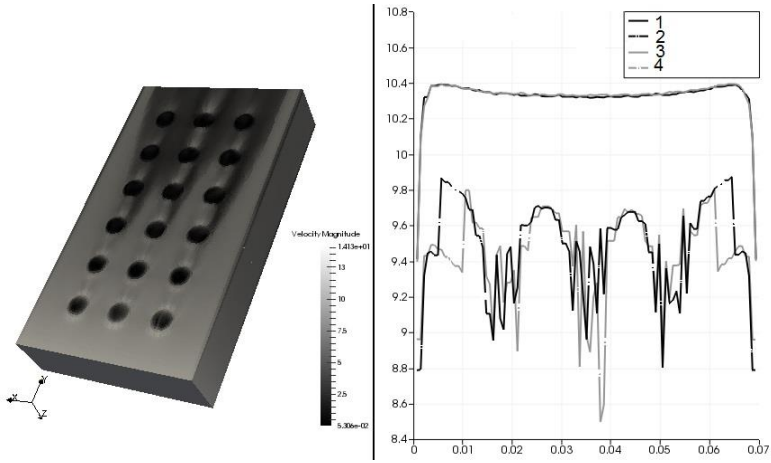
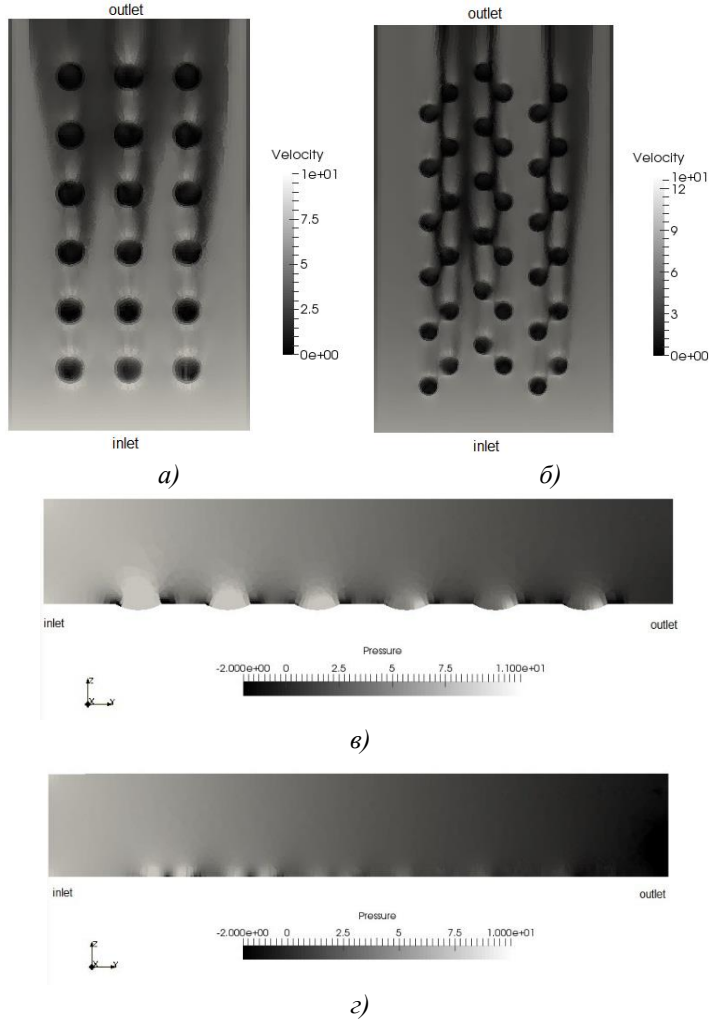


Рис. 5. Влияние качества дискретизации расчетной области на результаты расчета: 1, 2 – профиль скорости на оси канала и вблизи поверхности с лунками при N_{el}=766796; 3,4 – тоже при N_{el}=537459
Fig. 5. Impact of the quality of discretisation of the computational domain to results: of calculation, 1, 2 – velocity profile on the channel axis and near surface with dimples at Nel = 766,796; 3.4 – the same when Nel = 537459



а) *б)*

в)

г)

Рис. 6. Результаты численного моделирования: а, б – поле скоростей; в – поле статического давления в осевом сечении канала со сферическими лунками; г – тоже с гантелеобразными лунками

Fig. 6. The results of numerical modeling: a, b - the velocity field; a - static pressure field in the axial channel section with dimples; g - the same hole with dumbbell dimples

Полученные результаты показали существенную зависимость скорости течения от угла наклона гантелеобразной лунки к направлению потока. Выявлено образование области с пониженными скоростями, в которых совместное влияние гантелеобразных лунок ведет к ламинаризации потока вблизи стенки. Анализ поля давления для канала с гантелеобразными лунками

(рис. 6) показал, что гидравлическое сопротивление канала до 10 % ниже, чем для канала со сферическими лунками.

5. Заключение

Проведено численное моделирование течения в канале с неглубокими лунками на базе открытого пакета Code Saturne. Отработана процедура построения качественной расчетной сетки, учитывающей особенности течения вблизи поверхности с лунками. Проведена проверка адекватности разработанной модели путем сравнения результатов численного и физического эксперимента, показавшая адекватность выбранных методов и инструментов численного исследования.

В результате моделирования выявлена зависимость характеристик течения в канале от угла наклона гантелеобразной лунки к направлению течения. В результате анализа полученных данных выявлено, что гидравлическое сопротивление канала со сферическми лунками несколько выше, чем канала с гантелеобразными лунками.

Список литературы

[1]. А. Леонтьев, Н. Пилюгин, Ю. Полежаев, В. Поляев. Научные основы технологий XXI века. Москва, Энергомаш, 2000. 136 с.
[2]. Страница пакета Ansys — http://www.ansys.com/
[3]. Страница пакета CATIA — http://www.3ds.com/
[4]. Страница 11th. World Congress on Computational Mechanics (WCCM XI) — http://www.wccm-eccm-ecfd2014.org/frontal/default.asp
[5]. А. Цынаева, Е. Цынаева. Моделирование задач теплообмена и гидрогазодинамики с помощью свободного программного обеспечения. Вестник Ульяновского государственного технического университета, №4, 2014 г. стр. 42-45.
[6]. S. Isaev, A. Guzeev, S. Sapozhnikov, V. Mityakov, A. Mityakov. Visualization of a Flow in a Spherical Dimple Built in the Lower Wall of the Rectangular-Section Channel of a Water Tunnel and Numerical Identification of the Vortex-Jet Structures in It. Journal of Engineering Physics and Thermophysics, vol 88(2), 2015. P. 438-454. doi:10.1007/s10891-015-1210-x
[7]. С. Исаев, А. Леонтьев, Н. Корнев, Э. Хассель, Я. Чудновский. Интенсификация теплообмена при ламинарном и турбулентном течении в узком канале с однорядными овальными лунками, Теплофизика высоких температур, том 53, №3, 2015 г., стр. 390-402. doi:10.7868/S0040364415030060
[8]. G. Kiknadze , I. Gachechiladze , T. Barnaveli Jr. The mechanisms of the phenomenon of torn ado-like jets self-organization in the flow along the dimples on the initially flat surface. *Proceedings of the ASME 2012 International Mechanical Engineering Congress & Exposition IMECE*, 2012. pp. 3017-3026. doi: 10.1115/IMECE2012-93581
[9]. Страница инструмента Salome — http://www.salome-platform.org/
[10]. Страница инструмента Code Saturne — http://code-saturne.org/cms/
[11]. Страница Langley Research Center: Turbulence Modeling Resource — http://turbmodels.larc.nasa.gov/sst.html

[12]. V. Terekhov, S. Kalinina, and Yu. Mshviobadze. Heat Transfer Coefficient and Aerodynamic Resistance on a Surface with a Single Dimple. Journal of Enhanced Heat Transfer, 1997, vol. 4, pp. 131 -145. doi: 10.1615/JEnhHeatTransf.v4.i2.60

[13]. В. Воскобойник. Распределение давления на обтекаемой поверхности со сферической лункой. Водний транспорт, том 3, 2014 г. Стр.90-96.

[14]. V. Voskoboinick, N. Kornev, J. Turnow. Study of near wall coherent flow structures on dimpled surfaces using unsteady pressure measurements. Flow, turbulence and combustion, № 90(4), 2013, pp 709-722. doi: 10.1007/s10494-012-9433-9

[15]. Г. Коваленко, А. Халатов. Границы режимов течения в углублениях на плоской поверхности, имеющих форму сферических сегментов. Прикладна гідромеханіка, том 10, № 1, 2008 г. Стр. 23-32.

Numerical modeling of rectangular channel with shallow dumbbell dimples based Code Saturne

[1]A. Tsynaeva < a.tsinaeva@rambler.ru >
[2]M. Nikitin < nikitin.pro@gmail.com >
[1]SSUACE, 194 Molodogvardeiskaya Str., Samara, 443001, Russian Federation
[2]SamGTU, 224 Molodogvardeiskaya Str., Samara, 443010, Russian Federation

Abstract. Numerical study was conducted for rectangular channel with dimples. Developed model was tested for adequacy by simulating of experiment, conducted by Dr. Terekhov, which was found in good agreement. Experimental setup utilized a single spherical dimple which was set at 11 diameters from inlet. Test simulation was conducted for incompressible fluid (water) in accordance with experiment conditions: inlet velocity 0.43 m/s, Reynolds for dimple 20000 and channel length 1.34 m. A 3D computation domain was meshed for 0.8 million elements with six viscous layers totalling 3 mm thick applied to smooth walls. A turbulent flow (Re = 31627) in rectangular channel with shallow dumbbell dimples was modelled with open source Code_Saturne. An ideal gas (ρ = 1.205 kg/m^3) was considered as working medium. A 3D computation domain was meshed with open source Salome Meca for 0.77 million elements ranged 0.2...1.0 mm. Six viscous layers totalling 2 mm thick were applied to smooth walls. Unsteady flow simulated with k-w SST model utilizing 2nd order discretization schemes (SOLU) for velocity. 2000 iterations were calculated so far with pseudo time step of 0.1 ms. Additionally impact of mesh quality regarding elements size on computation results was shown. Generation time of mixed mesh (quadrangles and triangles on surface) was proved to be greater than of strictly triangular one. Obtained results showed a strong dependence of flow velocity from inclination of dumbbell towards flow axis. Adjacent dumbbell dimples cause partial flow laminarization. Developed model shows aerodynamic advantage up to 10 % of dumbbell dimples over spherical ones of the same depth (h = 1.2 mm) and contact patch area (S = 59.76 mm^2).

Keywords: free software, numerical simulation, aerodynamics, flow, dimples.

DOI: 10.15514/ISPRAS-2016-28(1)-10

For citation: Tsynaeva A., Nikitin M. Numerical modeling of rectangular channel with shallow dumbbell dimples based Code Saturne. Trudy ISP RAN /Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 185-196 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-10

References

[1]. A. Leont'ev, N. Pilyugin, Yu. Polezhaev, V. Poljaev. Nauchnye osnovy tekhnologiy XXI veka [Scientific basis of technologies in XXI century]. Moscow, Energomash, 2000. 136 s. (In Russian)

[2]. Page of software Ansys — http://www.ansys.com/

[3]. Page of software CATIA — http://www.3ds.com/

[4]. Page of 11th. World Congress on Computational Mechanics (WCCM XI) — http://www.wccm-eccm-ecfd2014.org/frontal/default.asp

[5]. A. Tsynaeva, E. Tsynaeva. Modelirovanie zadach teploobmena i gidrogazodinamiki s pomow'ju svobodnogo programmnogo obespechenija [Modelling of heat transfer and fluid dynamics problems using free software]. Vestnik Ul'janovskogo gosudarstvennogo tekhnicheskogo universiteta [Journal of Ulyanovsk State Technical University], no. 4, 2014. pp.42-45. (In Russian)

[6]. S. Isaev, A. Guzeev, S. Sapozhnikov, V. Mityakov, A. Mityakov. Visualization of a Flow in a Spherical Dimple Built in the Lower Wall of the Rectangular-Section Channel of a Water Tunnel and Numerical Identification of the Vortex-Jet Structures in It. Journal of Engineering Physics and Thermophysics, vol 88(2), 2015. pp. 438-454. doi:10.1007/s10891-015-1210-x

[7]. S. Isaev, A. Leont'ev, N. Kornev, Je. KHassel', Ya. Chudnovskij. Intensifikacija teploobmena pri laminarnom i turbulentnom techenii v uzkom kanale s odnorjadnymi oval'nymi lunkami [Heat transfer intensification for laminar and turbulent flows in a narrow channel with one-row oval dimples], Teplofizika vysokikh temperature [High Temperature], vol. 53, no. 3, 2015. pp. 390-402. doi:10.7868/S0040364415030060 (In Russian)

[8]. G. Kiknadze , I. Gachechiladze , T. Barnaveli Jr. The mechanisms of the phenomenon of torn ado-like jets self-organization in the flow along the dimples on the initially flat surface. *Proceedings of the ASME 2012 International Mechanical Engineering Congress & Exposition IMECE*, 2012. pp. 3017-3026. doi: 10.1115/IMECE2012-93581

[9]. Page of software Salome — http://www.salome-platform.org/

[10]. Page of software Code Saturne — http://code-saturne.org/cms/

[11]. Page of Langley Research Center: Turbulence Modeling Resource — http://turbmodels.larc.nasa.gov/sst.html

[12]. V. Terekhov, S. Kalinina, and Yu. Mshviobadze. Heat Transfer Coefficient and Aerodynamic Resistance on a Surface with a Single Dimple. Journal of Enhanced Heat Transfer, 1997, vol. 4, pp. 131 -145. doi: 10.1615/JEnhHeatTransf.v4.i2.60

[13]. V. Voskobojnik. Raspredelenie davlenija na obtekaemoj poverkhnosti so sfericheskoj lunkoj [Pressure distribution on streamlined surface with spherical dimple]. Vodnij transport [Water transport], vol. 3, 2014. pp.90-96. (In Ukrainian)

[14]. V. Voskoboinick, N. Kornev, J. Turnow. Study of near wall coherent flow structures on dimpled surfaces using unsteady pressure measurements. Flow, turbulence and combustion, № 90(4), 2013, pp 709-722. doi: 10.1007/s10494-012-9433-9

[15]. G. Kovalenko, A. Khalatov. Granicy rezhimov techenija v uglublenijakh na ploskoj poverkhnosti, imejuwikh formu sfericheskikh segmentov [The boundaries of flow regimes in the pits on a flat surface in the shape of spherical segments.]. Prikladna gidromekhanika [Applied Fluid Mechanics], vol. 10, no 1, 2008. pp. 23-32. (In Ukrainian)

MHD supersonic flow control: OpenFOAM simulation

A.I. Ryakhovskiy <alexey.i.ryakhovskiy@mail.ioffe.ru>
A.A. Schmidt <alexander.schmidt@mail.ioffe.ru>
Ioffe Institute, 26 Politekhnicheskaya Str., St Petersburg, 194021, Russian Federation

Abstract. MHD flow control is a relevant topic in today's aerospace engineering. An OpenFOAM density-based solver that is capable of handling MHD supersonic flow problems with constant magnetic field is developed. The proposed solver is based on Balbas-Tadmor central difference schemes. This solver can be applied to studying the potential of MHD flow control systems for atmospheric entry vehicles. A supersonic flow around a spherically blunt cone both with and without MHD interaction is studied. Gases with thermodynamic parameters characteristic for Earth's and Martian atmospheres are considered. The results show visible effect of magnetic field on surface temperature of the body. The differences between shock standoff distances and general shockwave configurations of MHD and non-MHD flow are also apparent. The solution is stable for Stuart number below 0.2. Conditional instability of the solver can be attributed to the MHD term's contribution to the local speed of sound and can be avoided by taking it into account. The developed application has proven the suitability of the used schemes for resolving steep gradients in MHD supersonic flow problems. The study itself has shown theoretical possibility of studying the MHD flow control using OpenFOAM. Further research may include an effort to stabilize the solver and to enhance the mathematical model of the flow.

Keywords: magnetohydrodynamics, finite volume method in fluid dynamics, supersonic flows, numerical simulations, shock waves in fluid dynamics

DOI: 10.15514/ISPRAS-2016-28(1)-11

For citation: Ryakhovskiy A.I., Schmidt A.A. MHD supersonic flow control: OpenFOAM simulation. Trudy ISP RAN /Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 197-206 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-11

1. Introduction

Non-mechanical ways to control the bow shock is a relevant problem in today's aerospace engineering. Heavy dynamic and thermal loads that affect the aircrafts and space vessels during atmospheric reentry or traveling with a supersonic velocity can cause severe damage and even destruction. Aerodynamic form of probes and vessels can help minimize the damaging effects, but mere mechanical means are not

always sufficient. Different approaches to solve this problems were proposed over the years, one of the most promising of which is magnetohydrodynamic (MHD) flow control. The idea behind it is using a generated magnetic field that would interact with ionized gas in front of the vehicle and alter the flow around it in such a way that the aerodynamic heating and stress effects are diminished. This concept has many applications, such as MHD heat shield [1] or non-mechanical flight trajectory control systems [2]. A sufficient degree of ionization can be achieved both naturally by heating of the gas, or artificially, using electron beam [3] or similar system. Since experiments, involving this kind of apparatus is extremely expensive, mathematical modeling becomes an important research tool. Commercial CFD packages often lack proper instruments, extensibility and flexibility required to model such a phenomenon. That's why we opted to use OpenFOAM numerical simulation platform. We set to numerically investigate the supersonic around the body with magnetic interaction, for which a new solver needs to be developed. If developed successfully, such a solver can prove useful for any research related to MHD supersonic flow.

2. Physical formulation

We investigate the flow around a spherically blunted cone. This shape is characteristic for various reentry vehicles as it provides flight stability with relatively low aerodynamic heating. We do not include an afterbody of any kind into our geometry, since the region of interest for us is in the front, where magnetic interaction takes place. Cone base has a radius of 1m, and its vertex is spherically blunted with a curvature radius of 10 cm. The field generating coil is located in the frontal part of the body and is set to generate a magnetic field of 1 Tesla directly in front of the body. The opening angle of the cone is 120 degrees.

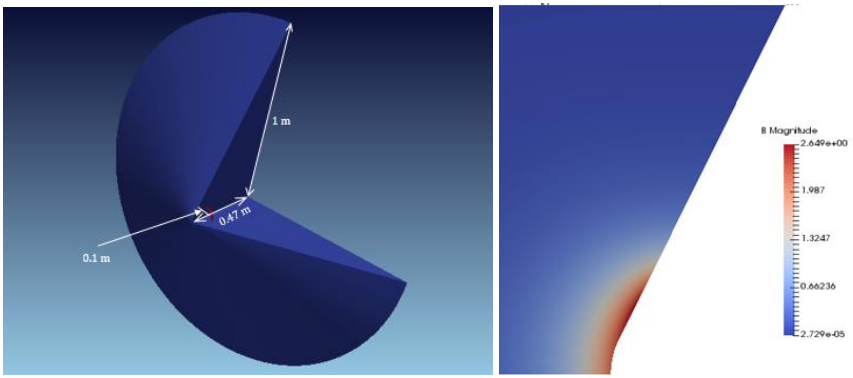


Fig. 1. The geometry of the body with a section removed (left) and the distribution of the magnetic field (right).

Thermophysical properties of the surrounding gas correspond to those of upper and mid layers of Earth and Mars atmosphere. The table below (tab. 1) shows those properties as well as Mach number corresponding to the velocity of 1000 m/s in each case. Values for Martian atmosphere were taken from [4]. Atmospheric properties of the 2 planets differ dramatically, so the pressure on the same altitude can be a hundred times higher on the Earth than it is on the Mars. The velocity of incoming flow is ranging between 1000 and 2000 m/s, so that the Mach number is between 3 and 5. The electrical conductivity of the gas is set to be constant, it's value - 80 $Ohm \cdot m$. This value can be realistically obtained by heating, further increase may require an auxiliary ionizing system, such as electron beam. Constant conductivity can be justified by the fact that magnetic field magnitude fades as a square of the distance from the coil, so the conductivity effect far from the body can be neglected.

Tab. 1. Thermophysical parameters

	Earth	***Mars***
Gas	Air	CO_2
Molar weight (g/mol)	28.97	44.01
$\boldsymbol{p_\infty}$**(Pa)**	3000	30
$\boldsymbol{t_\infty}$**(K)**	216.5	190
$\boldsymbol{Mach}|_{\boldsymbol{u=1000\frac{m}{s}}}$	3.37	4.46

3. Mathematical model

To model the flow we use magnetohydrodynamic (MHD) approximation. The set of governing equations in our case consists of conservation laws for mass, momentum and energy. The equations of electromagnetic field are omitted, since we consider the field to be constant. Low magnetic Reynolds number ($Re_{mag} \ll 1$) allows us to disregard velocity's effect on magnetic field. In our case it is of order 10^{-1}. Knudsen number for the studied flows ranges from 10^{-5} to 10^{-3}, which is well within the area of continuous model applicability. MHD terms values are determined by the generalized Ohm's law and the respective formulas.

$$\boldsymbol{j} = \sigma(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}), \qquad \boldsymbol{F} = \boldsymbol{j} \times \boldsymbol{B}, \qquad \boldsymbol{Q} = \boldsymbol{j} \cdot E.$$

Thus, the MHD system for our case can be written as follows:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0,$$

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot \left[\rho \mathbf{v}\mathbf{v} + \left(p + \frac{1}{2}B^2\right) I_{3\times 3} - \mathbf{B}\mathbf{B}\right] = 0,$$

$$\frac{\partial}{\partial t}\left(\frac{1}{2}\rho v^2 + \rho e + \frac{1}{2}B^2\right) + \nabla \cdot \left[\left(\frac{1}{2}\rho v^2 + \rho e + p + B^2\right)\mathbf{v} - \mathbf{v} \cdot \mathbf{B}\mathbf{B}\right] = 0.$$

3.1 Case geometry and boundary conditions

We take the advantage of the cylindrical symmetry of our problem and pose it in 2 dimensions. In OpenFOAM this can be achieved by making the computational domain a sector of a cylinder and setting "wedge" boundary conditions on its' sides. The remaining boundary conditions are "outlet" and "inlet" for the respective surfaces, "zeroGradient" condition for upper boundary and a specific set for the surfaces representing the body. Velocity condition on the body surface is "slip". Temperature condition on this boundary may vary, since we try to estimate the heat flux towards the body without knowing in thermophysical properties. We opted to set flux conditions rather than temperature. This way we can estimate the flux between the body and surrounding gas by the resulting temperature.

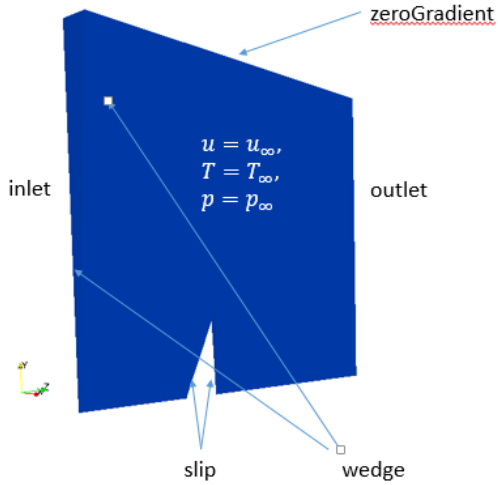


Fig. 2. Computational domain shape and boundary conditions

4. Solver development

OpenFOAM has a considerable range of capabilities in modeling the hypersonic flow, however not all of them are viable for our case, even without the imposed magnetic field. Pressure-based solvers like sonicFOAM tend to have so called overshoots in the solution they provide for pressure profile between the body and bow shock. The best performing density-based solver is rhoCentralFOAM, however it also has its issues, since it requires computational grid to fit no less than about 30 cells into the aforementioned layer between the bow shock and the body to properly calculate its parameters.

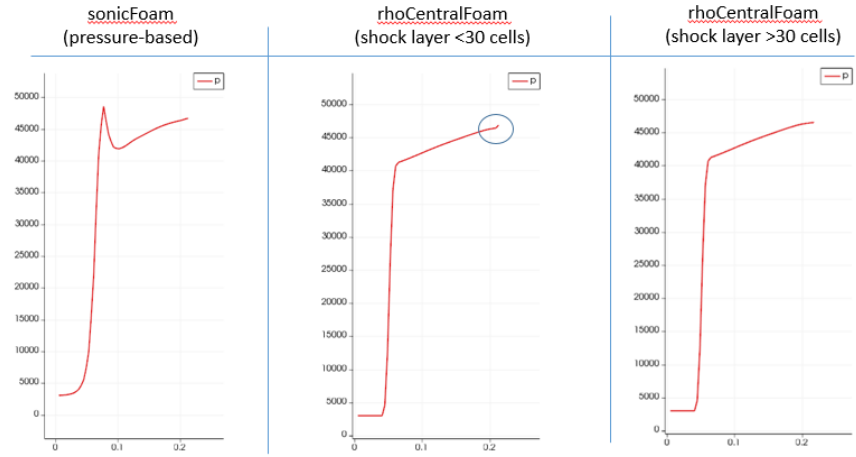


Fig. 3. Standard OpenFOAM solvers capabilities in resolving bow shock

Attempts have been made in the past to develop an MHD solver for OpenFOAM, some of which were relatively successful [5]. The proposed solvers, however, are based on outdated now methods and have stability problems.

We developed a solver that can resolve bow shock problems in presence of constant magnetic field by modifying *rhoCentralFoam*. *rhoCentralFoam* uses central difference schemes of Kurganov and Tadmor [6]. Our modification is based on the MHD extension of those schemes, proposed by Balbas and Tadmor [7]. Since there is no need at the moment to add magnetic field equations, we can simply include the MHD terms to the equations that already are in *rhoCentralFoam*, along with the procedures to interpolate and reconstruct the additional fluxes. Balbas and Tadmor schemes are expected to perform well on resolving the steep gradient fields that are characteristic for problems with bow shock.

5. Results

Our developed solver was tested on the problem of MHD controlled flow over the spherically blunted cone. The performance varied depending on the Stuart number: the solver is stable when the Stuart number is below 0.2, instabilities start to occur when it exceeds this value. Distributions of thermodynamical parameters were obtained as a result.

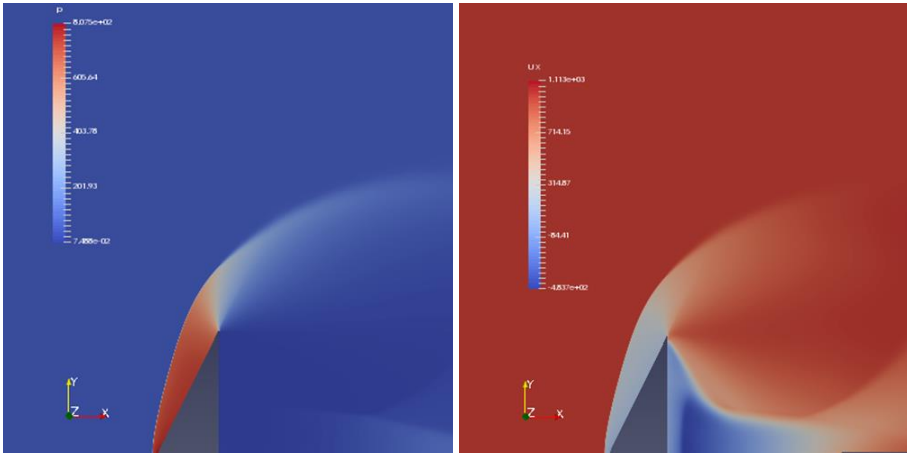


Fig. 4. Pressure (left) and longitudinal velocity (right) calculated for MHD case.

One of the main goals of this research is the development of an OpenFOAM solver that is capable of resolving bow shock with magnetic interaction. Our solver's stability is restricted to the low Stuart number flows, when the Stuart number exceeds 0.2-0.25 in the large region of the domain, the solution tends to diverge.

To illustrate the solvers capabilities and the difference between the MHD and non-MHD flow we considered the temperature and pressure profiles on the front surface of the body. Figures below show the discrepancy between the two cases: the temperature is lowered overall and especially in the regions where the magnetic field magnitude is greatest.

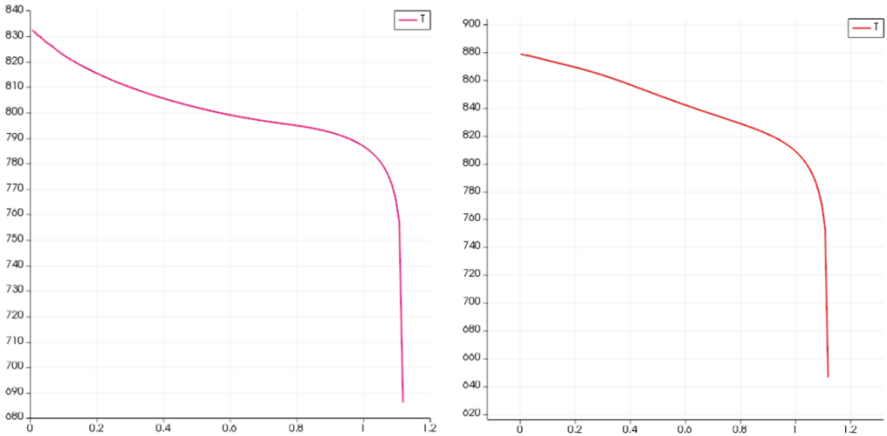


Fig.5. Front surface temperature for MHD (left) and non-MHD (right) flows

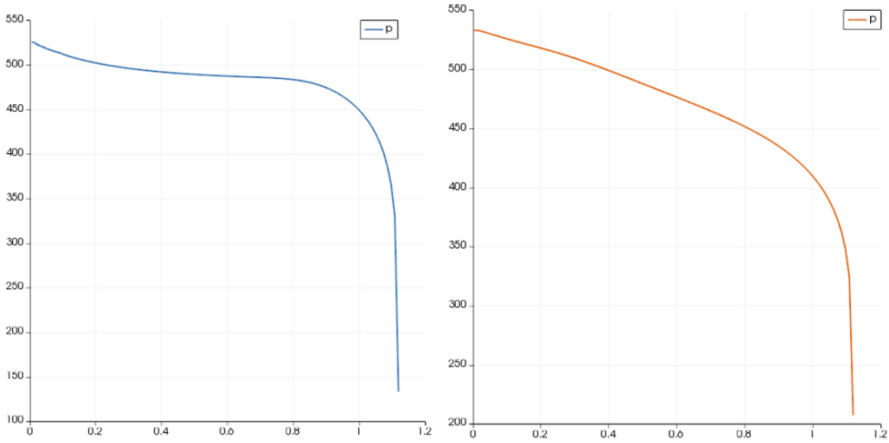


Fig.6. Front surface pressure for MHD (left) and non-MHD (right) flows

Another observed result is an increase of the shockwave standoff distance in presence of the magnetic field. Overall results can be said to be fitting the qualitative theoretical expectations. To compare them quantitatively additional experimental data is required.

Conclusion

We have developed a solver that is capable of resolving the MHD supersonic flow problems with constant magnetic field, such as MHD flow control around a blunted body, within the limited range of Stuart numbers. The results can be considered as a proof of suitability of Balbas-Tadmor schemes for simulation of the MHD supersonic flow. Further research should include the enhancement of the solver: widening the range of problems it can be applied to and improvement of its stability. Other approaches to the solution of MHD problems (mainly Godunov-type schemes) should also be considered. Finally, further investigation into the phenomenon will require developing an OpenFOAM thermophysical model that includes the relevant properties, such as conductivity and ionization.

References

[1]. Kantrowitz, A. R. "A survey of physical phenomena occurring in flight at extreme speeds." Proceedings of the Conference on High-Speed Aeronautics. New York: Polytechnic Inst. of Brooklyn, 1955.

[2]. Ericson, William B., and A. Maciulaitis. "Investigation of magnetohydrodynamic flight control." Journal of Spacecraft and Rockets 1.3 (1964): 283-289.

[3]. Macheret, Sergey O., Mikhail N. Shneider, and Richard B. Miles. "Magnetohydrodynamic control of hypersonic flows and scramjet inlets using electron beam ionization." AIAA journal 40.1 (2002): 74-81.

[4]. Forget, Francois, et al. "Density and temperatures of the upper Martian atmosphere measured by stellar occultations with Mars Express SPICAM."Journal of Geophysical Research: Planets (1991–2012) 114.E1 (2009).
[5]. Xisto, Carlos M., et al. "Implementation of a 3D compressible MHD solver able to model transonic flows." Proc. ECCOMAS CFD 2010-V European Conference on Computational Fluid Dynamics. 2010.
[6]. Kurganov, Alexander, and Eitan Tadmor. "New high-resolution central schemes for nonlinear conservation laws and convection–diffusion equations." Journal of Computational Physics 160.1 (2000): 241-282.Bisek N. J., Poggie J., Boyd I. D. Numerical study of a MHD-heat shield
[7]. Balbás, Jorge, Eitan Tadmor, and Cheng-Chin Wu. "Non-oscillatory central schemes for one-and two-dimensional MHD equations: I." Journal of Computational Physics 201.1 (2004): 261-285.

Численное моделирование МГД управления сверхзвуковым потоком в среде OpenFOAM

А.И. Ряховский <alexey.i.ryakhovskiy@mail.ioffe.ru>
А.А. Шмидт < alexander.schmidt@mail.ioffe.ru>
ФТИ им. Иоффе, 194021, Россия, г. Санкт-Петербург,
ул. Политехническая, дом 26

Аннотация. МГД управление сверхзвуковым потоком является важной проблемой в современных аэрокосмических исследованиях. На основе центральных разностных схем Балбаса и Тадмора был разработан OpenFOAM-солвер, способный решать задачи моделирования сверхзвукового МГД потока. С его помощью была решена задача обтекания твердого тела в форме сферически затупленного конуса. Получаемое решения оказывается устойчивым для потоков с числом Стюарта не более 0.2. В дальнейшем планируется улучшение устойчивости солвера и совершенствование математической модели.

Ключевые слова: магнитная гидродинамика, сверхзвуковые течения, численное моделирование, метод конечных элементов в гидрогазодинамике, численное моделирование, ударные волны в газодинамике.

DOI: 10.15514/ISPRAS-2016-28(1)-11

Для цитирования: Ряховский А.И., Шмидт А.А. Численное моделирование МГД управления сверхзвуковым потоком в среде OpenFOAM. Труды ИСП РАН, том 28, вып. 1, 2016 г..с. 197-206. DOI: 10.15514/ISPRAS-2016-28(1)-11

Список литературы

[1]. Kantrowitz, A. R. "A survey of physical phenomena occurring in flight at extreme speeds." Proceedings of the Conference on High-Speed Aeronautics. New York: Polytechnic Inst. of Brooklyn, 1955.
[2]. Ericson, William B., and A. Maciulaitis. "Investigation of magnetohydrodynamic flight control." Journal of Spacecraft and Rockets 1.3 (1964): 283-289.

[3]. Macheret, Sergey O., Mikhail N. Shneider, and Richard B. Miles. "Magnetohydrodynamic control of hypersonic flows and scramjet inlets using electron beam ionization." AIAA journal 40.1 (2002): 74-81.

[4]. Forget, Francois, et al. "Density and temperatures of the upper Martian atmosphere measured by stellar occultations with Mars Express SPICAM."Journal of Geophysical Research: Planets (1991–2012) 114.E1 (2009).

[5]. Xisto, Carlos M., et al. "Implementation of a 3D compressible MHD solver able to model transonic flows." Proc. ECCOMAS CFD 2010-V European Conference on Computational Fluid Dynamics. 2010.

[6]. Kurganov, Alexander, and Eitan Tadmor. "New high-resolution central schemes for nonlinear conservation laws and convection–diffusion equations." Journal of Computational Physics 160.1 (2000): 241-282.Bisek N. J., Poggie J., Boyd I. D. Numerical study of a MHD-heat shield

[7]. Balbás, Jorge, Eitan Tadmor, and Cheng-Chin Wu. "Non-oscillatory central schemes for one-and two-dimensional MHD equations: I." Journal of Computational Physics 201.1 (2004): 261-285.

Высокопроизводительное численное моделирование стратифицированных течений около клина в OpenFOAM

[1,2]*Н.Ф. Димитриева < dimitrieva@list.ru >*
[1]*Ю.Д Чашечкин < chakin@ipmnet.ru >*
[1]*ИПМех РАН, 119526, Россия, г. Москва, пр-т Вернадского, д. 101, корп. 1*
[2]*ИГМ НАНУ, 03680, Украина, г. Киев, ул. Желябова, д. 8/4*

Аннотация. Представлены результаты численного моделирования течений непрерывно стратифицированной жидкости, которые характеризуются широким диапазоном значений внутренних масштабов, отсутствующих в однородной жидкости. Поставленная задача решалась с использованием метода конечных объемов в открытом пакете OpenFOAM. Тестирование разработанной модели проводилось для течений непрерывно стратифицированных жидкостей около неподвижного и движущегося клиновидного тела с прямыми и искривленными гранями. Расчеты, проведенные с использованием вычислительных ресурсов web-лаборатории UniHUB, показали сложную структуру течений, включающую высокоградиентные прослойки около неподвижного препятствия и присоединенные внутренние волны вблизи острых кромок движущегося препятствия.

Ключевые слова: численное моделирование; открытые вычислительные пакеты; стратифицированные течения, тонкая структура.

DOI: 10.15514/ISPRAS-2016-28(1)-12

Для цитирования: Димитриева Н.Ф., Чашечкин Ю.Д. Высокопроизводительное численное моделирование стратифицированных течений около клина в OpenFOAM. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 207-220. DOI: 10.15514/ISPRAS-2016-28(1)-12

1. Введение

Одной из наиболее актуальных научных проблем является построение и численная реализация полных моделей механики неоднородных сред, описывающих природные процессы, протекающие в окружающей среде и промышленных аппаратах, которые допускают прямое сравнение с экспериментом. Расчет и измерения всех макро- и микрокомпонент течений представляют сложные задачи, все еще не решенные с практически необходимой степенью точности.

В настоящей работе представлены результаты расчета течений, индуцированных диффузией на неподвижном непроницаемом препятствии в непрерывно стратифицированной жидкости и возмущений около движущегося тела, которые активно изучаются последние семьдесят лет [1-3]. Они возникают в результате прерывания молекулярного потока стратифицирующей примеси на границах произвольной формы, увлечения жидкости и деформации равновесного поля плотности. В силу внутренней многомасштабности впервые полный расчет такого течения даже на простой топографии (пластина конечной длины) на основе фундаментальной системы уравнений [4].удалось выполнить только с применением суперкомпьютерных технологий [5].

Современные методы распараллеливания численных алгоритмов позволяют проводить вычисления на кластерных установках за вполне приемлемое время с использованием высокой пространственной дискретизации расчетной области. Применение высокопроизводительных вычислительных систем дает возможность более точно описать тонкоструктурные компоненты течений, проводить широкий параметрический анализ поставленных задач, а также допускает сравнение с данными лабораторных экспериментов и наблюдений в природных условиях [3, 6].

2. Постановка задачи

В данной работе решается нестационарная плоская задача формирования течений непрерывно стратифицированных жидкостей около неподвижного и движущегося клиновидного тела.

Математическое описание изучаемых физических процессов проводится на основе фундаментальной системы дифференциальных балансных уравнений механики неоднородных многокомпонентных жидкостей. В систему входят уравнения, выражающие в дифференциальной форме законы сохранения наблюдаемых физических величин – массы, импульса и вещества, а также замыкающее уравнение состояния [3, 4]. В традиционном приближении Буссинеска, когда малые изменения плотности на масштабах задачи учитывается только в членах с силой тяжести, система определяющих уравнений принимает вид

$$\rho = \rho_{00}\left(\exp\left(-y/\Lambda\right) + s\right); \tag{1}$$

$$\nabla \mathbf{v} = 0; \tag{2}$$

$$\frac{\partial s}{\partial t} + \mathbf{v}\cdot\nabla s = \kappa_s \Delta s + \frac{v_y}{\Lambda}; \tag{3}$$

$$\frac{\partial \mathbf{v}}{\partial t} + \left(\mathbf{v}\nabla\right)\mathbf{v} = -\frac{1}{\rho_{00}}\nabla P + \nu\Delta\mathbf{v} - s\mathbf{g}. \tag{4}$$

Здесь $S = S_0(y) + s$ – полная соленость, s – ее возмущенная составляющая, ρ_{00} – плотность на нулевом уровне (горизонте нейтральной плавучести), $\rho(y)$ – невозмущенное распределение плотности, которое задается профилем солености $S_0(y)$, где ось y направлена вертикально вверх, $\mathbf{v}$ – вектор скорости жидкости, P – давление за вычетом гидростатического, ν – коэффициент кинематической вязкости, κ_s – коэффициент диффузии соли, t – время, $\mathbf{g}$ – ускорение свободного падения, ∇ и Δ – операторы Гамильтона и Лапласа, $\Lambda = \left(d \ln \rho_0 / dy\right)^{-1}$ – длина плавучести, $N = \sqrt{g/\Lambda}$ – частота плавучести. Теоретико-групповой анализ системы (1-4) показал ее соответствие базовым принципам физики, положенным в основу вывода определяющих уравнений, в отличие от многих распространенных конститутивных или редуцированных систем [7].

В начальный момент времени $t = 0$ в покоящуюся непрерывно стратифицированную жидкость помещается непроницаемое препятствие, на поверхности которого задается условие прилипания для скорости и непротекания для вещества:

$$\mathbf{v}, s\big|_{t\le 0} = 0, \qquad v_{x,y}\big|_{\Sigma} = 0, \qquad \mathbf{v}, s\big|_{x,y\to\infty} = 0, \tag{5}$$

$$\left.\frac{\partial S}{\partial n}\right|_{\Sigma} = -\frac{1}{\Lambda}\frac{\partial y}{\partial n} + \left.\frac{\partial s}{\partial n}\right|_{\Sigma} = 0, \tag{6}$$

где $\mathbf{n}$ – внешняя нормаль к поверхности препятствия Σ. На большом удалении от препятствия задаются условия затухания всех возмущений.

Граничные условия (5) описывают течения, индуцированные диффузией на неподвижном препятствии [8]. Установившееся поля физических переменных такого течения служат начальными условиями для задачи обтекания препятствия потоком непрерывно стратифицированной жидкости, когда на удалении от препятствия задается невозмущенный поток [9]:

$$v_x\big|_{x,y\to\infty} = U, \quad v_y\big|_{x,y\to\infty} = 0. \tag{7}$$

Адекватность выбранной математической модели подтверждается соответствием основополагающим принципам механики и согласованностью независимых аналитических, численных и экспериментальных исследований стратифицированных течений около пластины и полуплоскости [3, 9, 10].

3. Масштабный анализ

Масштабный анализ задачи играет важную роль при разработке методики численного эксперимента: макромасштабы характеризуют размер области решения задачи, которая должна содержать все изучаемые компоненты течения, а микромасштабы – пространственное разрешение расчетной сетки.

Ранее выполненный анализ свойств линеаризированных фундаментальных уравнений и результаты лабораторного моделирования показывают, что все компоненты полного решения, как регулярно возмущенные, которые характеризуют волны и вихри, так и обширное семейство сингулярно возмущенных, описывающих сопутствующие тонкоструктурные элементы течений, проявляются в широком диапазоне параметров [3, 10].

Размерные параметры задачи формируют характерные масштабы: времени ($T_b = 2\pi/N$), скорости ($U_N^{\nu} = \sqrt{\nu N}$, $U_N^{\kappa_S} = \sqrt{\kappa_S N}$, U), а также длины. Большие линейные масштабы характеризуют исходную стратификацию (длину плавучести Λ) и геометрию течения (размер препятствия L). Скорость источника U задает длину гравитационных поверхностных $\lambda_s = 2\pi U^2/g$ и внутренних $\lambda_i = UT_b$ гравитационных волн. Микромасштабы диссипативной природы (вязкий $\delta_N^{\nu} = \sqrt{\nu/N}$ и диффузионный $\delta_N^{\kappa_S} = \sqrt{\kappa_S/N}$ микромасштабы) определяют поперечные размеры тонкоструктурных компонентов. Компоненты структур с масштабами Прандтля $\delta_U^{\nu} = \nu/U$ и $\delta_U^{\kappa_s} = \kappa_s/U$ выражены в струях и следах. Широкий диапазон значений масштабов длины (4-6 порядков) указывают на сложность внутренней структуры стратифицированного течения, которую необходимо учитывать при разработке программ.

Волны и вихри в неоднородных средах существуют одновременно и активно взаимодействуют между собой наряду с формирующейся тонкой структурой, которая влияет на перенос вещества, процессы разделения компонент течений, а также повышения локальной концентрации примеси. Основное достоинство рассмотренной постановки задачи в том, что она позволяет одновременно изучать все элементы течений в рамках единого описания в естественных физических переменных без привлечения дополнительных констант и связей.

4. Метод решения

Численные методы позволяют преодолеть трудности, возникающие при построении точных аналитических решений, учесть сложность геометрии задачи и подробно изучить структуру и динамику нестационарных течений в полной нелинейной постановке и естественных переменных без привлечения дополнительных ограничивающих приближений.

Численное решение поставленной задачи строится методом конечных объемов в открытом пакете OpenFOAM (Open Source Field Operation And Manipulation) с использованием объектно-ориентированного языка программирования C++. Выбор OpenFOAM обоснован открытостью его исходного кода, что существенно упрощает и ускоряет разработку собственных численных моделей. Анализ имеющегося инструментария пакета показал отсутствие готовых решений системы фундаментальных уравнений (1-4) и необходимость создания собственной численной модели. Для учета эффектов стратификации и диффузии стандартный решатель icoFoam, моделирующий нестационарные течения однородной жидкости, был дополнен новыми переменными (ρ и s) и соответствующими уравнениями (1, 3), а также новыми вспомогательными параметрами (N , Λ , κ_s , $\mathbf{g}$ и др.) [10].

Граничное условие возмущения солености (6) реализовано с помощью расширенной утилиты funkySetBoundaryField, которая позволяют задавать аналитические выражения для физических переменных. На передней и задней поверхностях задавалось граничное условие empty с целью исключения расчета в третьем измерении для плоской задачи.

Дискретизация расчетной области осуществлялась в открытой интегрируемой платформе SALOME. Простота геометрии позволяет построить блочно-структурированную гексаэдральную расчетную сетку с совмещением линий на границах блоков. Процедура построения была параметризирована с целью сокращения временных затрат на перестройку сетки при изменении геометрических параметров расчетной области. Проведенные тестовые расчеты с различным измельчением сетки подтвердили необходимость разрешения минимальных масштабов задачи [11].

Вычисления проводились в параллельном режиме с привлечением ресурсов web-лаборатории UniHUB (www.unihub.ru) [12]. Значения входных параметров приведено в таблице 1.

Таблица. 1. Значения входных параметров
Table. 1. The values of input parameters

№	Обозначение	Описание	Значение
1	ρ_{00}	плотность на нулевом уровне, кг/м^3	1020
2	ν	коэффициент кинематической вязкости, м^2/с	10^{-6}
3	κ_s	коэффициент диффузии соли, м^2/с	$1{,}41\cdot10^{-9}$
4	N	частота плавучести, с$^{-1}$	1
5	L	длина клина, м	0,1
6	h	высота основания клина, м	0,02

Использование суперкомпьютерных систем допускает высокую пространственную дискретизацию расчетной области и проведение более широкого параметрического анализа задач [3, 11]. Для вычисления дополнительных физических переменных, не входящих в решатель, использовались утилиты vorticity, stressComponents, funkySetFields и др. Визуализация результатов расчетов выполнялась с использованием графических пакетов ParaView и Origin. Для преобразования цифровых данных в кодах OpenFOAM к другим форматам использовались стандартные утилиты sample и topoSet.

5. Результаты и обсуждение

В данной работе приводятся результаты расчета течений непрерывно стратифицированных жидкостей около неподвижного и движущегося клиновидного тела с использованием предложенной методики численного моделирования.

5.1 Индуцированные диффузией течения в покоящейся стратифицированной жидкости

Устойчиво стратифицированная среда находится в состоянии покоя только когда градиенты плотности параллельны силе тяжести. Если в покоящуюся стратифицированную жидкость поместить непроницаемое тело, оно прерывает естественный молекулярный поток вещества. Это приводит к переизбытку примеси в тонком слое под препятствием (красный цвет) и его дефициту над препятствием (синий), как показано на рис. 1. Таким образом формируются восходящие и нисходящие струйные течения вдоль сторон препятствия, компенсирующие возникающую неравновесность. Их называют течениями, индуцированными диффузией [1, 2]. К пристенным зонам примыкают области с обратным знаком возмущения солености, иллюстрирующие области дефицита и избытка стратифицирующей компоненты. При этом величина возмущения уменьшается в направлении от препятствия, а толщина слоя, наоборот, увеличивается.

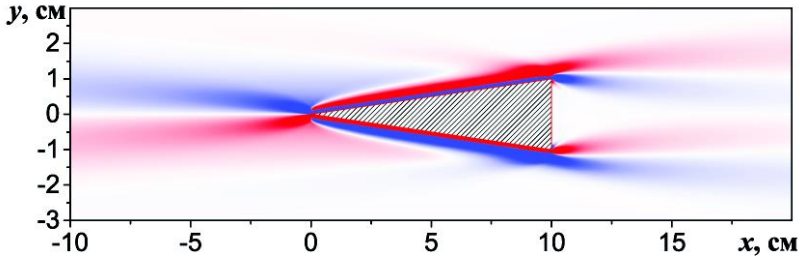
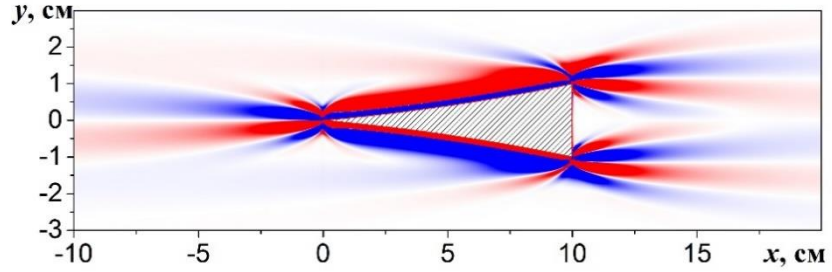
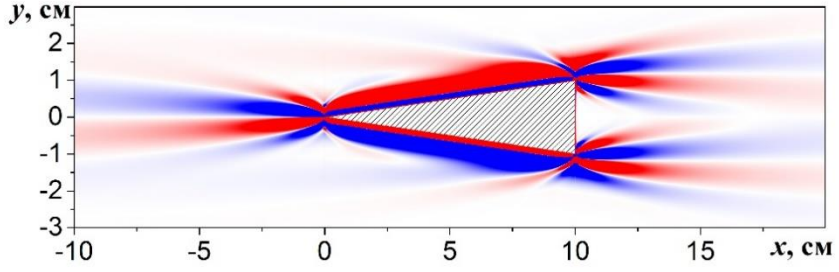


Рис. 1. Возмущение солености, индуцированной диффузией на клине
Fig. 1. Perturbation of salinity induced by diffusion on a wedge

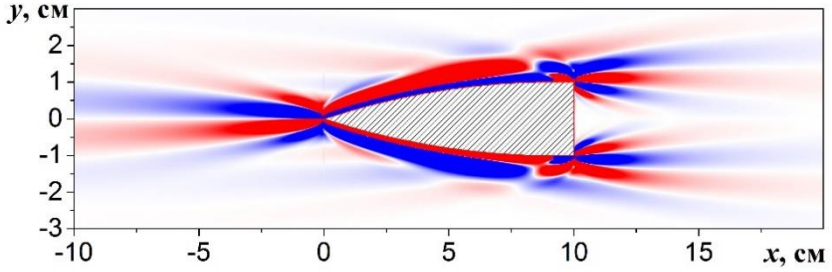
Важную роль играют краевые эффекты, где схождение с острых кромок клиновидного препятствия тонких струйных течений жидкости, формирующихся вдоль каждой из его сторон, порождает внутренние волны. Обычно тонкоструктурные эффекты вносят небольшие поправки в значения характеристик течений, но их действие усиливается большой величиной градиентов солености, поля которых отражает сложную периодическую структуру течений, индуцированных диффузией (рис. 2). Горизонтальные полосчатые структуры согласуются с экспериментальным картинам визуализации (“цветной теневой метод” с горизонтальной щелью и решеткой) распределения градиента коэффициента преломления в лабораторном бассейне для тел с другими геометрическими формами [3, 5].



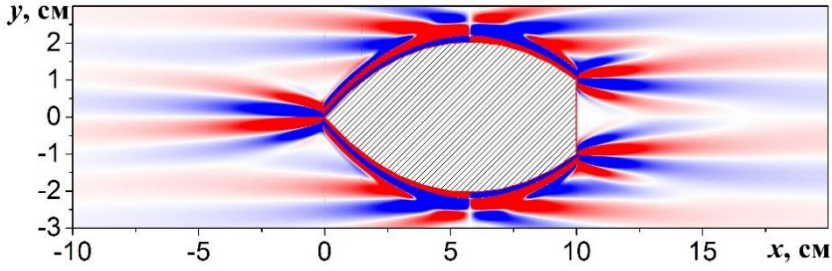
а)



б)



в)



г)

Рис. 2. Градиент возмущения солености для неподвижного клиновидного тела: y_0 = 4 мм (а), y_0 = 5 мм (б), y_0 = 8 мм (в), y_0 =20 мм (г)
Fig. 2. Gradient of salinity perturbation for fixed wedge-shaped body: y0 = 4 mm (a), y0 = 5 mm (б), y0 = 8 mm (в), y0 = 20 mm (г).

Общая структура изображения типична для стратифицированных течений, в которых силы плавучести подавляют вертикальное движение. Неоднородности вертикального молекулярного потока вещества, вызванные непроницаемыми препятствиями в толще жидкости или наклоном ее границ, создают горизонтальные градиенты плотности, которые образуют течения даже при отсутствии дополнительных силовых факторов и вызывают самодвижение тел свободных тел нейтральной плавучести.

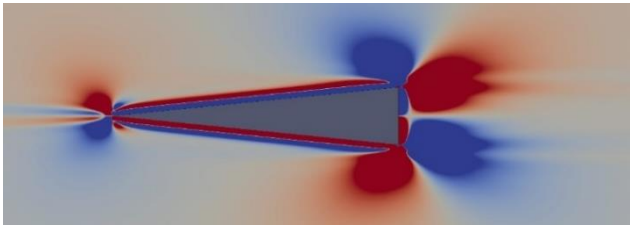
С целью изучения влияния формы препятствия на структуру течений, индуцированных диффузией, рассматривались клинья с прямыми и искривленными гранями симметрично относительно продольной оси *x*. Искривление боковой грани получено путем проведения дуги окружности через три точки. В центральной точке при x = 50 мм см для клина с прямой гранью y_0 = 5 мм (рис. 2, *б*), для вогнутого клина y_0 < 5 мм (рис. 2, *а*), для выпуклого – y_0 > 5 мм (рис. 2, *в, г*).

Около угловых точек клина формируются дополнительные тонкоструктурные компоненты. Чем острее экстремальная вершина, тем ярче выражены визуализируемые пучки знакопеременных полос (рис. 2, *а*). Для выпуклого клина (рис. 2, *в*), у которого угол между основанием и боковой гранью приближается к 90°, пучок тонкоструктурных элементов расплывается от вершины вдоль грани. Принципиально меняется картина течения, если сторону клина изогнуть таким образом, что экстремальная точка смещается от вершины к точке на грани (рис. 2, *г*). В этом случае визуализируемые структуры подобны теневым картинам течений, индуцированных диффузией на цилиндре [6].

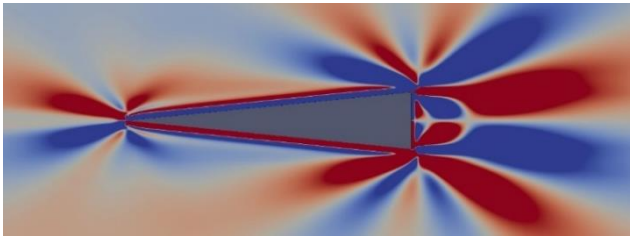
Течения, индуцированные диффузией на клиновидном препятствии, представляют научный и практический интерес в связи с возникновением ненулевой интегральной силы, приводящий к самодвижению. При экспериментальных и теоретических исследованиях погруженных в

стратифицированную жидкость тел нейтральной плавучести, симметричных относительно линии действия силы тяжести (пластина, цилиндр и т.п.), эта отличительная особенность не была обнаружена [3, 5].

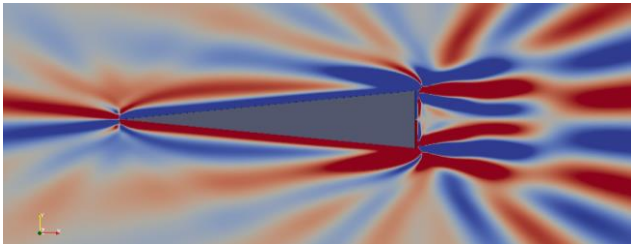
Анализ структуры поля давления выявил протяженную область отрицательного давления у острой вершины клиновидного препятствия [8]. Разность давлений – подпор у основания и дефицит перед клином объясняет возникновение интегральной силы, толкающей горизонтальный клин в направлении его вершины. Экспериментальные исследования [13] подтвердили возможность самодвижения клиновидного тела в устойчиво стратифицированной жидкости.



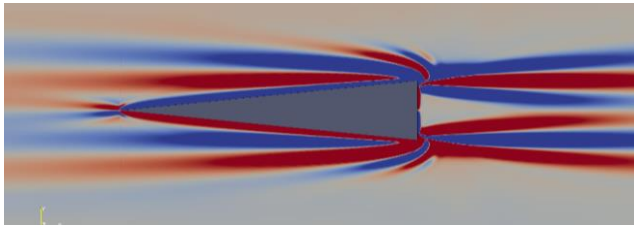
а)



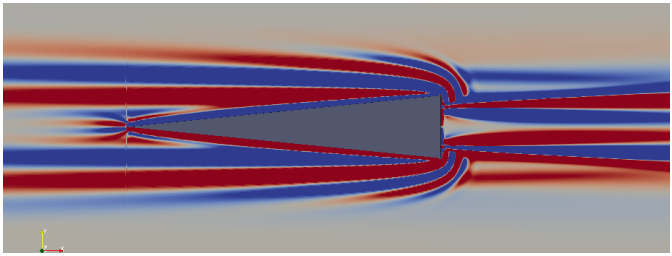
б)



в)



г)



д)

Рис. 3. Эволюция поля градиента возмущения солености при движении клина со скоростью U = 10^{-4} м/с: (а - д) $\tau = t / T_b =$ *0.3, 1.1, 3.2, 7.9, 15.9*

Fig. 3. Evolution of the field of the salinity perturbation gradient when the wedge moves with velocity U = 10-4 m / s (a - d), $\tau = t / T_b =$ *0.3, 1.1, 3.2, 7.9, 15.9*

5.2 Возмущения покоящейся стратифицированной жидкости движущимся клином

Полученные результаты расчетов индуцированного диффузией течения на неподвижном клине служат начальными условиями задачи обтекания тел внешним потоком. На рис. 3 продемонстрирована эволюция картины обтекания клина стратифицированным потоком, который начал равномерно двигаться из состояния покоя со скоростью $U = 10^{-4}$ м/с в жидкости с периодом плавучести $T_b = 6.28$ с. Начальная структура течения, индуцированного диффузией на непроницаемом клине, кардинально меняется с началом движения препятствия. В толще непрерывно стратифицированной жидкости начинают формироваться опережающие возмущения, розетки нестационарных и поля присоединенных внутренних волн, а также протяжённый след за экстремальными точками. Необходимо отметить, что заданные ненулевые начальные условия задачи обтекания клина сохраняют свое влияние на структуру течения только на начальном этапе (около $1,5T_b$). Число наблюдаемых присоединенных волн, не проникающих в спутный след позади тела растет со временем.

Сформированная картина обтекания клина (рис. 3, *д*) по своей структуре согласуется с результатами экспериментальных и численных исследований обтекания тел с другими геометрическими формами потоком стратифицированной жидкости [3, 6, 9, 14]. Источником внутренних волн служат краевые сингулярности, генерирующие интенсивное вертикальное вытеснение жидкости, что приводит к отклонению от изначального положения нейтральной плавучести и, как следствие, формированию периодических затухающих колебаний жидкости.

6. Заключение

Вычисления, проведенные в открытом пакете OpenFOAM с использованием суперкомпьютерных систем, показали возможность расчета многомасштабных структурированных течений на основе фундаментальной системы уравнений в широком диапазоне параметров задачи. Расчеты течений непрерывно стратифицированной жидкости около неподвижного и движущегося клиновидного тела показали высокую работоспособность предложенной численной модели**.**

Работа выполнена при финансовой поддержке РФФИ (проект 15-37-50382). Расчеты проводились с использованием вычислительных мощностей web-лаборатории UniHUB ИСП РАН (www.unihub.ru).

Список литературы

[1]. Л. Прандтль. Гидроаэромеханика. М.: ИИЛ, 1949 г. 488 с.

[2]. O. M. Phillips. On flows induced by diffusion in a stably stratified fluid. Deep-Sea Res., volume 17, 1970. P. 435–443.

[3]. Ю.Д. Чашечкин Дифференциальная механика жидкостей: согласованные аналитические, численные и лабораторные модели стратифицированных течений // Вестник МГТУ им. Н.Э. Баумана, сер. «Естественные науки», № 6, 2014 г. стр. 67-95. http://vestniken.bmstu.ru/articles/547/547.pdf

[4]. Л.Д. Ландау, Е.М. Лифшиц. Теоретическая физика, том VI, Гидродинамика. М.: Наука, 1986. 736 с.

[5]. Ю.Д. Чашечкин, Я.В. Загуменный Структура течения, индуцированного диффузией на наклонной пластине // Доклады Академии Наук, том 444, № 2, 2012 г. стр. 165-171. doi: 10.1134/S1028335812050047

[6]. Yu.D. Chashechkin, V.V. Mitkin Soaring interfaces, vortices and vortex systems inside the internal waves wake past the horizontally moving cylinder in a continuously stratified fluid. J. Visualiz, volume 9, issue 3, 2006. P. 301-308. doi: 10.1007/BF03181677

[7]. В.Г. Байдулов, Ю.Д. Чашечкин Сравнительный анализ симметрий моделей механики неоднородных жидкостей // Доклады Академии Наук,.том 444, № 1, 2012 г. стр. 38–41. doi: 10.1134/S1028335812050011

[8]. Н.Ф. Димитриева, Ю.Д. Чашечкин Численное моделирование динамики и структуры индуцированного диффузией течения на клине // Вычислительная механика сплошных сред, том 8, № 1, 2015 г. стр. 102– 110. doi: 10.7242/1999-6691/2015.8.1.9

[9]. Ю.Д. Чашечкин, Р.Н. Бардаков, Я.В. Загуменный. Расчет и визуализация тонкой структуры полей двумерных присоединенных внутренних волн, Морской гидрофизический журнал, № 6. 2010 г. стр. 3-15.

[10]. .Н.Ф. Димитриева, Я.В. Загуменный Численное моделирование стратифицированных течений с использованием OpenFOAM // Труды Института системного программирования РАН, том 26, № 5, 2014 г. стр. 187– 200. doi: 10.15514/ISPRAS-2014-26(5)-10

[11]. Н.Ф. Димитриева Расчет стратифицированных течений около клина с использованием открытых вычислительных пакетов // Прикладная гидромеханика, том 17, № 2, 2015 г. стр. 26-35.

[12]. О. Самоваров, С. Гайсарян. Архитектура и особенности реализации платформы UniHUB в модели облачных вычислений на базе открытого пакета OpenStack. Труды Института системного программирования РАН, том 26, вып. 1, 2014 г. стр. 403-420 doi: 10.15514/ISPRAS-2014-26(1)-17.

[13]. M. J. Mercier, F. M. Ardekani, M. R. Allshouse, B. Doyle, T. Peacock. Self-propulsion of immersed object via natural convection. Physical review letters, volume 112, 2014. P. 204501(5). doi: 10.1103/PhysRevLett.112.204501

[14]. Yu.D. Chashechkin, V.V. Mitkin. A visual study on flow pattern around the strip moving uniformly in a continuously stratified fluid, J. Visualiz, volume 7, Issue 2, 2004. P. 127-134. doi: 10.1007/BF03181585

High-performance numerical simulation of stratified flows around a wedge in OpenFOAM

[1,2]*N.F. Dimitrieva < dimitrieva@list.ru >*
[1]*Yu.D. Chashechkin < chakin@ipmnet.ru >*
[1]*IPMech RAS, 119526, Russia, Moscow, 101/1 Vernadskogo Avenue*
[2]*IHM NASU, 03680, Ukraine, Kiev, 8/4 Zheliabova Street*

Abstract. Results of numerical simulation of a continuously stratified fluid are presented. They are characterized by a wide range of values of internal scales that are not in a homogeneous liquid. Mathematical model is based on the fundamental set of differential equations of inhomogeneous multicomponent fluid mechanics. The problem is solved using the finite volume method in an open source package OpenFOAM. To take into account the stratification and diffusion effects a new own solver was developed and tested using the standard and extended libraries of the package. A particular attention is focused at construction of a high quality computational grid which satisfies basic requirements for resolution of all the microscales of the problem in high-gradient regions of the flow. Testing of the proposed numerical model Testing was conducted for continuously stratified fluid flows around a motionless and a moving wedge-shaped body with straight and curved edges. The calculations performed in parallel regime on computational facilities of the web-laboratory UniHUB (www.unihub.ru) demonstrated complex structure of flows. High-gradient layers near the sharp edges of the obstacles have been identified. Formation of an intensive zone of pressure depression in front of the leading vertex of the wedge is

responsible for generation of propulsive mechanism that results in a self-motion of the obstacle along its neutral buoyancy horizon in a stably stratified environment.

Keywords: numerical simulation; open source computational packages; stratified flows.

DOI: 10.15514/ISPRAS-2016-28(1)-12

For citation: Dimitrieva N.F., Chashechkin Yu.D. High-performance numerical simulation of stratified flows around a wedge in OpenFOAM. Trudy ISP RAN /Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 207-220 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-12

References

[1]. Prandtl L. Essentials of fluid dynamics. *London: Blackie and Son Publ.*, 1952. 452 p.

[2]. Phillips O. M. On flows induced by diffusion in a stably stratified fluid. *Deep-Sea Res.*, 1970, vol. 17, pp. 435–443.

[3]. Chashechkin Yu. D. Differentsial'naya mekhanika zhidkostej: soglasovannye analiticheskie, chislennye i laboratornye modeli stratifitsirovannyh techenij [Fluid mechanics: consistent analytical, numerical and laboratory models of stratified flows] *Herald of the Bauman MSTU, series "Natural Science"* [*Vestnik MGTU im. N.E. Baumana, series "Estestvennye nauki"*], 2014, no 6, pp. 67-95. (in Russian) http://vestniken.bmstu.ru/articles/547/547.pdf

[4]. L. D. Landau, E. M. Lifshits. Teoreticheskaya fizika, tom VI Gidrodinamika [Theoretical physics, volume VI, Hydrodynamics]. *Moscow, Nauka Publ.* [*Moscow: Science*], 1986. 736 p. (in Russian)

[5]. Chashechkin Yu.D, Zagumennyi Ya.V. Structure of Diffusion-Induced Flow on an Inclined Plate. *Doklady Physics*, 2012, vol. 57, no. 5, pp. 210–216. doi: 10.1134/S1028335812050047

[6]. Chashechkin Yu.D., Mitkin V.V. Soaring interfaces, vortices and vortex systems inside the internal waves wake past the horizontally moving cylinder in a continuously stratified fluid. *J. Visualiz*., 2006, vol. 9, no. 3, pp. 301-308. doi: 10.1007/BF03181677

[7]. Baydulov V.G., Chashechkin Yu.D. Comparative analysis of symmetries for the models of mechanics of nonuniform fluids. *Doklady Physics*, 2012, vol. 57, no. 5, pp. 192–196. doi: 10.1134/S1028335812050011

[8]. Dimitrieva N.F., Chashechkin Yu.D. Chislennoe modelirovanie dinamiki i struktury indutsirovannogo diffuziej techeniya na kline [Numerical simulation of the dynamics and structure of a diffusion-driven flow on a wedge], *Vychislitel'naya mekhanika sploshnykh sred* [*Computation continuum mechanics*], 2015, vol. 8, no. 1, pp. 102– 110. (in Russian) doi: 10.7242/1999-6691/2015.8.1.9

[9]. Chashechkin Yu. D., Bardakov R. N., Zagumennyi Ia. V. Raschet i vizualizatsiya tonkoj struktury polej dvumernykh prisoedinennykh vnutrennikh voln [Calculation and visualization of the fine structure of fields of two-dimensional attached internal waves], *Morskoj gidrofizicheskij zhurnal* [*Marine Hydrophysical Journal*], 2010, no. 6, pp. 3-15. (in Russian)

[10]. Dimitrieva N.F., Zagumennyi Ia. V. Chislennoe modelirovanie stratifitsirovannyih techeniy s ispolzovaniem OpenFOAM [Numerical simulation of stratified flows using OpenFOAM] *Trudy ISP RAN* [*The Proceedings of ISP RAS*], 2014, vol. 26, no. 1, pp. 187-200. (in Russian) doi: 10.15514/ISPRAS-2014-26(5)-10

[11]. Dimitrieva N.F. Raschet stratifitsirovannykh techenij okolo klina s ispol'zovaniem otkrytykh vychislitel'nykh paketov [Calculation of stratified flows around a wedge using open software packages] *Prikladnaya gidromekhanika* [*Applied Hydromechanics*], 2015, vol. 17, no. 2, pp. 26-35. (in Russian)

[12]. O. Samovarov, S. Gaysaryan. Arkhitektura i osobennosti realizatsii platformy UniHUB v modeli oblachnykh vychislenij na baze otkrytogo paketa OpenStack [The web-laboratory architecture based on the cloud and the UniHUB implementation as an extension of the OpenStack platform]. *Trudy ISP RAN* [*The Proceedings of ISP RAS*], 2014, vol. 26, no. 1, pp. 403-420. (in Russian). doi: 10.15514/ISPRAS-2014-26(1)-17

[13]. Mercier M. J., Ardekani F. M., Allshouse M. R., Doyle B., Peacock T. Self-propulsion of immersed object via natural convection. *Physical review letters*, 2014, vol. 112, pp. 204501(5). doi: 10.1103/PhysRevLett.112.204501

[14]. Chashechkin Yu.D., Mitkin V.V. A visual study on flow pattern around the strip moving uniformly in a continuously stratified fluid. *J. Visualiz*, 2004, vol. 7, no. 2, pp. 127-134. doi: 10.1007/BF03181585

Реализация параллельных вычислений в программном комплексе «LS-STAG_turb» для моделирования течений вязкой несжимаемой среды на системах с общей памятью

В.В. Пузикова <valeria.puzikova@gmail.com>
МГТУ им. Н.Э. Баумана,
105005, Россия, г. Москва, ул. 2-я Бауманская, дом 5.

Аннотация. Метод погруженных границ LS-STAG и его модификации для решения сопряженных задач гидроупругости с использованием моделей турбулентности Смагоринского, Спаларта – Аллмараса, $k-\varepsilon$, $k-\omega$ и $k-\omega$ SST в рамках RANS, LES и DES подходов к моделированию турбулентности реализованы в программном комплексе «LS-STAG_turb» для моделирования движения профилей в потоке вязкой несжимаемой среды. Комплекс позволяет моделировать обтекание движущихся профилей произвольной формы и систем из любого числа профилей, имеющих одну или две степени свободы, например, авторотацию роторов ветроэнергетических установок, ветровой резонанс систем профилей. Для сокращения затрат машинного времени на проведение расчетов разработана параллельная версия алгоритмов и проведена оптимизация участков последовательного кода. Использованы такие технологии параллельного программирования, как Intel® Cilk™ Plus, Intel® Threading Building Blocks и OpenMP.

Ключевые слова: технология OpenMP; технология Intel® Cilk™ Plus; технология Intel® Threading Building Blocks; вязкая несжимаемая среда; метод погруженных границ**.**

DOI: 10.15514/ISPRAS-2016-28(1)-13

Для цитирования: Пузикова В.В. Реализация параллельных вычислений в программном комплексе «LS-STAG_turb» для моделирования течений вязкой несжимаемой среды на системах с общей памятью. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 221-242. DOI: 10.15514/ISPRAS-2016-28(1)-13

1. Введение

Во многих инженерных приложениях возникает необходимость решения сопряженных задач гидроупругости. В качестве примеров можно привести

расчет обтекания роторов ветроэнергетических установок, труб теплообменников энергетических установок, удлиненных элементов конструкций зданий и сооружений, проводов линий электропередачи и т.п. Такие задачи из-за необходимости моделирования взаимного влияния течения жидкости и движения погруженного в нее тела являются достаточно сложными для численного решения и требуют применения высокоточных численных методов. Кроме того, поскольку из-за движения тела форма расчетной области изменяется в процессе расчета, при использовании сеточных методов с сеткой, связанной с телом, достаточно существенными становятся вычислительные затраты на перестроение сетки. Однако, существуют сеточные методы, в которых сетка не связана с границей тела и не изменяется на протяжении всего расчета, несмотря на движение обтекаемого тела. Такие методы относятся к классу методов погруженных границ [1]. Данные методы предполагают использование прямоугольных сеток. При пересечении прямоугольных ячеек сетки с границей области течения образуются ячейки неправильной формы, называемые усеченными. При использовании методов погруженных границ важно обеспечить высокую точность решения задачи именно в усеченных ячейках, поскольку на них задаются граничные условия, и, кроме того, решение вблизи границы обтекаемого тела (погруженной границы) может иметь большие градиенты.

К наиболее эффективным методам этого класса относят метод LS-STAG [2]. LS-STAG-сетка состоит из трех разнесенных сеток, ячейки которых представляют собой контрольные объемы для скоростей и давления. Для представления погруженной границы используется аппарат функций уровня [3], а LS-STAG-дискретизация производится по одним и тем же формулам, как в прямоугольных ячейках, так и в усеченных, причем шаблон дискретизации имеет в двумерном случае пятиточечную структуру. Все это позволяет значительно снизить затраты машинного времени на обработку усеченных ячеек. В работе [2] построена LS-STAG-дискретизация двумерных уравнений Навье – Стокса для вязкой несжимаемой среды. Автором настоящей статьи разработаны модификации метода LS-STAG, позволяющие использовать модели турбулентности Смагоринского, Спаларта – Аллмараса, $k-\varepsilon$, $k-\omega$ и $k-\omega$ SST в рамках RANS, LES и DES подходов [4, 5], а также модификации метода LS-STAG для решения сопряженных задач гидроупругости [6]. Как и многие «нестандартные» высокоточные методы, метод погруженных границ LS-STAG не реализован в широко распространенных пакетах вычислительной гидродинамики, поэтому весьма актуальной задачей является разработка эффективной программной реализации метода LS-STAG и его модификаций.

Автором разработан программный комплекс «LS-STAG_turb» для моделирования движения профилей в потоке вязкой несжимаемой среды [7]. Программа написана на языке C++ и имеет объектно-ориентированную легко расширяемую структуру. Комплекс позволяет моделировать обтекание

движущихся профилей произвольной формы, например вращение роторов ветроэнергетических установок. Кроме того, возможно моделирование обтекания систем, состоящих из любого числа подвижных профилей, имеющих одну или две степени свободы. Результаты верификации комплекса демонстрируют высокую точность метода LS-STAG и разработанных модификаций: удается качественно и количественно верно моделировать достаточно сложные и «тонкие» гидродинамические эффекты, например, эффект стабилизации следа за круговым профилем, совершающим высокочастотные вращательные колебания, известный как эффект Танеды [8, 9]. При этом для получения точных количественных результатов в случае моделирования явлений, отличающихся высокими скоростями движения профилей, и, соответственно, высокими значениями местного числа Рейнольдса, которые характерны, например, для ветрового резонанса профиля или системы профилей, необходимо сильное измельчение сетки, что приводит к резкому росту затрат вычислительных ресурсов. Для решения этой проблемы необходимо разработать параллельную версию программного комплекса «LS-STAG_turb».

Для распараллеливания вычислений при решении задач вычислительной механики часто используют методы декомпозиции области [10]. Идея методов декомпозиции заключается в том, что расчетная область разбивается на пересекающиеся или непересекающие подобласти и исходная задача представляется в виде совокупности вспомогательных краевых задач в этих подобластях. При этом на границах подобластей, совпадающих с границами исходной расчетной области, ставятся граничные условия из исходной задачи, а на остальных границах подобластей, называемых внутренними, ставятся условия сопряжения, которые записываются в виде граничных условий третьего рода. Решение вспомогательных задач может осуществляться параллельно, при этом эффективность полученного алгоритма зависит от многих факторов [11]: наличия и величины пересечения смежных подобластей, топологии декомпозиции области, типов граничных условий на внутренних границах подобластей, организации итерационных процессов, эффективности распараллеливания вычислений при решении вспомогательной задачи в подобласти. На LS-STAG-сетке дополнительные сложности также возникают из-за наличия твердых и усеченных ячеек и изменения их местоположения при перемещении подвижной погруженной границы. Таким образом, разработка параллельного алгоритма решения задачи методом LS-STAG, основанного на декомпозиции расчетной области, представляет собой предмет отдельного исследования. В рамках данной работы рассматриваются только вопросы оптимизации и распараллеливания вычислений при решении одной задачи в расчетной области без подобластей. Разработанные алгоритмы впоследствии также можно будет применять для эффективного решения вспомогательнх задач в подобластях.

2. Программный комплекс «LS-STAG_turb»

Общая схема работы программного комплекса «LS-STAG_turb» представлена на рис. 1. В блоке Б1 происходит заполнение полей структуры описания задачи из файла с постановкой, инициализация сервисной информации (текущее время, номер итерации, пути к папкам с результатами), открытие лог-файла, инициализация экземпляров структур, реализующих работу с решениями, LS-STAG-сеткой и разностыми аналогами решаемых уравнений. Затем запускается процесс моделирования: до выполнения заданного числа шагов по времени выполняются блоки Б2–Б7.

Рис. 1. Блок-схема работы программного комплекса «LS-STAG_turb»
Fig. 1. Block diagram of the «LS-STAG_turb» software system

При моделировании движения погруженных границ в блоке Б2 происходит пересчет функции уровня и зависящих от нее характеристик сетки. В комплексе «LS-STAG_turb» реализован алгоритм построения функции уровня для профиля произвольной формы при помощи аппроксимации границы кривой Безье [12], что позволяет моделировать обтекание профилей сложной формы и их систем (рис. 2).

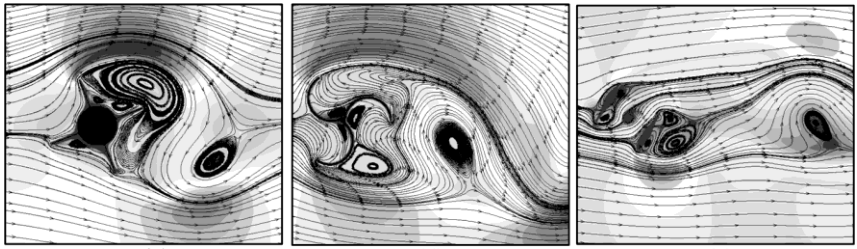


Рис. 2. Авторотация профилей различных форм, смоделированная в программном комплексе «LS-STAG_turb»: (а) пропеллер с четырьмя лопастями; (б) ротор Савониуса с тремя лопастями; (в) ротор Дарье с двумя лопастями в форме симметричного крылового профиля ЦАГИ серии В с относительной толщиной 20 %
Fig. 2. Autorotation of profiles of differbent shapes, modeled in the «LS-STAG_turb» software package: (a) propeller with four blades; (б) Savonius rotor with three lobes; (в) Darrieus rotor with two blades in the form of a symmetrical airfoil TsAGI of B Series with a relative thickness of 20%

При моделировании движения погруженных границ перед пересчетом правых частей решаемых систем линейных алгебраических уравнений в блоке Б3 также происходит решение разностных аналогов уравнений движения обтекаемого профиля или системы профилей и пересчет матриц разностных аналогов операторов и предобуславливателей. Все эти операции выполняются в методе `solve()` базовой структуры `DiscreteEquationInterface`, реализующей работу с разностными аналогами уравнений, перед решением систем линейных алгебраических уравнений (блок Б4). На каждом шаге по времени происходит решение двух разностных аналогов уравнения Гельмгольца для прогнозов скоростей и одного разностного аналога уравнения Пуассона для поправки давления. Значения скоростей и давления на текущем шаге по времени получаются после коррекции полученных прогнозов и поправок в блоке Б5. Помимо этого при использовании моделей турбулентности в блоке Б5 происходит решение разностных аналогов уравнений из используемой модели турбулентности и расчет рейнольдсовых или подсеточных напряжений, которые учитываются при пересчете правых частей систем для прогнозов скоростей на следующем шаге по времени. В зависимости от информации из структуры с описанием постановки задачи после этого может быть произведен расчет действующих на погруженные границы со стороны потока сил (блок Б6) или осуществлено сохранение в файл результатов моделирования и текущего состояния расчета (блок Б7). По окончании процесса моделирования происходит запись в лог-файл данных о времени завершения расчета и его продолжительности.

Для верификации разработанного программного комплекса «LS-STAG_turb» использовались тестовые задачи о моделировании обтекания неподвижных и движущихся профилей различных форм. В частности, смоделирован

наблюдавшийся в эксперименте [8] эффект стабилизации следа за профилем, совершающим высокочастотные вращательные колебания, известный как эффект Танеды и редко воспроизводимый численно (рис. 3). Все результаты верификационных расчетов хорошо согласуются с известными в литературе экспериментальными и расчетными данными.

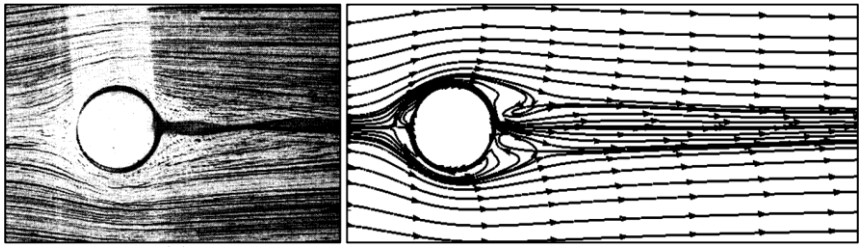


Рис. 3. Стабилизация следа за круговым профилем, совершающим высокочастотные вращательные колебания при $\mathrm{Re} = 111$*: (а) эксперимент [8]; (б) расчет методом LS-STAG*

Fig. 3. Track stabilization of circular profile committing high-frequency rotational oscillations at: (A) experiment [8]; (B) calculation with method LS-STAG

Помимо моделирования течений вязкой несжимаемой среды, описываемых уравнениями Навье – Стокса, программный комплекс «LS-STAG_turb» позволяет проводить расчеты с использованием моделей турбулентности Смагоринского, Спаларта – Аллмараса, $k-\varepsilon$, $k-\omega$ и $k-\omega$ SST в рамках RANS, LES и DES подходов к моделированию турбулентности. Пример такого расчета представлен на рис. 4. Также имеется возможность моделирования обтекания профилей и их систем, имеющих 1 или 2 степени свободы (рис. 5). Как было отмечено выше, для получения точных количественных результатов в задачах такого плана необходимо сильное измельчение сетки, приводящее к резкому росту вычислительных затрат. Решением данной проблемы может служить распараллеливание вычислений.

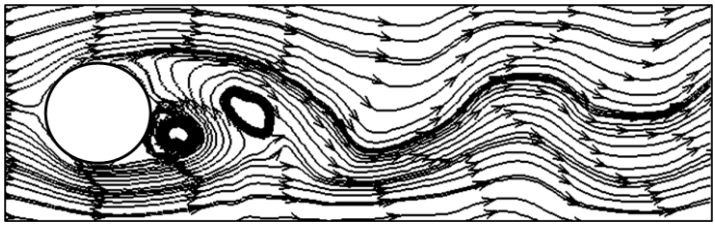


Рис. 4. Обтекание кругового профиля при $\mathrm{Re} = 3900$*, смоделированное в программном комплексе «LS-STAG_turb» с использованием модели турбулентности Спаларта – Аллмараса в рамках подхода RANS*

Fig. 4. Flow around a circular profile with Re = 3900 modeled in the software package «LS-STAG_turb» using Spalart--Allmaras turbulence model with approach RANS

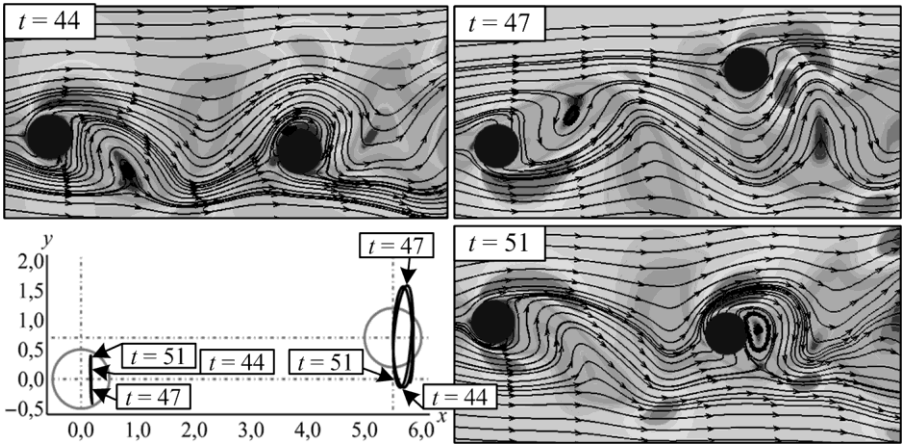


Рис. 5. Ветровой резонанс системы из двух круговых профилей, расположенных под углом выноса и имеющих 2 степени свободы, при $\mathrm{Re} = 100$*, смоделированный в программном комплексе «LS-STAG_turb»: траектории движения центров профилей и линии тока в моменты времени* $t = 44$, $t = 47$ *и* $t = 51$

Fig. 5. Wind resonance of a system of two circular profiles placed at an angle of carrying out and having two degrees of freedom, when modeled in the software package «LS-STAG_turb»: trajectory profiles centers and line of flow at times t = 44, t = 47, and t = 51

3. Постановка тестовой задачи и используемые аппаратно-программные средства

В качестве задачи для тестирования эффективности разрабатываемых параллельных алгоритмов рассмотрим задачу о моделировании обтекания кругового профиля диаметра D, совершающего в невозмущенном потоке вынужденные поперечные колебания по закону

$$\begin{cases} X_{\mathrm{C}} = X_C^0, \\ Y_{\mathrm{C}} = Y_C^0 + \begin{cases} A, & t < 10D/V_\infty, \\ A\cos(2\pi S_e[tV_\infty - 10D]/D), & t \geq 10D/V_\infty, \end{cases} \end{cases}$$

где A – амплитуда колебаний кругового профиля, S_e – кинематическое число Струхаля, (X_C^0, Y_C^0) – координаты центра профиля в среднем положении, (X_C, Y_C) – координаты центра профиля в текущий момент времени, V_∞ – скорость набегающего горизонтального потока, t – безразмерное время. При тестировании модификаций программного комплекса и параллельных алгоритмов будем моделировать 30 единиц безразмерного времени на

неравномерной сетке 240×296 (линейный размер ячейки сетки вблизи границы профиля $h = 0{,}03125$) с шагом по времени $\Delta t = 0{,}005$ при $A = 0{,}2D$, $D = 1{,}0$, $V_\infty = 1{,}0$, $\mathrm{Re} = 185$, $S_e/\mathrm{Sh} = 1{,}2$, где Re – число Рейнольдса, $\mathrm{Sh} \approx 0{,}201$ – число Струхаля (безразмерная частота схода вихрей), вычисленное при данном значении числа Рейнольдса для неподвижного профиля. Далее эту тестовую задачу будем обозначать VerOscTest.

Тестирование проводилось на рабочей станции, построенной на платформе Intel H81 с использованием двухъядерного процессора Intel Core i3-4350T (Haswell) с поддержкой HyperThreading (4 логических ядра), работающего на частоте 3100 МГц. Рабочая станция оснащена 8 ГБайт оперативной памяти DDR3-1333, SSD-накопителем Crucial объемом 128 ГБайт и жестким диском Seagate объемом 1 ТБайт. Внешние графические карты в рабочей станции не использовались. Далее эту рабочую станцию будем обозначать PC1.

Для исследования масштабируемости алгоритмов помимо PC1 также использовалась рабочая станция, построенная на платформе Intel Z97 с использованием 4-ядерного процессора Intel Core i7-4790K (Devil's Canyon) с поддержкой технологии HyperThreading (8 логических ядер), работающего на частоте 4400 МГц. Рабочая станция оснащена 16 ГБайт оперативной памяти DDR3-1600 и двумя SSD-накопителями Crucial объемом 256 ГБайт и 1 ТБайт. Внешние графические карты в рабочей станции не использовались. Данную рабочую станцию будем обозначать PC2.

Время решения тестовой задачи последовательным программным комплексом «LS-STAG_turb», при разработке которого использовался компилятор Microsoft Visual Studio 2010, на PC1 составляет 4666 с. Использование вместо этого компилятора оптимизирующего компилятора Intel C++ 15.0 позволяет без модификаций исходного кода уменьшить время счета на 0,5 %. Включение опции компилятора /MT предписывает приложению использовать многопоточную статическую версию библиотеки времени выполнения и разрешает дополнительные оптимизации компилятора, использующие многопоточность. После включения этой опции время решения тестовой задачи VerOscTest на PC1 составило 3968 с. Таким образом, время проведения расчета удалось сократить на 15 %. При проведении дальнейших экспериментов по оптимизации программы и распараллеливанию вычислений будем использовать компилятор Intel C++ 15.0 с опцией /MT.

4. Оценка эффективности распараллеливания вычислений

Определим, какие вычисления имеет смысл распараллелить в первую очередь. Для этого необходимо выделить участки программы, на выполнение которых расходуется наибольшее количество времени. Полная структура временны́х затрат при решении тестовой задачи VerOscTest (рис. 6) была определена с помощью профилировщика AMD CodeAnalyst [13]: 48,0 % времени работы программы занимает выполнение умножения разреженной матрицы на вектор

(операция 1), а 34,5 % – решение систем линейных алгебраических уравнений с трехдиагональными матрицами методом прогонки при выполнении сглаживания в многосеточном предобуславливателе (операция 2). Решение систем линейных алгебраических уравнений методом BiCGStab без учета затрат времени на работу предобуславливателей (операция 3) занимает 16,5 % времени выполнения расчета, прочие операции – 1 %.

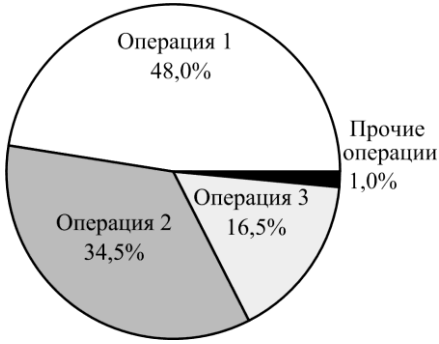


Рис. 6. Структура временны́х затрат при решении тестовой задачи VerOscTest
Fig. 6. Structure of time spent in solving test problem VerOscTest

Поскольку внутри операции 3 выполняются различные операции над векторами (вычисление скалярного произведения, нормы вектора и т.д.), сначала сосредоточимся на распараллеливании операций 1 и 2. Оценим максимальное ускорение, которое можно получить при распараллеливании только операции 1 или операций 1 и 2, при помощи закона Амдала [14], который гласит, что для системы из S вычислительных ялер максимально возможное ускорение программы с долей P параллельного кода и $(1-P)$ последовательного кода равно $\alpha = 1/((1-P)+P/S)$. Полученные оценки приведены в табл. 1. Они соответствуют случаю идеального (линейного) распараллеливания, при котором параллельный код на вычислительной системе с S ядрами выполняется в S раз быстрее. Реальное ускорение будет ниже, поскольку при переходе от последовательной программы к параллельной добавятся накладные расходы на поддержку многопоточных вычислений: расходы на создание задач и потоков, расходы на работу планировщика потоков, расходы на запуск и синхронизацию потоков, коммуникационные издержки на передачу информации между потоками, издержки в виде дисбаланса загрузки ядер из-за точек синхронизации или конфликтов в конвейере процессора и планировщике операционной системы. Далее для распараллеливания вычислений будем использовать такие технологии параллельного программирования, как Intel® Cilk™ Plus [15], Intel® TBB [16], OpenMP (реализация из Intel® Parallel Studio XE 2015,

стандарт 4.0). Эти технологии предполагают, что пользователь при помощи ключевых слов лишь обозначает задачи, выполняемые параллельно, а организация управления потоками и работы с ними определяются самой технологией. Таким образом, вышеперечисленные накладные расходы на поддержку многопоточности, а, значит, и реальное ускорение, зависят от используемой технологии параллельного программирования.

Табл. 1. Максимально возможное ускорение при распараллеливании операций 1 и 2
Table. 1. The maximum possible acceleration when parallelizing operations 1 and 2

Распараллеливаемые операции	*P*	Максимальное ускорение, раз		
		2 ядра	4 ядра	8 ядер
1	0,480	1,316	1,563	1,724
1, 2	0,825	1,702	2,623	3,596

Получить оценки ожидаемого ускорения в зависимости от используемой технологии параллельного программирования позволяет инструмент Intel® Advisor [17]. Для этого необходимо подключить заголовочный файл `advisor-annotate.h` и выделить содержимое операции при помощи команд `ANNOTATE_SITE_BEGIN()` и `ANNOTATE_SITE_END()`. Для операции 1 таким образом было получено, что наибольшее ускорение при использовании любой из трех рассматриваемых технологий прогнозируется при расчете на системе с четырьмя ядрами. Согласно прогнозу, использование OpenMP и Intel® TBB приведет к замедлению работы программы, а Intel® Cilk™ Plus – к незначительному ускорению (на 20 %). Тем не менее, поскольку оценки Intel® Advisor являются приблизительными, представляется целесообразной поддержка в разрабатываемом приложении всех перечисленных технологий параллельного программирования и переключение между ними при помощи директив препроцессора по определениям `LS_STAG_USE_CILK`, `LS_STAG_USE_OMP` и `LS_STAG_USE_TBB`.

5. Оптимизация и распараллеливание вычислений

Произведем преобразование последовательного кода в параллельный при помощи рассматриваемых технологий параллельного программирования на примере операции 1 – умножения разреженной матрицы на вектор, хранящийся в массиве `multiplier`, с сохранением результата в массив `Vector`. Разреженная матрица хранится в формате CSR [18]: элементы матрицы хранятся в массиве `m_Cell`, портрет – в массиве `m_Portrait`, а индексы элементов, с которых начинаются строки матрицы – в массиве `m_Num`. Последовательная реализация метода имеет следующий вид:

```
for(int i = 0; i < size; i++)
```

```
{ double aux = 0.0;
  for(int j = m_Num[i]; j < m_Num[i+1]; j++)
      aux += m_Cell[j] * multiplier[m_Portrait[j]];
  Vector[i] = aux;
};
```

Итерации внешнего цикла являются независимыми, поэтому для распараллеливания вычислений достаточно при использовании технологии Intel® Cilk™ Plus вместо `for` написать `cilk_for`, а при использовании OpenMP – перед `for` поставить `#pragma omp parallel for`. При использовании технологии Intel® TBB код параллельной версии данного метода получается более громоздким, но при помощи лямбда-выражений из стандарта C++11 его можно записать следующим образом:

```
tbb::parallel_for(tbb::blocked_range<int>(0,size),
  [=](const tbb::blocked_range<int>& r)
     { for(int i = r.begin(); i < r.end(); i++)
       { double aux = 0.0;
         for(int j = m_Num[i]; j < m_Num[i+1]; j++)
           aux += m_Cell[j] * multiplier[m_Portrait[j]];
         Vector[i] = aux;
       }
     }            )
```

Количество ядер для Intel® Cilk™ Plus и OpenMP задается при помощи `__cilkrts_set_param` и `omp_set_num_threads` соответственно, а для Intel® TBB указывается при создании планировщика – экземпляра класса `tbb::task_scheduler_init`.

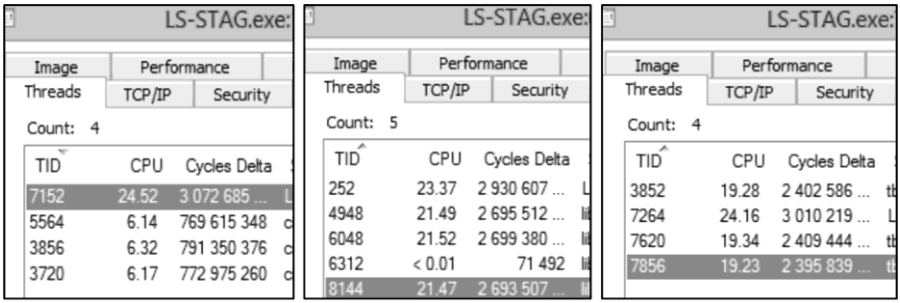
Поскольку Intel® Advisor показал, что оптимальным является использование вычислительной системы из четырех ядер, тестовая задача VerOscTest была решена на PC1 с использованием четырех логических ядер при помощи программы с распараллеленной операцией 1. Время счета и достигнутое ускорение представлено в табл. 2. Наибольшее ускорение, достаточно близкое к полученной по закону Амдала оценке (в 1,563 раза), достигнуто при использовании Intel® Cilk™ Plus. При этом для всех трех технологий полученные ускорения достаточно сильно превышают оценки Intel® Advisor. Также не подтвердился прогноз Intel® Advisor о том, что технология Intel® TBB позволит получить бóльшее ускорение, чем OpenMP. Это свидетельствует о целесообразности предусмотренной в разработанном программном комплексе возможности задания используемой технологии параллельного программирования при помощи определений препроцессора.

Табл. 2. Время счета на PC1 с использованием 4 логических ядер и ускорение при распараллеливании операции 1

Table. 2. Time of calculatiion on PC1 with 4 logical cores and acceleration when parallelizing operations 1

Технология	Intel® Cilk™ Plus	OpenMP	Intel® TBB
Время счета, с	2548	2713	2870
Ускорение, раз	1,557	1,463	1,383

Сравним также работу планировщиков (диспетчеров) потоков рассматриваемых технологий параллельного программирования. Для этого используем программу ProcessExplorer [19], предназначенную для мониторинга процессов в системе. Информация о потоках приложения при расчете на четырех логических ядрах представлена на рис. 7.



(а) *(б)* *(в)*

Рис. 7. Информация о потоках приложения при расчете на четырех логических ядрах с использованием: (а) Intel® Cilk™ Plus; (б) OpenMP; (в) Intel® TBB
Fig. 7. Information about the application threads based on four logical cores using: (а) Intel® Cilk ™ Plus; (б) OpenMP; (в) Intel® TBB

При использовании рассматриваемых технологий потоки создаются один раз, поскольку их идентификаторы (TID) не изменяются на протяжении всего расчета. При этом диспетчеры потоков работают в основном потоке приложения, однако диспетчер потоков OpenMP создает дополнительный дочерний поток (на рис. 7, *б* это поток с TID, равным 6312), который занимается мониторированием рабочих потоков (судя по низкой загрузке потока – обслуживанием конфликтов). В результате на основной диспетчер потоков OpenMP ложатся дополнительные накладные расходы по обслуживанию этого мониторирующего потока, что, по-видимому, приводит к увеличению времени проведения расчета и к меньшему по сравнению с Intel® Cilk™ Plus ускорению (табл. 2). Тем не менее, время счета при использовании OpenMP оказалось меньше, чем при использовании Intel® TBB. Это свидетельствует о том, что основной диспетчер потоков OpenMP из Intel Parallel Studio 2015 реализован достаточно эффективно. Также необходимо отметить, что при использовании технологии Intel® Cilk™ Plus время проведения расчета оказалось наименьшим, хотя загрузка ядер рабочими дочерними потоками приложения была примерно в 3 раза ниже, чем при

использовании OpenMP и Intel® TBB. Из этого можно сделать вывод, что диспетчер потоков Intel® Cilk™ Plus эффективно управляет средствами конвейеризации и кеширования процессора, снижая нагрузку на ядра, а также берет на себя бóльшую часть затрат по управлению критическими секциями, семафорами и другими средствами синхронизации потоков, в то время как планировщики потоков OpenMP и Intel® TBB часть функций синхронизации перекладывают на рабочие дочерние потоки, отрывая их от основной работы и создавая бóльшую загрузку ядер при меньшей эффективности.

Перейдем к распараллеливанию операции 2 – сглаживающих итераций многосеточного решателя. В качестве сглаживателя используется метод ADLJ – Alternating Damped Line Jacobi [20]. Данный метод предполагает решение систем линейных алгебраических уравнений с трехдиагональными матрицами, сформированными из матрицы исходной системы. Для полученных матриц хранится LU-разложение в массивах `L`, `D` и `U` (нижняя, главная и верхняя диагонали соответственно). Алгоритм вычисления решения `sol` системы с построенной трехдиагональной матрицей и правой частью `right_side` имеет следующий вид:

```
sol[0] = right_side[0] / D[0];
for(int i = 1, r = 0; i < NM; i++, r++)
    sol[i] = (right_side[i] - L[r] * sol[r]) / D[i];
for(int i = NM - 2; i > -1; i--)
    sol[i] -= U[i] * sol[i+1];
```

В таком виде алгоритм не может быть распараллелен, поскольку итерации циклов являются зависимыми: при вычислении значения элемента `sol[i]` используются значения `sol[r]` и `sol[i+1]`, полученные на предыдущих итерациях. Однако, поскольку исходная матрица была получена при дискретизации с пятиточечной структурой шаблона на прямоугольной сетке $N \times M$, сформированная из нее трехдиагональная матрица распадается на M независимых блоков размера $N \times N$, и алгоритм может быть переписан следующим образом:

```
for(int j = 0; j < M; j++) // блоки
{ int q = j * N; // номер первой строки блока
  sol[q] = right_side[q] / D[q];
  for(int r = q, i = r + 1; i < r + N; i++, r++)
      sol[i] = (right_side[i] - L[r] * sol[r]) / D[i];
  for(int q2 = q - 1, i = q2 + N - 1; i > q2; i--)
      sol[i] -= U[i] * sol[i+1];
}
```

Теперь итерации внешнего цикла являются независимыми, и его можно распараллелить аналогично циклу из операции 1. В табл. 3 представлены значения времени счета и достигнутых ускорений при решении задачи

VerOscTest на PC1 при помощи программы с распараллеленными операциями 1 и 2. Наибольшее ускорение, достаточно близкое к оценке по закону Амдала (в 2,623 раза), как и в случае распараллеливания операции 1, достигнуто при использовании технологии Intel® Cilk™ Plus.

Табл. 3. Время счета на PC1 с использованием четырех логических ядер и ускорение при распараллеливании операции 1 и 2
Table. 2. Time of calculatiion on PC1 with 4 logical cores and acceleration when parallelizing operations 1 and 2

Технология	Intel® Cilk™ Plus	OpenMP	Intel® TBB
Время счета, с	1547	1698	1774
Ускорение, раз	2,565	2,337	2,237

После распараллелиливания двух наиболее трудоемких операций было произведено распараллеливание ряда операций с векторами, на которые приходится бóльшая часть времени выполнения операции 3, и перестроение некоторых разреженных матриц и сеточных функций, происходящее при движении погруженной границы. Помимо продемонстрированного на примере операции 1 распараллеливания цикла `for` с независимыми итерациями, использовались приемы распараллеливания циклов с редукцией. Покажем различия в распараллеливании циклов такого типа при использовании различных технологий параллельного программирования на примере распараллеливания расчета евклидовой нормы вектора, элементы которого хранятся в массиве `Vector`:

```
double getNorm()
{ ElemType aux = 0;
  for(int i = 0; i < size; i++)
  { ElemType cur = Vector[i]; aux += cur * cur; }
  return sqrt((double)aux);
}
```

При использовании OpenMP перед ключевым словом `for` необходимо добавить директиву `#pragma omp parallel for reduction(+:aux)`. При использовании технологии Intel® Cilk™ Plus алгоритм принимает следующий вид:

```
double getNorm()
{ cilk::reducer<cilk::op_add<ElemType>> aux(0);
  cilk_for(int i = 0; i < size; i++)
  { ElemType cur = Vector[i]; *aux += cur * cur; }
  return sqrt((double)aux.get_value());
}
```

При использовании технологии Intel® TBB код получается более громоздким, но может быть записан при помощи лямбда-выражений:

```
double getNorm()
{ ElemType res =
  tbb::parallel_reduce(tbb::blocked_range<int>(0,size),
     ElemType(0), [=](const tbb::blocked_range<int>& r,
                                 ElemType aux)->ElemType
       {
         for(int i = r.begin(); i < r.end(); i++)
         { ElemType cur = Vector[i]; aux += cur * cur; }
         return aux;
       }, [](ElemType x, ElemType y)->ElemType
                                    { return x + y; } );
  return sqrt((double)res);
}
```

Кроме того, была проведена оптимизация кода. После этого время выполнения расчетов значительно уменьшилось даже при использовании одного ядра (табл. 4). Поскольку диспетчеры потоков Intel® Cilk™ Plus, OpenMP и Intel® TBB не отключаются при работе приложения на одном ядре, особенности их работы, рассмотренные выше, напрямую сказываются на быстродействии даже в однопоточной версии. Таким образом, представленные в табл. 4 данные подтверждают предположение о том, что диспетчер потоков Intel® Cilk™ Plus реализован эффективнее планировщиков потоков OpenMP и Intel® TBB.

Табл. 4. Время счета на PC1 с использованием одного ядра и полученное в результате оптимизации кода ускорение

Table. 4. Ttime of calculation on PC1 using a single core and the acceleration with code optimization

Технология	Intel® Cilk™ Plus	OpenMP	Intel® TBB
Время счета, с	1958	2207	2318
Ускорение, раз	2,027	1,798	1,712

Для исследования масштабируемости комплекса была проведена серия вычислительных экспериментов (рис. 8). Наименьшее время проведения расчета независимо от технологии параллельного программирования и на PC1, и на PC2 получается при использовании четырех логических ядер. На PC2, так же как и на PC1, при расчете на одном ядре быстродействие приложения, использующего Intel® Cilk™ Plus, оказывается выше чем у приложения, использующего OpenMP, а дольше всего решение тестовой задачи VerOscTest идет при использовании Intel® TBB. Однако приложение с Intel® Cilk™ Plus на

обеих рабочих станциях масштабируется хуже приложения, в котором используется OpenMP, а оно, в свою очередь, – хуже приложения с Intel® TBB. Из-за этого при проведении расчета с использованием четырех логических ядер на PC1 получаем, что приложение с Intel® TBB опережает приложение с OpenMP (при этом наименьшее время счета получается по-прежнему при использовании Intel® Cilk™ Plus), а на PC2 приложение, использующее OpenMP, по быстродействию опережает приложение с Intel® Cilk™ Plus, уступающее и приложению с Intel® TBB. Таким образом, подтверждается необходимость поддержки программным комплексом всех рассматриваемых технологий параллельного программирования и возможности выбора пользователем используемой технологии.

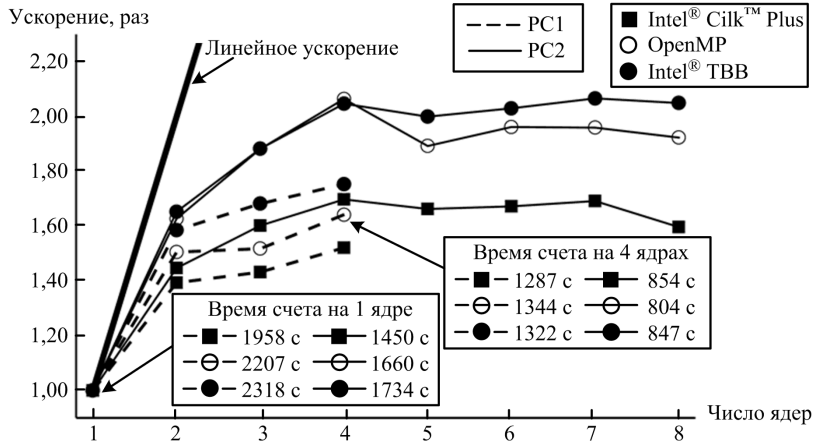


Рис. 8. Масштабируемость программного комплекса «LS-STAG_turb» после оптимизации кода и распараллеливания операций
Fig. 8. Scalability of the «LS-STAG_turb» software complex after the code optimization and parallelization of operations

6. Выбор решателя и исследование эффективности его реализации

Также возможно уменьшение продолжительности расчета за счет модификаций решателя. В описанных выше экспериментах для решения систем линейных алгебраических уравнений использовался метод BiCGStab [20] (метод бисопряженных градиентов со стабилизацией) с предобуславливанием. Для разностного аналога уравнения Гельмгольца использовалось ILU-предобуславливание [18], а для разностного аналога уравнения Пуассона – многосеточное предобуславливание [20]. При этом на задаче VerOscTest решение разностных аналогов уравнения Гельмгольца (70744 и 70800 уравнений) с точностью $\varepsilon = 10^{-6}$ получается за 2 итерации,

а решение разностного аналога уравнения Пуассона (70744 и 70800 уравнений) с той же точностью – за 7–20 итераций.

Заменим метод BiCGStab на метод FGMRES (гибкий метод обобщенных невязок) без изменения предобуславливателей. Метод FGMRES, как и метод BiCGStab, относится к методам крыловского типа [20]. Основное различие между этими двумя методами заключается в способе построения базиса в подпространстве Крылова: в методе BiCGStab для этого используется биортогонализация Ланцоша, а в FGMRES – ортогонализация Арнольди [20]. После оптимизации и реализации алгоритма метода FGMRES [20] была проведена серия вычислительных экспериментов для исследования масштабируемости алгоритма (рис. 9, черные линии). Использование разработанной реализации метода FGMRES позволило получить ускорение по сравнению с решением систем методом BiCGStab на одном ядре в среднем в 2,235 раза, а на четырех логических ядрах – в 1,829 раза, поскольку программа с новым решателем несколько хуже масштабируется. При этом решение разностного аналога уравнения Гельмгольца при использовании как метода BiCGStab, так и метода FGMRES получается за 2 итерации, в то время как решение разностного аналога уравнения Пуассона при использовании метода FGMRES получается за 4–7 итераций, что говорит о более быстрой сходимости метода FGMRES по сравнению с методом BiCGStab.

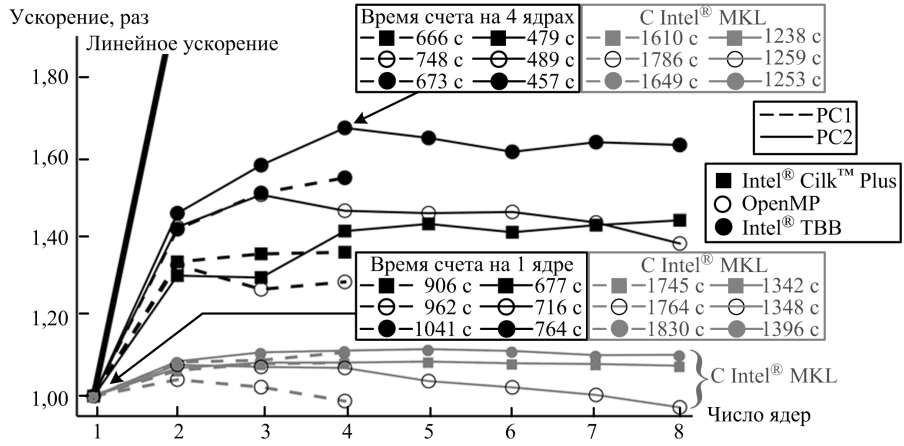


Рис. 9. Масштабируемость комплекса «LS-STAG_turb» при решении тестовой задачи VerOscTest (черные линии) и сравнение с решателем из Intel® MKL (серые линии)
Fig. 9. Scalability of the «LS-STAG_turb» complex in solving the test problem VerOscTest (black lines) and the comparison with the solver from the Intel® MKL (gray lines)

Наилучшее быстродействие при расчетах на одном ядре демонстрируют приложения, использующие технологию Intel® Cilk™ Plus, однако при расчете на PC2 с четырьмя ядрами благодаря хорошей масштабируемости наименьшая

продолжительность расчета получается при использовании Intel® TBB. Отметим, что по сравнению с исходным последовательным кодом время проведения расчета уменьшилось примерно в 5 раз при использовании одного ядра и в 7 раз при использовании четырех ядер.

Сравним эффективность разработанного решателя с аналогом из библиотеки Intel® Math Kernel Library (MKL) 11.2 [21], содержащей реализацию метода FGMRES. Эта библиотека оптимизирована для работы с разными процессорами Intel и обеспечивает использование их расширенных возможностей. Из предобуславливателей Intel® MKL содержит только ILU-предобуславливание, поэтому используем его для решения не только разностного аналога уравнения Гельмгольца, но и для решения разностного аналога уравнения Пуассона. При этом число итераций, совершаемых при решении разностного аналога уравнения Пуассона, возрастает до 25–140. Масштабируемость полученного алгоритма оказывается очень низкой (рис. 9, серые линии), поэтому на четырех ядрах он значительно уступает разработанному решателю. Использование собственной реализации метода FGMRES позволило получить ускорение по сравнению с решателем из Intel® MKL на одном ядре в среднем в 1,869 раза, а на четырех ядрах – в 2,526 раза.

7. Заключение

Разработан параллельный программный комплекс «LS-STAG_turb» для моделирования движения профилей в потоке вязкой несжимаемой среды. В данном комплексе реализован высокоточный метод погруженных границ LS-STAG и разработанные модификации данного метода для и решения сопряженных задач гидроупругости с использованием моделей турбулентности Смагоринского, Спаларта – Аллмараса, $k-\varepsilon$, $k-\omega$ и $k-\omega$ SST в рамках RANS, LES и DES подходов к моделированию турбулентности. Разработанный программный комплекс поддерживает использование таких технологий параллельного программирования, как Intel® Cilk™ Plus, Intel® TBB и OpenMP. Использование метода FGMRES для решения систем линейных алгебраических уравнений позволило достичь существенного сокращения времени проведения расчета (примерно в 2 раза) по сравнению с методом BiCGStab. Кроме того, разработанная программная реализация метода FGMRES оказалась эффективнее аналогичного решателя из библиотеки Intel® MKL, как при проведении расчетов на одном ядре, так и при использовании нескольких ядер.

Главным направлением дальнейшего развития параллельного программного комплекса «LS-STAG_turb» является разработка параллельного алгоритма решения задачи методом LS-STAG, основанного на декомпозиции расчетной области. При этом для эффективного решения вспомогательнх задач в подобластях будет использоваться представленная параллельная реализация решателя. Также планируется разработать препроцессор для подготовки

файлов с исходными данными. Кроме того, в перспективе планируется разработка и реализация метода LS-STAG для решения трехмерных задач.

Список литературы

[1]. Mittal R., Iaccarino G. Immersed boundary methods. Annu. Rev. Fluid Mech. 2005. № 37. P. 239–261.

[2]. Cheny Y., Botella O. The LS-STAG method: A new immersed boundary/level-set method for the computation of incompressible viscous flows in complex moving geometries with good conservation properties. J.Comp. Phys.2010. №229. P.1043-1076.

[3]. Osher S., Fedkiw R.P. Level set methods and dynamic implicit surfaces. N. Y.: Springer, 2003. 273 p.

[4]. Puzikova V.V., Marchevsky I.K. Extension of the LS-STAG immersed boundary method for RANS-based turbulence models and its application for numerical simulation in coupled hydroelastic problems. Proc. VI International Conference on Coupled Problems in Science and Engineering. Venice. 2015. P. 532–543.

[5]. Puzikova V.V. On generalization of the LS-STAG immersed boundary method for Large Eddy Simulation and Detached Eddy Simulation. Proc. Advanced Problems in Mechanics International Summer School-Conference. St.-Petersburg. 2015. P. 411-417.

[6]. Marchevsky I., Puzikova V. Application of the LS-STAG Immersed Boundary Method for Numerical Simulation in Coupled Aeroelastic Problems. Proc. 11th World Congress on Computational Mechanics, 5th European Conference on Computational Mechanics, 6th European Conference on Computational Fluid Dyn. Barcelona. 2014. P.1995-2006.

[7]. Пузикова В.В. Архитектура программного комплекса для численного моделирования движения профилей в потоке вязкой несжимаемой среды методом LS-STAG. Молодежный научно-технический вестник. 2014. № 7. С. 11.

[8]. Taneda S. Visual observation of the flow past a circular cylinder performing a rotary oscillation. J. Phys. Soc. Japan. 1978. Vol. 45, № 3. P. 1038–1043.

[9]. Марчевский И.К., Пузикова В.В. Моделирование обтекания кругового профиля, совершающего вращательные колебания, методом LS-STAG. Вестник МГТУ им. Н.Э. Баумана. Естественные науки. 2014. № 3. С. 93–107.

[10]. Quarteroni A., Valli A. Domain decomposition methods for partial differential equations. Oxford: Clarendon Press, 1999. 360 p.

[11]. Ильин В.П., Кныш Д.В. Параллельные методы декомпозиции в пространствах следов. Вычислительные методы и программирование. 2011. Т. 12. С. 110–119.

[12]. Пузикова В.В. Построение функции уровня для профиля произвольной формы при моделировании его обтекания методом LS-STAG. Инженерный журнал: наука и инновации. 2013. № 4. С. 8.

[13]. CodeAnalyst Performance Analyzer. URL: http://developer.amd.com/tools-and-sdks/archive/amd-codeanalyst-performance-analyzer/ (accessed: 25.10.2015).

[14]. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. М.: Изд-во Моск. ун-та, 2010. 544 с.

[15]. Intel® Cilk™ Plus. URL: https://software.intel.com/ru-ru/node/522579 (accessed: 25.10.2015).

[16]. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. Sebastopol: O'Reilly, 2007. 336 p.

[17]. Intel® Advisor Tutorials. URL: https://software.intel.com/en-us/articles/advisorxe-tutorials (accessed: 25.10.2015).

[18]. Баландин М.Ю., Шурина Э.П. Методы решения СЛАУ большой размерности. Новосибирск: Изд-во НГТУ, 2000. 70 с.
[19]. Process Explorer v16.05. URL: https://technet.microsoft.com/ru-ru/sysinternals/bb896653.aspx (accessed: 25.10.2015).
[20]. Saad Y. Iterative Methods for Sparse Linear Systems. N.Y.: PWS Publ., 1996. 547 p.
[21]. Intel® Math Kernel Library – Documentation . URL: https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation (accessed: 25.10.2015).

Realization of parallel computations in the software package «LS-STAG_turb» for viscous incompressible flow simulation on systems with shared memory

V. Puzikova <valeria.puzikova@gmail.com>
BMSTU, 5 2nd Baumanskaya st., Moscow, 105005, Russian Federation

Abstract. Immersed boundary methods have become popular in Computational Fluid Dynamics over recent years for simulating flows through complex solid geometries and in coupled hydroelasctic problems. The advantage of these methods over a method with a body-fitted mesh is their computational efficiency: they do not require regridding when domain shape changes in the simulation process due to hydroelastic body motion. The LS-STAG method for viscous incompressible flows simulation combines the advantages of immersed boundary methods, the marker and cells (MAC) method and level-set method. The LS-STAG method and its modifications for numerical simulation in coupled hydroelastic problems and for turbulence simulation by using RANS, LES and DES approaches are implemented in the software package «LS-STAG_turb». This software allows to simulate viscous incompressible flows around a moving airfoil of arbitrary shape or airfoils system with one or two degrees of freedom. For example, it allows to simulate rotors autorotation and airfoils system wind resonance. These phenomena are characterized by high velocities of airfoils and high values of local Reynolds number. So, extremely small space and time steps are required to obtain accurate quantitative results. It leads to significant increase in computational cost. To decrease it, the «LS-STAG_turb» parallel version is developed. Intel® Cilk™ Plus, Intel® TBB and OpenMP parallel programming technologies are used. Also, serial code sections are optimized. The FGMRES method usage for linear algebraic equations systems solving allows to achieve 2-fold computation time reduction in comparison with the BiCGStab method. In addition, the developed software implementation of the FGMRES method is more efficient than the similar solver implemented in Intel® MKL library both for single-core and multi-core computations.

Keywords: OpenMP technology; Intel® Cilk™ Plus technology; Intel® Threading Building Blocks technology; viscous incompressible flow; immersed boundary method.

DOI: 10.15514/ISPRAS-2016-28(1)-13

For citation: Puzikova V. Realization of parallel computations in the software package «LS-STAG_turb» for viscous incompressible flow simulation on systems with shared memory. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 221-242 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-13

References

[1]. Mittal R., Iaccarino G. Immersed boundary methods. *Annu. Rev. Fluid Mech*. 2005. no. 37. P. 239–261.

[2]. Cheny Y., Botella O. The LS-STAG method: A new immersed boundary/level-set method for the computation of incompressible viscous flows in complex moving geometries with good conservation properties. *J.Comp.Phys.*2010. no.229. P.1043-1076.

[3]. Osher S., Fedkiw R.P. Level set methods and dynamic implicit surfaces. *N. Y.: Springer*, 2003. 273 p.

[4]. Puzikova V.V., Marchevsky I.K. Extension of the LS-STAG immersed boundary method for RANS-based turbulence models and its application for numerical simulation in coupled hydroelastic problems. *Proc. VI International Conference on Coupled Problems in Science and Engineering*. Venice. 2015. P. 532–543.

[5]. Puzikova V.V. On generalization of the LS-STAG immersed boundary method for Large Eddy Simulation and Detached Eddy Simulation. *Proc. Advanced Problems in Mechanics International Summer School-Conference*. St.-Petersburg. 2015. P. 411-417.

[6]. Marchevsky I., Puzikova V. Application of the LS-STAG Immersed Boundary Method for Numerical Simulation in Coupled Aeroelastic Problems. *Proc. 11th World Congress on Computational Mechanics, 5th European Conference on Computational Mechanics, 6th European Conference on Computational Fluid Dyn.* Barcelona. 2014. P.1995-2006.

[7]. Puzikova V.V. Arkhitektura programmnogo kompleksa dlya chislennogo modelirovaniya dvizheniya profilei v potoke vyazkoi neszhimaemoi sredy metodom LS-STAG [Architecture of software package for numerical simulation of the motion airfoiles in the viscous incompressible flow by using the LS-STAG method]. *Molodezhnyi nauchno-tekhnicheskii vestnik* [*Youth Science and Technology Herald*]. 2014. no. 7. P. 11. (in Russian)

[8]. Taneda S. Visual observation of the flow past a circular cylinder performing a rotary oscillation. *J. Phys. Soc. Japan*. 1978. Vol. 45, no. 3. P. 1038–1043.

[9]. Marchevskii I.K., Puzikova V.V. Modelirovanie obtekaniya krugovogo profilya, sovershayushchego vrashchatel'nye kolebaniya, metodom LS-STAG [Flow-around simulation of circular cylinder performing rotary oscillations by LS-STAG method]. *Vestnik MGTU im. N.É. Baumana. Estestvennye nauki* [*Herald of the Bauman Moscow State technical University. Series "Natural Sciences"*]. 2014. no. 3. P. 93–107. (in Russian)

[10]. Quarteroni A., Valli A. Domain decomposition methods for partial differential equations. *Oxford: Clarendon Press*, 1999. 360 p.

[11]. Il'in V.P., Knysh D.V. Parallel'nye metody dekompozitsii v prostranstvakh sledov [Parallel decomposition methods in traces spaces]. *Vychislitel'nye metody i programmirovanie* [*Computational methods and programming*]. 2011. Vol. 12. P. 110–119. (in Russian)

[12]. Puzikova V.V. Postroenie funktsii urovnya dlya profilya proizvol'noi formy pri modelirovanii ego obtekaniya metodom LS-STAG [Construction of level-set function for an airfoil of arbitrary topology whem modelimg a flow past it using the LS-STAG method]. *Inzhenernyi zhurnal: nauka i innovatsii* [*Engineering Journal: science and innovation*]. 2013. no. 4. P. 8. (in Russian)

[13]. CodeAnalyst Performance Analyzer. URL: http://developer.amd.com/tools-and-sdks/archive/amd-codeanalyst-performance-analyzer/ (accessed: 25.10.2015).

[14]. Gergel' V.P. Vysokoproizvoditel'nye vychisleniya dlya mnogoprotsessornykh mnogoyadernykh sistem [High-performance computing for multi-core systems]. *Moscow: Moscow University Publ.*, 2010. 544 p. (in Russian)
[15]. Intel® Cilk™ Plus. URL: https://software.intel.com/ru-ru/node/522579 (accessed: 25.10.2015).
[16]. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. *Sebastopol: O'Reilly*, 2007. 336 p.
[17]. Intel® Advisor Tutorials. URL: https://software.intel.com/en-us/articles/advisorxe-tutorials (accessed: 25.10.2015).
[18]. Balandin V. Yu., Shurina E.P. Metody resheniya SLAU bol'shoi razmernosti [Methods for solving linear systems of large dimension]. *Novosibirsk: Novosibirsk State Technical University Publ.*, 2000. 70 p. (in Russian)
[19]. Process Explorer v16.05. URL: https://technet.microsoft.com/ru-ru/sysinternals/bb896653.aspx (accessed: 25.10.2015).
[20]. Saad Y. Iterative Methods for Sparse Linear Systems. *N.Y.: PWS Publ*., 1996. 547 p.
[21]. Intel® Math Kernel Library – Documentation . URL: https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation (accessed: 25.10.2015).

Свободное программное обеспечение для моделирования жидкости со свободной поверхностью

Е.В. Давыдова <alenka-davidova@rambler.ru>
В.Н. Корчагова <ko_viktoria@inbox.ru>
МГТУ им. Н.Э. Баумана,
105005, Россия, г. Москва, ул. 2-я Бауманская, дом 5, стр. 1

Аннотация. Задачи течения вязкой несжимаемой жидкости со свободной поверхностью представляют собой отдельный класс задач механики сплошной среды и имеют большую практическую значимость. При моделировании таких процессов исследователь сталкивается с достаточно большим числом особенностей и ограничений, критически важных для получения корректного решения. Целью данной работы является обзор существующих численных методов, которые можно применить к решению задач течения жидкости со свободной поверхностью, и программных комплексов с открытым исходным кодом, в которых эти методы реализованы, а также выявление границ применимости рассмотренных программных комплексов. Были рассмотрены несколько численных методов (Volume of Fluid, Smoothed Particle Hydrodynamics, Particle Finite Element Method v.2), реализованные в пяти разных пакетах программ с открытым исходным кодом: OpenFOAM, Gerris, pySPH, DualSPHysics, Kratos. В качестве тестовых задач были выбраны следующие примеры: обрушение колонны жидкости и падение капли в слой жидкости. Решения, полученные в выбранных пакетах, были сопоставлены с известными результатами экспериментов. Проводилось сравнение времени, затраченного на решение задач в различных пакетах. Наилучшие результаты для выбранных тестов показали пакеты OpenFOAM и Gerris: в них содержатся необходимые для решения задач инструменты – возможность расчета задач в двумерной, трехмерной либо осесимметричной постановке, а также корректный учет поверхностного натяжения жидкости. При этом пакет Gerris дает существенное ускорение решения "крупных" задач за счет использования динамически перестраиваемых сеток. Пакет DualSPHysics направлен на решение задач прибрежной инфраструктуры, где рассматривают, как правило, трехмерные задачи и нет необходимости учета поверхностного натяжения. Пакет pySPH разрабатывался с целью наглядной демонстрации работы численного метода SPH, поэтому исходный код записан на языке Python без оптимизации. Пакет Kratos, в частности, модуль PFEM-2, в настоящий момент находится в разработке, поэтому некоторые возможности в нем еще не реализованы.

Ключевые слова: гидродинамика; переменная область течения; жидкость со свободной поверхностью; VOF; SPH; PFEM; открытый исходный код.

DOI: 10.15514/ISPRAS-2016-28(1)-14

Для цитирования: Давыдова Е.В., Корчагова В.Н. Свободное программное обеспечение для моделирования жидкости со свободной поверхностью. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 243-258. DOI: 10.15514/ISPRAS-2016-28(1)-14

1. Введение

Исследование течений жидкости со свободной поверхностью представляет собой отдельный класс задач механики сплошной среды. Такие задачи имеют большую практическую значимость и возникают, к примеру, при моделировании плотин, дамб, решении экологических задач (загрязнение поверхностей, распространение примесей и т. д.). Как правило, при решении подобных задач рассматривают процессы брызгообразования, волнообразования, возникновения струйных течений и т. п.

С математической точки зрения специфика таких задач состоит в том, что область течения является переменной, форма ее поверхности заранее неизвестна, и на этой поверхности задаются граничные условия специального вида. Во многих случаях в различные моменты времени область течения может состоять из разного числа связных компонент.

Даже в сравнительно простом случае моделирования ламинарного течения несжимаемой изотермической среды разработчик численного алгоритма или инженер-исследователь, выполняющий расчеты, может столкнуться с рядом условий, соблюдение которых может быть критически важным для успешного решения задачи. Этими условиями являются обеспечение консервативности и устойчивости разностной схемы, а также как можно более низкой ее диссипативности и монотонности в области разрыва (межфазной границы). Не менее важными для практики являются требования высокой вычислительной производительности и масштабируемости модели.

К дополнительным факторам, повышающим трудоемкость решения подобных задач, могут относиться необходимость учета движения границ расчетной области; высокие значения отношения плотностей жидкой и газообразной фаз; требования тщательного разрешения мелкомасштабных структур (пленок, пузырьков, капель), характерный размер которых может быть намного меньше, чем характерный масштаб длины в основной задаче и т. п. Весьма нетривиальным может быть обеспечение возможности сопряжения решения или вычислительного алгоритма в целом с математическими моделями из других разделов механики сплошной среды в рамках решения многодисциплинарных задач.

К сожалению, на практике удовлетворить всем перечисленным требованиям в полной мере невозможно. Поэтому численный метод решения поставленной задачи выбирается на основе некоторого компромисса в зависимости от целей исследования.

Целью данной работы является обзор существующих численных методов, которые можно применить к решению задач моделирования течения жидкости со свободной поверхностью, и программных комплексов с открытым исходным кодом, в которых эти методы реализованы, а также выявление границ применимости рассмотренных программных комплексов.

В данной работе рассмотрены три численных метода:

- метод контрольных объемов с расчетом переноса объёмной доли жидкой фазы (FVM + VOF);
- метод сглаженных частиц (SPH);
- метод конечных элементов с частицами (PFEM).

Рассмотрены 5 программных средств, реализующих три вышеупомянутых численных метода решения уравнений Навье – Стокса для моделирования несжимаемых течений со свободной поверхностью: OpenFOAM, Gerris, pySPH, DualSPHysics, Kratos.

Для сравнения программ было выбрано два расчетных случая: задача моделирования обрушения двумерного столба жидкости и задача моделирования падения капли в слой жидкости.

2. Определяющие соотношения

Рассмотрим задачу о расчете течения вязкой несжимаемой жидкости со свободной поверхностью. Для описания течений жидкости и газа с учетом выбранных допущений о несжимаемости, изотермичности, наличии вязких сил и сил поверхностного натяжения используются следующие уравнения:

– условие несжимаемости:

$$\nabla \cdot \vec{U} = 0; \tag{1}$$

– уравнение сохранения импульса (уравнение Навье – Стокса), учитывающее наличие массовых сил:

$$\rho\left(\frac{\partial \vec{U}}{\partial t} + (\vec{U} \cdot \nabla)\vec{U}\right) = -\nabla p + \nabla \cdot \hat{\tau} + \rho\vec{g}. \tag{2}$$

Здесь $\vec{U}$ — поле скоростей среды; p — поле давления; $\hat{\tau} = \mu(\nabla\vec{U} + \nabla\vec{U}^T)$ — тензор вязких напряжений в соответствующей фазе; ρ — плотность; μ — коэффициент динамической вязкости; $\vec{g}$ — ускорение свободного падения.

Начальное распределение скоростей считается заданным.

Граничные условия на бесконечности записываются следующим образом:

$$\vec{U} \to 0, \ \nabla p \to -\rho\vec{g} \ \text{ при } |\vec{r}| \to \infty.$$

На поверхностях раздела фаз ставятся следующие условия [1]:

$$\vec{U}_{\text{г}} = \vec{U}_{\text{ж}}, \qquad (p_{\text{г}} - p_{\text{ж}})\vec{n} + (\hat{\tau}_{\text{г}} - \hat{\tau}_{\text{ж}}) \cdot \vec{n} = -2\sigma\kappa\vec{n},$$

где κ — средняя кривизна поверхности раздела фаз; σ — коэффициент поверхностного натяжения; индекс "ж" обозначает значение физической величины в жидкости, индекс "г" — значение физической величины в газе.

Предположение о несжимаемости газа справедливо в случае, когда $U \ll c_{\text{г}}$, где $c_{\text{г}}$ – скорость звука в газовой фазе.

3. Численные методы и программные комплексы

3.1. Метод контрольных объемов с расчетом переноса объёмной доли жидкой фазы

Суть подхода, основанного на расчете переноса объемной доли жидкости — Volume of Fluid (VOF), заключается в следующем: для аппроксимации скачков физических величин в рамках метода контрольных объемов вблизи поверхности раздела фаз используется «приближение смеси», «сглаживающее» границу раздела фаз в пределах некоторой области и позволяющее искать решение методом «сквозного счета». Для этого вводят маркерную функцию γ (рис. 1), физический смысл которой — объемная доля жидкости [2]:

- $\gamma = 0$ в областях, заполненных газом;
- $\gamma = 1$ в областях, заполненных жидкостью;
- $0 < \gamma < 1$ в области перехода между фазами.

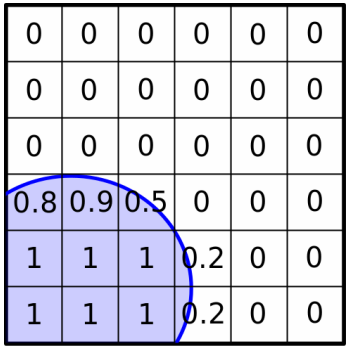


Рис. 1. Значения маркерной функции в ячейках сетки
Fig.1. Values of marker function in the cells of grid

Уравнения (1) и (2) рассматриваются для смеси, скорость, плотность и динамическая вязкость которой определяются следующим образом:

$$\vec{U} = \gamma \vec{U}_{\text{ж}} + (1-\gamma)\vec{U}_{\text{г}};$$
$$\rho = \gamma \rho_{\text{ж}} + (1-\gamma)\rho_{\text{г}};$$
$$\mu = \gamma \mu_{\text{ж}} + (1-\gamma)\mu_{\text{г}}.$$

Для маркерной функции решают уравнение ее переноса жидкой фазой

$$\frac{\partial \gamma}{\partial t} + (\vec{U}_{\text{ж}} \cdot \nabla)\gamma = 0,$$

которое можно записать в виде:

$$\frac{\partial \gamma}{\partial t} + \nabla \cdot (\vec{U}\,\gamma) + \nabla \cdot (\vec{V}\,\gamma(1-\gamma)) = 0, \qquad (3)$$

где $\vec{V}$ — относительная скорость движения тяжелой фазы (жидкости) относительно легкой (газа). Эта величина является неизвестной, поэтому при аппроксимации соотношения (3) ее заменяют на некоторое поле скоростей, вводимое искусственным образом. Введенное поле скоростей «сжимает» границу раздела фаз, не допуская слишком сильного ее «размазывания». При этом оно мало влияет на решение в целом, т. к. его действие проявляется в тонком слое вокруг поверхности раздела.

Для численного интегрирования определяющих соотношений используется метод контрольного объема. Подробно эта процедура описана в [2].

3.2. Метод сглаженных частиц

Метод сглаженных частиц — Smoothed Particle Hydrodynamics (SPH) принадлежит к классу бессеточных методов. Суть метода заключается в том, что сплошная среда, находящаяся в расчетной области, аппроксимируется множеством частиц [3]. Каждая частица ассоциирована с плотностью, скоростью, давлением среды в точке, в которой она расположена в текущий момент времени. Для восстановления непрерывных полей физических величин между частицами производится интерполирование на основе выкалывающего свойства дельта-функции:

$$A(\vec{x}) = \int_{\mathbb{R}^d} A(\vec{x}')\delta(\vec{x}' - \vec{x})\, d\Omega_{\vec{x}'}, \qquad \forall\, \vec{x} \in \mathbb{R}^d.$$

Здесь $A(\vec{x})$ — произвольное непрерывное распределение; $\delta(\vec{x})$ — d-мерная дельта-функция Дирака с присущими ей свойствами; d — размерность пространства.

В методе SPH вместо дельта-функции $\delta(\vec{x})$ используется некоторая гладкая функция $W(\vec{x}, h)$, называемая функцией ядра, которая зависит от пространственной координаты и параметра h, отлична от нуля лишь в малой окрестности начала координат (либо экспоненциально убывает на бесконечности) и удовлетворяет условию нормировки

$$\int_{\mathbb{R}^d} W(\vec{x}, h)\, d\Omega_{\vec{x}} = 1.$$

Параметр h, называемый длиной сглаживания, определяет размер окрестности центра функции ядра, вне которой она обращается в ноль либо становится пренебрежимо малой. В качестве функции ядра может выступать функция Гаусса, полиномы и сплайны разных порядков и др.

Таким образом, интерполирование неизвестной физической величины $A(\vec{x})$ по ее значениям в точках, в которых располагаются частицы, производится следующим образом (рис. 2):

$$A(\vec{x}) \approx \sum_j A(\vec{x}_j) W(\vec{x}_j - \vec{x}, h)\, \Delta V_j = \sum_j A(\vec{x}_j) W(\vec{x}_j - \vec{x}, h) \frac{m_j}{\rho_j}.$$

Здесь m_j — масса частицы с координатами $\vec{x}_j$, ρ_j — плотность среды в точке $\vec{x}_j$.

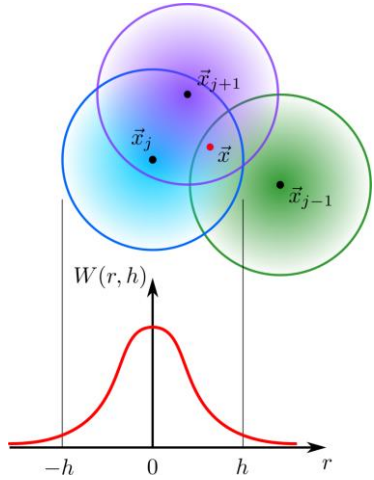


Рис. 2. Круги радиусами h и характерный профиль функции ядра
Fig.2. Circles of radius h and the characteristic profile of the core function

3.3. Метод конечных элементов с частицами

Метод конечных элементов с частицами — Particle Finite Element Method (PFEM) — это гибридный эйлерово-лагранжевый метод, объединивший в себе достоинства бессеточных методов и метода конечных элементов [4].

Первая версия этого метода появилась в 2001 году. В области, занятой жидкостью, вводится конечно-элементная сетка, узлы которой рассматриваются как лагранжевы частицы. Частицы движутся вместе со средой, что позволяет "автоматически" учесть конвективное слагаемое в уравнениях Навье — Стокса. Это приводит к необходимости перестроения (деформирования) сетки на каждом шаге расчета по времени. На вновь построенной сетке решаются уравнения гидродинамики (которые в отсутствие

конвективного члена становятся линейными) методом конечных элементов. Сетка может разделяться на несвязанные между собой подобласти (например, при моделировании брызг), в этом случае уравнения решаются независимо в каждой подобласти (рис. 3).

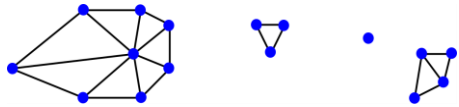


Рис. 3. Подобласти, занятые жидкой фазой, в методе PFEM
Fig.3. Subregions employed by liquid phase in the PFEM method

Такой подход позволяет упростить процесс решения уравнений (так как нет необходимости аппроксимации нелинейных слагаемых), но процедура перестроения сетки является нетривиальной и трудоемкой задачей: время перестроения сетки сопоставимо со временем решения уравнений и даже может превышать его. Также частицы в процессе решения могут сильно сближаться, что приводит к уменьшению размера ячеек сетки и, как следствие, шага расчета по времени.

Для ускорения счета в 2012 году была разработана модификация метода — PFEM-2 [5]. В ней конечно-элементная сетка является неподвижной и достаточно грубой, а частицы, которые вводятся для учета конвективного члена, перемещаются из одной ячейки сетки в другую вместе со средой (рис. 4). Движение частиц рассчитывается явным методом с малым шагом по времени, соответствующим числу Куранта CFL ~ 0.1. Учет остальных сил (вязкие силы, градиент давления, внешние массовые силы) осуществляется путем решения соответствующего линейного уравнения на сетке с большим числом Куранта (CFL ~ 1…10), что позволяет существенно снизить время выполнения расчета. Разработаны специальные процедуры проецирования характеристик частиц на узлы сетки и наоборот.

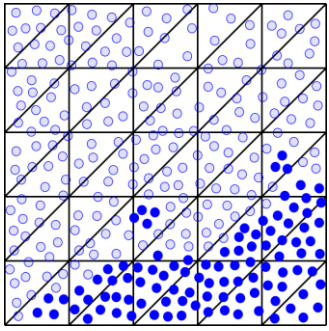


Рис. 4. Частицы в ячейках неподвижной сетки в методе PFEM-2
Fig. 4. Particles in cells of fixed grid in the method PFEM-2

Важной особенностью метода PFEM-2 является наличие "монолитных" решателей, не предполагающих использования процедур расщепления по физическим процессам.

3.4. Программные комплексы

Рассмотрены пять программных комплексов с открытым исходным кодом, в которых реализованы указанные выше численные методы:

- OpenFOAM (http://www.openfoam.org/) – классический метод Volume of Fluid, реализованный на C/C++ и успешно себя зарекомендовавший во многих практических приложениях;
- Gerris (http://gfs.sourceforge.net/wiki/) – метод Volume of Fluid с динамическими адаптивными сетками, реализованный на C/C++, позиционируемый как пакет, позволяющий максимально точно учитывать капиллярные явления;
- pySPH (https://pysph.readthedocs.org/) – метод SPH, реализованный с использованием языка программирования Python и технологии компиляции при исполнении Cython;
- DualSPHysics (http://www.dual.sphysics.org/) – метод SPH, реализация на C/C++ с возможностью проведения расчетов как на CPU, так и на графических ускорителях, поддерживающих технологию CUDA;
- Kratos (http://www.cimne.com/kratos/) – метод PFEM-2, реализация — на C/C++ для дискретизации и решения уравнений в частных производных и на языке Python для подготовки исходных данных и управления процессом интегрирования.

4. Тестовые задачи

Для сравнения программ было выбрано два расчетных случая. Первым тестовым примером является задача моделирования обрушения двумерного столба жидкости (воды), в которой влияние поверхностного натяжения пренебрежимо мало. Данная задача является одним из наиболее известных бенчмарк-тестов численных моделей механики жидких сред с разделом фаз, для которого в большом количестве доступны экспериментальные данные.

Во втором тестовом примере – задаче о моделировании падения капли в слой жидкости – поверхностное натяжение играет существенную роль. Во всех вышеперечисленных программных комплексах вопрос учёта поверхностного натяжения и связанных с ним эффектов стоит особняком. В некоторых пакетах не реализованы численные модели поверхностного натяжения либо их корректность вызывает сомнения, поэтому этот тестовый пример был рассмотрен только в двух из пяти представленных пакетов: OpenFOAM и Gerris.

4.1. Обрушение колонны жидкости

Рассмотрим двумерную задачу об обрушении колонны жидкости (воды) в прямоугольном резервуаре с гладкими стенками [6]. Геометрия расчетной области указана на рис. 5. В качестве характерного размера выбрана ширина колонны $L = 14.6 \cdot 10^{-2}$ м. Плотность жидкости ρ равна 999 кг/м^3, коэффициент динамической вязкости μ равен 10^{-3} Па·с, коэффициент поверхностного натяжения σ равен 0.0727 Н/м.

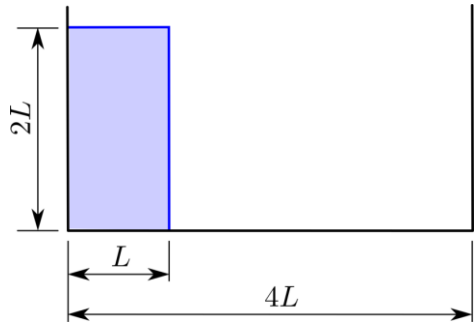


Рис. 5. Расчетная схема задачи об обрушении колонны жидкости
Fig. 5. Computational scheme of the problem of liquid column collapse

В [6] приведены экспериментальные данные для $T \leq 1$ с. Шаг по времени выбирался так, чтобы число Куранта не превышало значения 0.2. В расчетной области строится сетка либо задается распределение частиц так, чтобы в начальный момент времени в области, занятой жидкостью, было около 3000 частиц (либо узлов сетки).

Полученные результаты хорошо соотносятся с экспериментальными данными (рис. 6, 7). На рис. 6 представлены качественные результаты, позволяющие визуально сравнить полученные решения. На рис. 7 указаны соответствующие значения длины растекания жидкости в разные моменты времени для трех серий экспериментов и для полученных в пакетах численных решений.

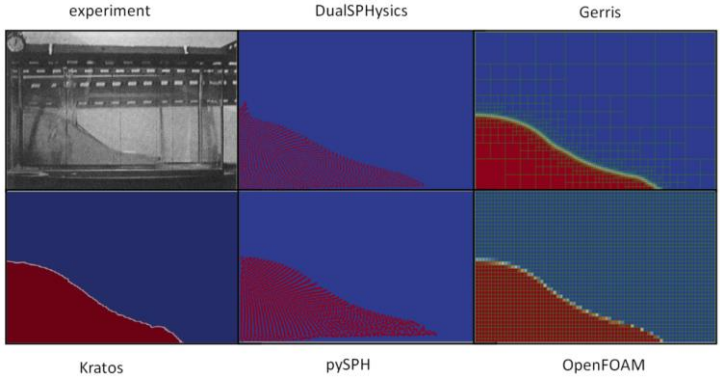
Характерные оценки времени счета в разных пакетах следующие.

- OpenFOAM: 1 мин;
- Gerris: 4 мин;
- pySPH: 9 мин;
- DualSPHysics: 3 мин;
- Kratos: 3 мин.

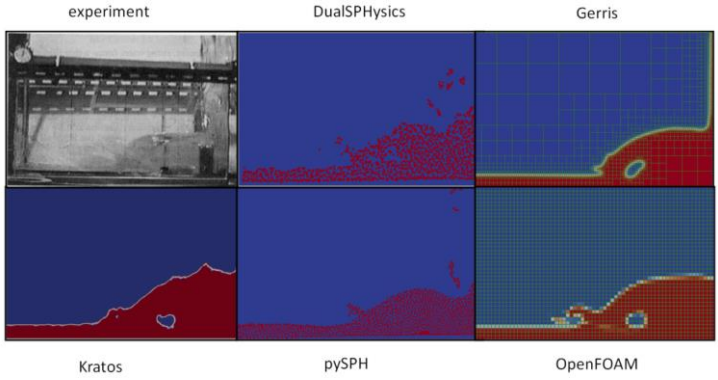
Пакет pySPH решил задачу за наибольшее время. Это может быть вызвано тем, что пакет написан на языке программирования высокого уровня Python, а

также тем, что код не подвергался оптимизации, так как пакет разрабатывался для учебных целей (наглядной демонстрации работы метода SPH).

Задача была решена в пакете Gerris в четыре раза медленнее, чем в OpenFOAM, несмотря на то, что в этих пакетах используется один и тот же численный метод (VOF). Это связано с тем, что на "маленькой" задаче время, затрачиваемое на адаптивное перестроение сетки в Gerris, существенно превышает время решения уравнений гидродинамики.



а



б

Рис. 6. Качественное сравнение результатов расчетов в различных пакетах с экспериментом для задачи обрушения водяной колонны: а – t=0.2 с; б – t=0.8 с
Fig. 6. A qualitative comparison of results of calculation in a variety of packages for the experiment with the problem of collapse of the water column: a – t=0.2 s; б – t=0.8 s

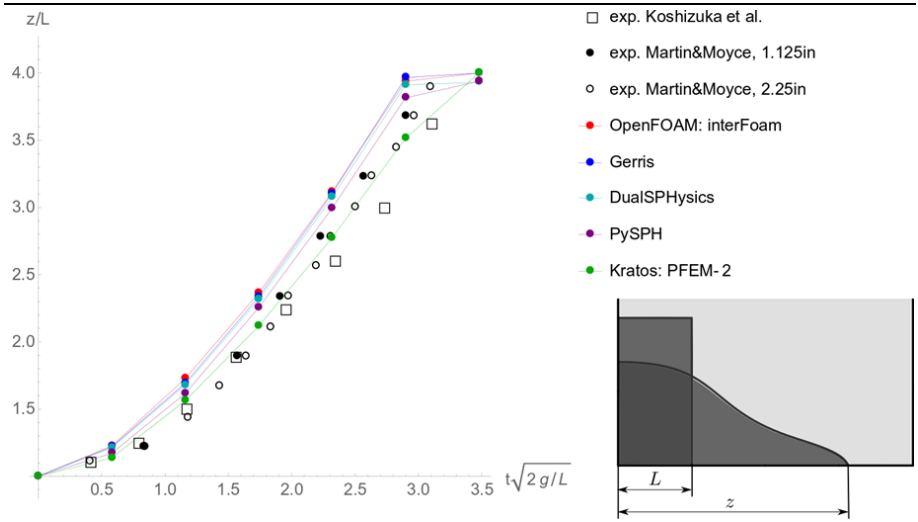


Рис. 7. Зависимость приведенной длины растекания жидкости от безразмерного времени для полученных численных решений и сравнение их с экспериментальными данными

Fig. 7. Dependence of the reduced length of liquid spreading on the dimensionless time for obtaining numerical solutions and their comparison with experimental data

4.2. Падение капли в слой жидкости

Капля диаметром $d = 2.67 \cdot 10^{-3}$ м из вязкой несжимаемой жидкости (смеси глицерина и воды) падает в слой такой же жидкости толщиной $H = 2d$ вертикально вниз (рис. 8). Ширина слоя жидкости считается бесконечно большой по сравнению с размерами падающей капли. Скорость капли U в момент соударения с жидкостью равна 2.56 м/с. Плотность жидкости ρ равна 1179 кг/м^3, коэффициент динамической вязкости μ равен $18.57 \cdot 10^{-3}$ Па·с, коэффициент поверхностного натяжения σ равен 0.0668 Н/м.

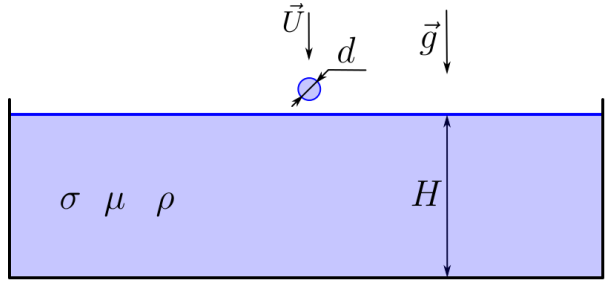


Рис. 8. Расчетная схема для задачи о падении капли в слой жидкости

Fig. 8. Computational scheme for the problem of falling drops into a layer of liquid

В численной постановке задача считается осесимметричной. Размеры расчетной области подбираются так, чтобы возмущения решения на краях области были пренебрежимо малы. В соответствии с [7] размеры расчетной области составляют 45×45 мм. Время расчетов T составляет 50 мс. Число Куранта, по которому производится автоматический выбор шага по времени, равно 0.2. Сетка, используемая в расчетной области, подбирается так, чтобы в радиусе капли содержалось около 30 ячеек.

В пакетах DualSPHysics и Kratos нет моделей поверхностного натяжения и возможности решения задач в осесимметричной постановке. Модели поверхностного натяжения, реализованные в пакете pySPH, требуют дополнительного анализа, так как результаты, показанные им, вызывают сомнения. В связи с этим задача о падении капли в жидкость была решена в пакетах OpenFOAM и Gerris.

На рис. 9, 10 показаны количественные результаты расчетов: измерены глубина кратера, появляющегося после падения капли в жидкость, и диаметр этого кратера, измеренный на половины высоты слоя жидкости. Наблюдается хорошее согласие между экспериментальными данными и расчетами.

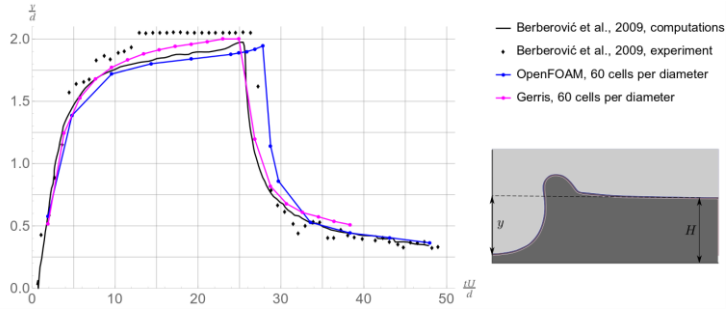


Рис. 9. Зависимость приведенной глубины кратера от безразмерного времени

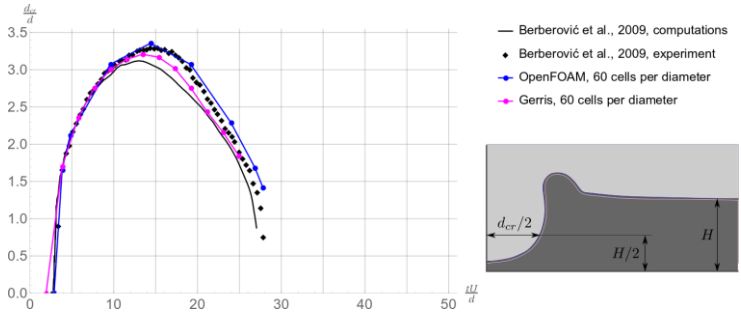


Рис. 10. Зависимость приведенного диаметра кратера, измеренного на половине высоты слоя жидкости, от безразмерного времени

Fig. 10. The dependence of the reduced diameter of a crater measured at half the height of the liquid layer on dimensionless time

Задача о падении капли в жидкость является достаточно "большой": сетка содержит около 250 000 ячеек (в OpenFOAM). Расчеты проводились в параллельном режиме, поэтому целесообразно указать характерные оценки времени решения в расчете на одно ядро.

Характерное время решения задачи в OpenFOAM составило около 12 часов, а в Gerris – 0.5 часа. Высокая эффективность пакета Gerris обеспечивается адаптивным перестроением сетки, которое позволяет существенно снизить ее размер (в 30–100 раз) и, соответственно, сильно снизить время расчетов.

5. Заключение

Рассмотрены три основных метода решения задач расчета течения вязкой несжимаемой жидкости со свободной поверхностью — Volume of Fluid, Smoothed Particle Hydrodynamics, Particle Finite Element Method, и разные их реализации в пяти программных комплексах с открытым исходным кодом. В этих пакетах программ были решены тестовые задачи. На основе полученных решений и при рассмотрении исходных кодов и документации к проектам сделаны следующие выводы о каждом из рассмотренных программных комплексов.

- OpenFOAM: универсальный пакет — реализованы математические модели многих физических эффектов; хорошо масштабируется.
- Gerris: узкоспециализированный пакет для решения задач расчета течений жидкости со свободной поверхностью; на "больших" задачах работает существенно быстрее, чем OpenFOAM.
- DualSPHysics: инженерный пакет для исследования прибрежной инфраструктуры; не реализован учет поверхностного натяжения, отсутствует возможность решения осесимметричных задач; есть возможность использования CUDA.
- pySPH: позволяет решать широкий класс задач; прост в использовании; имеются несколько моделей поверхностного натяжения; производительность — низкая.
- Kratos (PFEM-2): возможность решения ограниченного класса задач, в том числе сопряженных; наличие «монолитных» решателей; отсутствует возможность решения осесимметричных задач, нет учета поверхностного натяжения; высокая производительность.

Список литературы

[1]. J.U. Brackbill, D.B. Kothe, C. Zemach. A Continuum Method for Mode. Journal of Computation Physics, 100(2), 1992. P. 335–354. DOI: 10.1016/0021-9991(92)90240-Y

[2]. J.H. Ferziger, M. Perić. Computational Methods for Fluid Dynamics. Berlin: Springer-Verlag, 2002. 426 p. DOI:10.1007/978-3-642-56026-2

[3]. Shaofan Li, Wing Kam Liu. Meshfree and Particle Methods. Berlin: Springer-Verlag, 2004. 502 p. DOI:10.1007/978-3-540-71471-2

[4]. S.R. Idelsohn, E. Oñate, F. Del Pin. The particle finite element method: a powerful tool to solve incompressible flows with free-surfaces and breaking waves. International Journal for Numerical Methods in Engineering, 61(7), 2004. P. 964–989. DOI:10.1002/nme.1096

[5]. J.M. Gimenez, N.M. Nigro, S.R. Idelsohn. Evaluating the performance of the particle finite element method in parallel architectures. Comp. Part. Mech., 1(1), 2014. P. 103–116. DOI: 10.1007/s40571-014-0009-4

[6]. S. Koshizuka, Y. Oka. Moving-Particle Semi-Implicit Method for Fragmentation of Incompressible Fluid. Nuclear Science and Engineering, 123, 1996. P. 421–434.

[7]. E. Berberović, N.P. van Hinsberg, S. Jakirlić, I.V. Roisman, C. Tropea. Drop impact onto a liquid layer of finite thickness: Dynamics of the cavity evolution. Physical Review, 79, 2009. P. 1–15. DOI: 10.1103/PhysRevE.79.036306

Open-source software for modelling of free surface flows

E. Davydova <alenka-davidova@rambler.ru>
V. Korchagova <ko_viktoria@inbox.ru>
BMSTU, ul. Baumanskaya 2-ya, 5/1, Moscow, 105005, Russian Federation

Abstract. Problems of free-surface flow of viscous incompressible fluid are very useful in different practical cases. There are many specifies and limitations in these problems which are critically important for correct solving. The main goal is the review of existing numerical methods which can apply for modeling of free-surface flows and open-source programs where these methods are realized. Three methods for solving problems of free-surface flow were considered: Volume of Fluid, Smoothed Particle Hydrodynamics, Particle Finite Element Method v.2. They are realized in five open-source packages: OpenFOAM, Gerris, pySPH, DualSPHysics, Kratos. These packages were compared by modeling of two chosen cases: breaking of a dam and droplet impact to the liquid layer. Results of computations were compared with experimental results. There are good coincidence between them. The best results were obtained in OpenFOAM and Gerris. All main tools for modeling of free-surface flow are realized in these packages – the possibility of computations in 2D, 3D and axisymmetric model setup and also correct modeling of surface tension. Gerris can significantly accelerate computations in "big cases" due to dynamically adaptive remeshing. Further, DualSPHysics is the package for modeling of problems of coastal infrastructure where the most number of cases is 3D and the surface tension effect is negligible. The package pySPH was designed for clear demonstration of SPH working. The pySPH source code is on the Python language and not optimized. Kratos is the new package, which is in development now, therefore some tools are not developed in this moment.

Keywords: hydrodynamics; variable flow region; free-surface flow; VOF; SPH; PFEM; open-source packages.

DOI: 10.15514/ISPRAS-2016-28(1)-14

For citation: Davydova E., Korchagova V. Open-source software for modelling of free surface flows. Trudy ISP RAN /Proc. ISP RAS, 2014, vol. 28, issue 1, pp. 243-258 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-14

References

[1]. J.U. Brackbill, D.B. Kothe, C. Zemach. A Continuum Method for Mode. Journal of Computation Physics, 100(2), 1992. P. 335–354. DOI: 10.1016/0021-9991(92)90240-Y

[2]. J.H. Ferziger, M. Perić. Computational Methods for Fluid Dynamics. Berlin: Springer-Verlag, 2002. 426 p. DOI:10.1007/978-3-642-56026-2

[3]. Shaofan Li, Wing Kam Liu. Meshfree and Particle Methods. Berlin: Springer-Verlag, 2004. 502 p. DOI:10.1007/978-3-540-71471-2

[4]. S.R. Idelsohn, E. Oñate, F. Del Pin. The particle finite element method: a powerful tool to solve incompressible flows with free-surfaces and breaking waves. International Journal for Numerical Methods in Engineering, 61(7), 2004. P. 964–989. DOI:10.1002/nme.1096

[5]. J.M. Gimenez, N.M. Nigro, S.R. Idelsohn. Evaluating the performance of the particle finite element method in parallel architectures. Comp. Part. Mech., 1(1), 2014. P. 103–116. DOI: 10.1007/s40571-014-0009-4

[6]. S. Koshizuka, Y. Oka. Moving-Particle Semi-Implicit Method for Fragmentation of Incompressible Fluid. Nuclear Science and Engineering, 123, 1996. P. 421–434.

[7]. E. Berberović, N.P. van Hinsberg, S. Jakirlić, I.V. Roisman, C. Tropea. Drop impact onto a liquid layer of finite thickness: Dynamics of the cavity evolution. Physical Review, 79, 2009. P. 1–15. DOI: 10.1103/PhysRevE.79.036306

Об оценках вычислительной сложности и погрешности быстрого алгоритма в методе вихревых элементов

К.С. Кузьмина <kuz-ksen-serg@yandex.ru>
И.К. Марчевский <iliamarchevsky@mail.ru >
МГТУ им. Н.Э. Баумана, 105005,
Россия, г. Москва,ул. 2-я Бауманская, д. 5

Аннотация. Основная вычислительная сложность при использовании вихревых методов сосредоточена в вычислении конвективных и диффузионных скоростей всех вихревых элементов**.** Рассматривается один из эффективных способов ускорения вычислений в методе вихревых элементов — алгоритм типа Барнса – Хата. Метод основан на построении иерархической структуры областей (дерева). При вычислении конвективных скоростей вихревых элементов данный метод позволяет учитывать взаимное влияние вихревых элементов, находящихся далеко друг от друга, приближенно по формуле, полученной с помощью разложения выражения для конвективной скорости в ряд Тейлора. Влияние вихревых элементов, находящихся в ближней зоне, рассчитывается напрямую по закону Био — Савара. При реализации данного алгоритма возникают параметры, влияющие на вычислительную трудоемкость и точность решения задачи: $\boldsymbol{k}$ — количество уровней дерева и $\boldsymbol{\theta}$ — параметр дальности ячеек. Влияние вихревых элементов на диффузионные скорости друг друга экспоненциально затухает с увеличением расстояния между ними. Поэтому для вычисления диффузионных скоростей также построен алгоритм, позволяющий с помощью использования структуры дерева находить вихревые элементы, находящиеся в ближней зоне и вычислять влияние только от них. На основе решения модельных задач получены оценки вычислительной сложности алгоритмов вычисления конвективных и диффузионных скоростей, которые зависят от параметров алгоритма и количества вихревых элементов. Также получены оценки погрешности вычисления конвективных и диффузионных скоростей в зависимости от параметров алгоритма. На практике эти оценки позволяют выбирать оптимальные значения параметров алгоритма и добиваться максимального ускорения вычислений при заданном уровне допустимой погрешности расчета.

Ключевые слова: метод вихревых элементов, закон Био – Савара, вязкая жидкость, диффузионная скорость, алгоритм Барнса – Хата, быстрый мультипольный метод, вычислительная сложность, оценка погрешности.

DOI: 10.15514/ISPRAS-2016-28(1)-15

Для цитирования: Кузьмина К.С., Марчевский И.К. Об оценках вычислительной сложности и погрешности быстрого алгоритма в методе вихревых элементов. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 259-274. DOI: 10.15514/ISPRAS-2016-28(1)-15

1. Введение

Метод вихревых элементов — бессеточный лагранжев метод вычислительной гидродинамики, позволяющий моделировать течения идеальной или вязкой несжимаемой среды. Этот метод весьма эффективен при моделировании внешнего обтекания профилей, когда область течения является неограниченной. Основными достоинствами вихревых методов при решении таких задач являются низкая численная диссипация и точное выполнение граничного условия на бесконечности, в то время как при использовании сеточных методов для моделирования внешнего обтекания необходимо ограничивать расчетную область и обеспечивать удовлетворение некоторых «искусственных» граничных условий. Кроме того, вихревые методы позволяют моделировать не только течения вокруг неподвижных жестких тел, но также и гидроупругие колебания тел, сохраняя при этом примерно ту же вычислительную сложность.

Вихревые методы достаточно популярны в различных инженерных приложениях. Они позволяют вычислять как параметры течения, так и аэрогидродинамические силы, действующие на тело, с достаточной точностью, при этом их вычислительная сложность может быть значительно ниже по сравнению с сеточными методами, особенно при использовании в вихревых методах приближенных быстрых алгоритмов.

Целью данной работы является построение оценки вычислительной сложности быстрой реализации алгоритма вихревых методов и исследование зависимости погрешности приближенного решения от времени выполнения расчета.

2. Краткое описание вихревых методов

Существует несколько модификаций вихревых методов для моделирования двумерных и пространственных течений, иногда существенно отличающихся друг от друга как в части реализации отдельных операций и алгоритмов, так и в смысле используемых методов и подходов. Однако их общей особенностью является то, что первичной расчетной величиной является завихренность, а ее распределение в области течения моделируется большим количеством отдельных вихревых элементов. Каждый вихревой элемент генерирует «элементарное» поле скоростей во всей области течения, а общее поле скоростей является суперпозицией этих «элементарных» полей и выражается с помощью закона Био – Савара:

$$\vec{V}(\vec{r},t) = \vec{V}_\infty + \frac{1}{2(d-1)\pi}\int\limits_S \frac{\vec{\Omega}(\vec{\xi},t)\times(\vec{r}-\vec{\xi})}{\left|\vec{r}-\vec{\xi}\right|^d} dS_\xi. \qquad (2.1)$$

Здесь S — область с ненулевой завихренностью; d — размерность пространства в задаче; $\vec{\Omega}(\vec{\xi},t) = \nabla\times\vec{V}(\vec{\xi},t)$ — вектор завихренности в точке с радиус-вектором $\vec{\xi}$.

Способ аппроксимации распределения завихренности определяется выбором модели вихревого элемента. В двумерных задачах обычно используется модель круглого вихря Рэнкина; для пространственных задач известно несколько моделей вихревых элементов, каждая из которых имеет свои достоинства и недостатки. Далее будем рассматривать двумерный случай, но основные идеи после некоторой адаптации могут быть применены и к решению пространственных задач с помощью вихревых методов.

Течение вязкой несжимаемой жидкости описывается с помощью уравнений Навье – Стокса, которые могут быть записаны в форме Гельмгольца:

$$\frac{\partial\vec{\Omega}}{\partial t} + (\vec{V}\cdot\nabla)\vec{\Omega} + \nu\Delta\vec{\Omega} = 0,$$

где ν — коэффициент кинематической вязкости.

Наиболее простым способом учета конвективного слагаемого $(\vec{V}\cdot\nabla)\vec{\Omega}$ является перемещение всех вихревых элементов вдоль линий тока, при этом их интенсивности (циркуляции) остаются постоянными; такой алгоритм будет соответствовать моделированию течения идеальной жидкости.

Для учета слагаемого, отвечающего за влияние вязкости, известно несколько подходов: метод случайных блужданий [1], метод обмена интенсивностями между вихревыми элементами [2], метод диффузионной скорости [3,4,5]. В соответствии с подходом, предполагающим введение диффузионной скорости, уравнение Навье – Стокса в двумерном случае может быть записано в следующей форме:

$$\frac{\partial\vec{\Omega}}{\partial t} = \nabla\times\left((\vec{V}+\vec{W})\times\vec{\Omega}\right), \quad \vec{\Omega} = \Omega\vec{k}.$$

Здесь $\vec{W} = \nu\frac{\nabla\Omega}{\Omega}$ — диффузионная скорость. Таким образом, при моделировании вязкого течения методом вязких вихревых доменов (ВВД) [4] необходимо численное решение следующей системы на каждом временном шаге:

$$\begin{cases}\dfrac{d\Gamma_i}{dt} = 0,\\ \dfrac{d\vec{r}_i}{dt} = \vec{V}(\vec{r}_i) + \vec{W}(\vec{r}_i),\end{cases} \qquad i = 1,\dots,N.$$

Первое уравнение означает, что интенсивности вихревых элементов Γ_i остаются постоянными (как в случае идеальной жидкости), а их положения меняются таким образом, что они движутся вдоль линий тока поля скоростей $\vec{V} + \vec{W}$.

3. Вычислительная сложность метода вихревых элементов

Основная вычислительная сложность вихревых методов сосредоточена в вычислении конвективных и диффузионных скоростей всех вихревых элементов. Для вихревых элементов Рэнкина согласно закону Био –Савара (2.1) получаем следующее выражение для конвективной скорости

$$\vec{V}(\vec{r},t) = \vec{V}_\infty + \sum_{i=1}^{N} \frac{\Gamma_i}{2\pi} \frac{\vec{k} \times (\vec{r} - \vec{r}_i)}{\max\{|\vec{r} - \vec{r}_i|^2, \varepsilon^2\}}. \qquad (3.1)$$

Здесь ε — радиус вихревого элемента, который полагается постоянным; $\vec{r}_i$ — положение -го вихревого элемента; Γ_i — его интенсивность; N — количество вихревых элементов. Ясно, что вычислительная сложность процедуры вычисления конвективных скоростей пропорциональна N^2. Если учитывать только операции умножения и деления, она равна $6N^2$. Можно добиться уменьшения сложности, если учесть, что $\vec{V}_{ij}$ и $\vec{V}_{ji}$ имеют противоположные знаки и отличаются только множителями Г, однако сложность в любом случае будет превышать $3N^2$ ($\vec{V}_{ij}$ — вклад -го вихревого элемента в конвективную скорость i-го вихревого элемента).

Прямое вычисление скоростей (с использованием формулы (3.1)) возможно, когда количество вихревых элементов не превышает нескольких десятков тысяч, в противном случае время выполнения расчета становится недопустимо большим. Обычно для моделирования нестационарного течения, особенно в гидроупругих задачах, необходимо выполнять до нескольких тысяч шагов по времени; для повышения точности моделирования течения необходимо увеличивать количество вихревых элементов, что в свою очередь приведет к увеличению времени расчета одного шага по времени.

В данной работе не затрагивается проблема моделирования обтекания профиля, т.е. не рассматривается учет граничных условий прилипания (непротекания в случае идеальной жидкости) на поверхности профиля. Отметим лишь, что для учета влияния профиля на течение в вихревых методах профиль заменяется вихревым слоем (и слоем источников в случае движущегося или деформируемого профиля), интенсивность которого может быть определена из решения некоторого интегрального уравнения [6,7,8]. В общем случае вихревой слой является свободным, его также можно моделировать с помощью вихревых элементов, которые становятся частью вихревого следа.

Для снижения временных затрат на выполнение вычислений используются разные методы ускорения расчетов. Один из наиболее популярных подходов — использование алгоритма типа Барнса – Хата [9], изначально разработанного для приближенного быстрого решения гравитационной задачи N тел. К вихревым методам этот алгоритм был адаптирован в работе [10]. Этот алгоритм основан на построении иерархической структуры областей — дерева. Напрямую по закону Био – Савара (3.1) вихревое влияние рассчитывается только от вихревых элементов, которые находятся в достаточно близких к рассматриваемой ячейках, влияние от вихревых элементов, находящихся далеко от рассматриваемой ячейки, рассчитывается приближенно по упрощенным формулам.

Используя данный подход, можно добиться значительного снижения вычислительной сложности алгоритма: при правильном выборе количества уровней дерева сложность будет пропорциональна $N \log N$. Это позволяет за разумное время решать задачи с большим количеством (десятками или даже сотнями тысяч) вихревых элементов.

При решении практических задач недостаточно знать только порядок вычислительной сложности, необходима более подробная оценка, которая может помочь при выборе оптимальных параметров алгоритма. Известные оценки [11, 12] позволяют выбирать оптимальное количество уровней дерева, но они имеют невысокую точность. В [13] авторами получена более точная оценка количества операций:

$$Q = \frac{24\,N^2}{2^k}\left(\frac{4}{\theta}\right)\left(1-\alpha\frac{\left(\sqrt{2}\right)^k-1}{\sqrt{N}}\right)^2\left(1-\frac{4}{\theta\left(\sqrt{2}\right)^k}\left(1-\alpha\frac{\left(\sqrt{2}\right)^k-1}{\sqrt{N}}\right)\right)+$$

$$+\frac{896\cdot 2^k\cdot\beta}{\theta^2}\left(4\left(\frac{1}{4+\theta}+\frac{1}{4-\left(\sqrt{2}\right)^k\theta}\right)+\ln\left(\frac{\left(\sqrt{2}\right)^k-4}{4+\theta}\right)\right)+4N. \qquad (3.2)$$

Здесь N — количество вихревых элементов в области течения, k — количество уровней дерева, θ — параметр точности. Значения коэффициентов α и β определяются эмпирически, они зависят от решаемой задачи. Значение α
$(0 < \alpha < 1)$ значительно зависит от распределения вихревых элементов в вихревом следе, а величина β $(\beta > 0)$ зависит от формы вихревого следа.

При малых значениях параметра θ $(0 < \theta \ll 1)$ быстрый метод имеет высокую точность, но в этом случае он имеет также и высокую вычислительную сложность; большие значения θ позволяют понизить сложность вычислений, но при этом повышается погрешность решения.

Известны работы, в которых получены и математически доказаны оценки погрешности алгоритма Барнса – Хата [14,15], но эти оценки имеют асимптотический характер и содержат некоторые неизвестные параметры. Эти

оценки могут быть полезны для «теоретической» оценки, но на практике их применение затруднительно. Целью данной работы является построение приближенных оценок для вычислительной сложности быстрого алгоритма, а также для погрешности вычисления скоростей.

4. Модельная задача

Рассмотрим тестовую задачу о моделировании течения вязкой жидкости, которая соответствует известному явлению диффузии круглого вихря в неограниченной области (вихря Ламба). Пусть в начальный момент времени $t = 0$ в вязкую несжимаемую жидкость введена бесконечная вихревая нить с циркуляцией Γ. Точное решение данной задачи для распределения завихренности в момент времени t имеет вид

$$\Omega(r,t) = \frac{\Gamma}{4\pi\nu t}\exp\left(-\frac{r^2}{4\nu t}\right), \qquad (4.1)$$

где r — расстояние до центра вихря. Найдем суммарную завихренность внутри окружности радиуса R:

$$\Gamma_R(t) = \int_0^{2\pi} d\varphi \int_0^R \Omega(r,t)\, r\, dr = \Gamma\left(1 - \exp\left(-\frac{R^2}{4\nu t}\right)\right).$$

Будем считать, что $\Gamma = 1$, коэффициент кинематической вязкости среды $\nu = (2000\pi)^{-1} \approx 0{,}00016$, время начала моделирования распределения завихренности $t_0 = 2000\pi \approx 6283$. Тогда более 99,8 % всей завихренности содержится в окружности радиуса $R = 5$, остальной пренебрежем. Количество вихревых элементов, которое было использовано для моделирования распределения завихренности внутри этого круга, взято из интервала $N = 60\,000 \ldots 300\,000$, при этом их пространственное распределение близко к равномерному. Пример такого распределения (с небольшим количеством вихревых элементов $N \approx 1000$) показано на рис. 1. Интенсивности вихревых элементов вычисляются аналитически при помощи интегрирования точного решения (4.1).

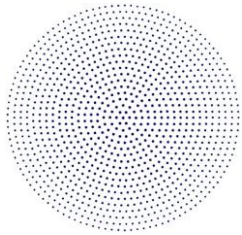


Рис.1. Распределение вихревых элементов в круглом вихре ($N \approx 1000$)
Fig. 1. The distribution of the vortex elements in a circular vortex

Для построения дерева для алгоритма типа Барнса – Хата был использован стандартный метод [10], ячейки дерева для разных уровней дерева показаны на рис. 2.

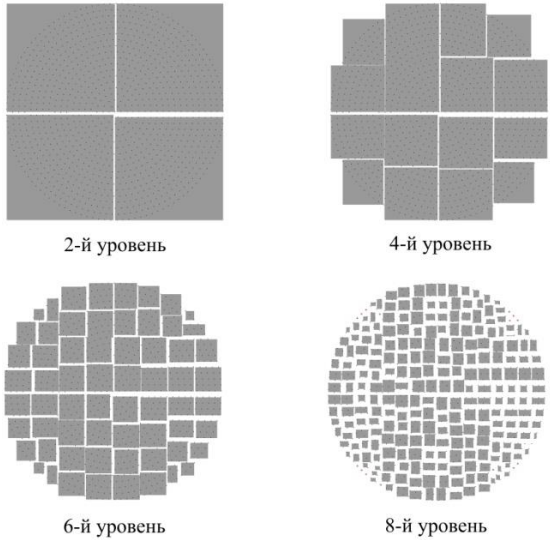


Рис. 2. Ячейки для разных уровней дерева
Fig.2. Cells for different levels of the tree

5. Оценка вычислительной сложности

Для построения эффективного алгоритма необходима подробная оценка вычислительной сложности всех его частей. Отметим, что построение дерева является очень «легкой» процедурой (даже при количестве вихревых элементов порядка сотен тысяч) по сравнению с вычислением конвективных и диффузионных скоростей всех вихревых элементов.

5.1. Вычисление конвективных скоростей

Оценка вычислительной сложности алгоритма расчета конвективных скоростей состоит из двух частей: первая соответствует прямому вычислению скоростей с помощью закона Био – Савара от вихревых элементов, ячейки которых расположены достаточно близко друг к другу, согласно [13] эта оценка имеет следующую форму:

$$Q_{BS} = \frac{24N^2}{2^k}\left(\frac{4}{\theta}\right)^2\left(1-\alpha\frac{\left(\sqrt{2}\right)^k-1}{\sqrt{N}}\right)^2 \cdot$$

$$\cdot\left(1-\frac{4}{\theta\left(\sqrt{2}\right)^{k}}\left(1-\alpha\frac{\left(\sqrt{2}\right)^{k}-1}{\sqrt{N}}\right)\right)+4N. \qquad (5.1)$$

Вторая часть оценки касается приближенного вычисления скоростей от ячеек, которые находятся достаточно далеко от рассматриваемой ячейки:

$$Q_{Far}=\frac{896\cdot 2^{k}\cdot\beta}{\theta^{2}}\left(4\left(\frac{1}{4+\theta}+\frac{1}{4-\left(\sqrt{2}\right)^{k}\theta}\right)+\ln\left(\frac{\left(\sqrt{2}\right)^{k}-4}{4+\theta}\right)\right). \qquad (5.2)$$

Критерий дальности ячеек имеет следующую форму:

$$\theta\cdot\delta>h+h_0,$$

где δ — октаэдрическая норма ($\|\cdot\|_1$) вектора, соединяющего центры «влияющей» и «рассматриваемой» ячеек, h и h_0 — суммы длин сторон этих ячеек.

Для того чтобы получить оценку коэффициента α в формулах (3.2) и (5.1), с помощью специальной процедуры в алгоритме был произведен точный подсчет количества операций, осуществляемых в расчетах. Оценка для Q_{BS} (5.1) содержит только параметр α, а параметр β не входит в это выражение, что позволяет получить хорошую оценку для параметра α.

Результаты подсчета количества операций Q_{BS} с помощью численного эксперимента (реальное количество операций) и с помощью аналитической оценки (5.1) для $150\,000$ вихревых элементов при значении параметра $\alpha=0{,}8$ показаны на рис. 3.

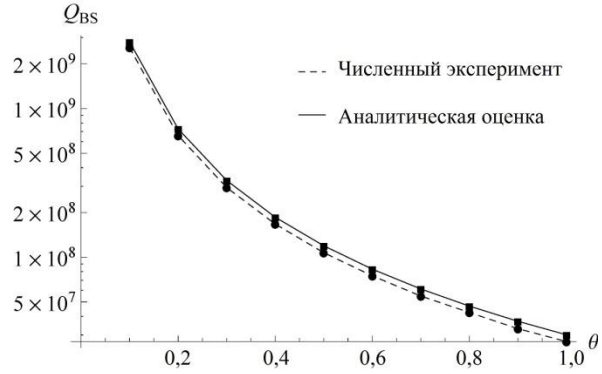


Рис. 3. Количество операций Q_{BS} ($N=150\,000,\ \alpha=0{,}8$)

Fig. 3. The number of operations Q_{RS} ($N = 150000,\ \alpha = 0{,}8$)

Для получения оптимального значения параметра α были проведены численные эксперименты с разным количеством вихревых элементов ($N = 60\,000 \ldots 300\,000$), моделирующих исходный вихрь, для разных значений параметра $\theta = (0{,}1 \ldots 1{,}0)$. Количество уровней дерева при этом выбиралось исходя из оценок, приведенных в работах [12, 13]. Для каждого значения N было вычислено значение коэффициента α, минимизирующее сумму квадратов относительных погрешностей:

$$\sum_{i=1}^{10}\left(\frac{Q_{BS}(\theta_i, N, \alpha) - Q^*_{BS}(\theta_i, N)}{Q^*_{BS}(\theta_i, N)}\right)^2 \to \min,$$

где $\theta_i = 0{,}1i$; $Q^*_{BS}(\theta_i, N)$ — количество операций, полученное из численного эксперимента для N вихревых элементов и $\theta = \theta_i$.

Вычисленные оптимальные значения параметра α приведены в табл. 1. Полученные значения близки друг к другу, поэтому на практике можно использовать среднее значение $\alpha = 0{,}844$.

Похожая процедура подсчета количества операций и минимизации погрешностей была проведена для вычисления оптимального значения параметра β в оценке (5.2). Полученные оптимальные значения параметра β для разных N приведены в табл. 1. Снова можно заметить, что эти величины близки, поэтому предлагается, так же как и в случае с α, использовать среднее значение $\beta = 0{,}561$.

Табл. 1. Оптимальные значения α и β для разных значений N (k — количество уровней дерева)
Table 1. Optimal values α and β for different N (k – the number of levels of the tree)

N	60 000	90 000	120 000	150 000	180 000	210 000	240 000	270 000	300 000
k	14	15	15	16	16	16	16	16	17
α	0,885	0,793	0,826	0,837	0,863	0,876	0,871	0,876	0,776
β	0,521	0,558	0,587	0,539	0,554	0,566	0,582	0,587	0,582

Таким образом, получена оценка вычислительной сложности алгоритма вычисления конвективных скоростей.

5.2. Вычисление диффузионных скоростей

Вычисление диффузионных скоростей производится с помощью похожего подхода: во внимание принимается влияние только от тех вихревых элементов, которые находятся в ячейках нижнего уровня, расположенных достаточно близко к рассматриваемой ячейке и удовлетворяющих условию:

$$\theta_{dif} \cdot \left(\delta - 0.5(h + h_0)\right) < \varepsilon^*,$$

где θ_{dif} — параметр дальности ячеек для диффузионных скоростей; чем меньше этот параметр, тем точнее приближенное решение; δ, h и h_0 — те же величины, что и в (5.3); ε^* — характерное расстояние между вихревыми элементами. Расчетная формула для вычисления диффузионной скорости приведена, к примеру, в [4].

Для того, чтобы получить выражение для оценки вычислительной сложности алгоритма расчета диффузионных скоростей, можно использовать те же рассуждения, что и для конвективных скоростей [13]. Опуская промежуточные вычисления, отметим, что вычислительная сложность для алгоритма расчета диффузионных скоростей пропорциональна сложности Q алгоритма для конвективных скоростей (3.2), но при этом θ нужно заменить на θ_{dif}:

$$Q_{dif} = Q|_{\theta=\theta_{dif}} \cdot N \cdot \theta_{dif} \cdot \frac{\gamma}{2^k}.$$

Здесь N и k, как и ранее, количество вихревых элементов и уровней дерева; γ — коэффициент, который может быть оценен численно (табл. 2).

Табл. 2. Оптимальные значения коэффициента γ для разных значений N (k — количество уровней дерева)

Table 2. Optimal values of coefficient γ for different N (k – the number of levels of the tree)

N	60 000	90 000	120 000	150 000	180 000	210 000	240 000	270 000	300 000
k	14	15	15	16	16	16	16	16	17
γ	0,0775	0,0775	0,0731	0,0712	0,0694	0,0689	0,0686	0,0679	0,0686

Видно, что значения коэффициента γ при различных N близки, поэтому в расчетах будем использовать среднее значение $\gamma \approx 0{,}7$.

6. Погрешности вычисления скоростей с помощью быстрых алгоритмов

Основной целью применения алгоритма Барнса – Хата является ускорение вычислений. Полученные формулы (5.1), (5.2) и (5.4) позволяют оценить, сколько времени требуется для моделирования течения, но кроме этого немаловажным вопросом является оценка величины погрешности быстрого метода. Далее будем рассматривать относительную погрешность вычисления конвективных и диффузионных скоростей

$$\varepsilon = \frac{\max\limits_{i=1,\dots,N} \left|\vec{V}_i - \vec{V}_i^*\right|}{\left|\vec{V}_{conv}^*\right|},$$

где $\vec{V}_i$ — конвективная либо диффузионная скорость -го вихревого элемента, вычисленная быстрым методом; $\vec{V}_i^*$ — скорость того же вихревого элемента, вычисленная напрямую по формуле (3.1) для конвективных скоростей и по формуле [4] для диффузионных скоростей; $\left|\vec{V}_{conv}^*\right|$ — максимальное значение конвективных скоростей всех вихревых элементов. Для модельной задачи, которая рассматривается в данной работе, $\left|\vec{V}_{conv}^*\right| \approx 0{,}05$ и это значение практически не зависит от количества вихревых элементов N.

6.1. Погрешность вычисления конвективных скоростей

Погрешность вычисления конвективной скорости значительно зависит от значения параметра θ; бóльшие значения параметра θ соответствуют бóльшим погрешностям.. Для того, чтобы оценить, как параметр θ влияет на погрешность вычисления конвективной скорости, был выполнен ряд вычислительных экспериментов. Были произведены расчеты для $N = 60\,000 \ldots 300\,000$ с использованием быстрого метода с разными параметрами $\theta = 0.1 \ldots 1.0$. Полученные результаты изображены на рис. 4; тонкие линии соответствуют относительным погрешностям вычисления конвективной скорости для экспериментов с разными N, жирная линия соответствует кривой, которая является их оценкой сверху.

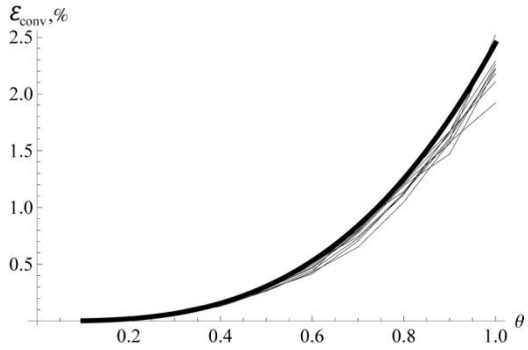


Рис. 4. Относительная погрешность для конвективных скоростей (тонкие линии соответствуют $N = 60\,000 \ldots 300\,000$, жирная линия — оценка сверху)
Fig 4. The relative error for the convective velocity (thin lines correspond to N = 60 000 ... 300 000, thick line – the upper bound)

Можно заметить, что для каждого значения N зависимость погрешности от θ хорошо аппроксимируется функцией $\varepsilon_{conv} = c \cdot \theta^3$, при этом для разных N значение коэффициента c варьируется от $0{,}020$ до $0{,}025$. Таким образом, мажоранта этих кривых

$$\varepsilon_{conv} = 0{,}025 \cdot \theta^3$$

соответствует достаточно точной оценке погрешности вычисления конвективных скоростей быстрым методом.

6.2. Погрешность вычисления диффузионных скоростей

Погрешность вычисления диффузионной скорости с помощью быстрого метода зависит не только от параметра θ_{dif}, но и от некоторых других параметров. Для каждого N относительная погрешность близка к квадратичной функции θ_{dif} со своим коэффициентом пропорциональности. На рис. 5 показаны соответствующие кривые для $N = 90\,000$ и $N = 300\,000$. Соответствующий коэффициент пропорциональности может быть найден с помощью метода наименьших квадратов.

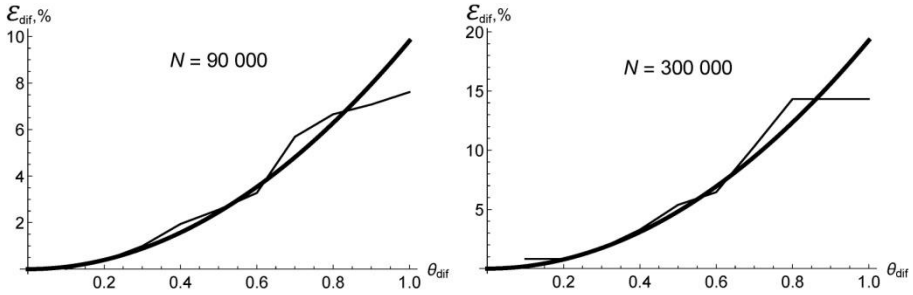


Рис. 5. Относительные погрешности для диффузионных скоростей (тонкие линии — численный эксперимент; жирные линии — квадратичные функции $\varepsilon = c_{dif}\,\theta_{dif}^2$)
Fig. 5. The relative error for the diffusion rates (thin line - numerical experiment; thick lines – quadratic functions $\varepsilon = c_{dif}\,\theta_{dif}^2$)

Для больших значений параметра θ_{dif} наблюдается значительное различие между квадратичной оценкой и результатами численных экспериментов, но с практической точки зрения это не имеет большого значения, поскольку погрешность в 10 – 20 % недопустима и необходимо производить расчеты при малых значениях параметра θ_{dif}, для которых оценка и результаты эксперимента хорошо согласуются.

Значения коэффициента c_{dif}, вычисленные для разных экспериментов ($N = 60\,000 \ldots 300\,000$), представлены в табл. 3.

Табл. 3. Значения коэффициента c_{dif} для разных значений N (k — количество уровней дерева)
Table 3. Values of the coefficient c_{dif} for different N (k – the number of levels of the tree)

N	60 000	90 000	120 000	150 000	180 000	210 000	240 000	270 000	300 000
k	14	15	15	16	16	16	16	16	17

c_{dif}	6,86	9,81	9,76	13,70	13,76	13,19	13,51	13,72	19,28

Из табл. 3 видно, что значение коэффициента c_{dif} зависит только от количества уровней дерева, т.е. $c_{dif} = c_{dif}(k)$, а для фиксированного значения k оно практически постоянно. Можно попробовать записать приближенную формулу для $c_{dif}(k)$ (используя метод наименьших квадратов или какой-либо другой способ), но она будет чисто эмпирической и сложно применимой на практике. Получение аналитической оценки этого явления является целью дальнейших исследований.

7. Заключение

Описана подробная оценка вычислительной сложности алгоритма типа Барнса – Хата для процедуры вычисления конвективных и диффузионных скоростей в методе вихревых элементов. Получена зависимость погрешности вычисления конвективных и диффузионных скоростей для модельной задачи. Если принять допустимой погрешность $\varepsilon = 0{,}2\ \%$, то можно выбрать следующие значения параметров дальности для алгоритма быстрого метода: $\theta = 0{,}4,\ \ \theta_{dif} = 0{,}1$. Посчитаем суммарную вычислительную сложность расчета конвективных и диффузионных скоростей быстрым методом Q_{tot} и сравним ее со сложностью прямого метода $Q^*_{tot} = 15N^2$ (коэффициент 15 получен с учетом того, что расчет одного парного влияния на конвективные скорости требует 6 операций умножения и деления, а на диффузионные — 9 операций). Величина ускорения $\delta = Q^*_{tot}/Q_{tot}$ показывает эффективность использования быстрого метода (табл. 4).

Табл. 4. Вычислительные сложности прямого и быстрого метода и ускорение
Table. 4. Computational complexity of direct and rapid methods and the acceleration

N	60 000	90 000	120 000	150 000	180 000	210 000	240 000	270 000	300 000
k	14	15	15	16	16	16	16	16	17
$\frac{Q_{tot}}{10^9}$	0,58	0,94	1,26	1,70	1,93	2,24	2,60	2,97	3,69
$\frac{Q^*_{tot}}{10^9}$	54,3	122,0	215,9	343,2	486,4	670,5	864,3	1097,4	1357,0
δ	93,6	129,8	171,3	201,9	252,0	299,3	332,4	369,5	367,8

Таким образом, при $N \approx 70\,000$ достигается 100 кратное ускорение вычислений по сравнению с прямым методом. А при расчете $N \approx 250\,000$ можно добиться 350 кратного ускорения.

Полученные оценки позволяют выбирать параметры для алгоритма быстрого метода таким образом, чтобы получить минимально возможную вычислительную сложность для требуемой величины погрешности.

Список литературы

[1]. A.J. Chorin, Numerical study of slightly viscous flow, J. Fluid. Mech., 57 (1973), pp. 785–796.
[2]. P. Degond, and S. Mas-Gallic, The weighted particle method for convection-diffusion equations. Part I: The case of an isotropic viscosity, Math. Comp., 53 (1989), pp. 485–507.
[3]. Ogami Y., Akamatsu T. Viscous flow simulation using the discrete vortex model — the diffusion velocity method // Computers and Fluids. 1991. V. 19, N 3/4. P. 433–441.
[4]. Дынникова Г.Я. Лагранжев подход к решению нестационарных уравнений Навье – Стокса // Доклады Академии наук. 2004. Т. 399, №1. С. 42–46.
[5]. S. Guvernyuk, and G. Dynnikova, Modeling the ow past an oscillating airfoil by the method of viscous vortex domains, Fluid Dynamics, 42 (2007), pp. 1–11.
[6]. I. K. Lifanov, and S. M. Belotserkovskii, Methods of Discrete Vortices. CRC Press, 1993.
[7]. S. N. Kempka, M. W. Glass, J. S. Peery, and J. H. Strickland, Accuracy Considerations for Implementing Velocity Boundary Conditions in Vorticity Formulations. SANDIA Report SAND96-0583, 1996.
[8]. K. S. Kuzmina, and I. K. Marchevsky The Modified Numerical Scheme for 2D Flow-Structure Interaction Simulation Using Meshless Vortex Element Method, in Proc. IV Int. Conference on Particle-based Methods — Fundamentals and Applications (PARTICLES-2015), Barcelona (2015), pp. 680–691.
[9]. J. Barnes, and P. Hut, A hierarchical O($N \log N$) force-calculation algorithm, Nature, 324 (1986), pp. 446–449.
[10]. Дынникова Г.Я. Использование быстрого метода решения «задачи N тел» при вихревом моделировании течений // Журнал вычислительной математики и математической физики. 2009. Т. 49, № 8. С. 1458–1465.
[11]. Гирча А.И. Быстрый алгоритм решения «задачи N тел» в контексте численного методя вязких вихревых доменов // Информационные технологии моделирования и управления. 2008. № 1. С. 47–52.
[12]. Морева В.С. Способы ускорения вычислений в методе вихревых элементов // Вестн. Моск. гос. техн. ун-та им. Н.Э. Баумана. Сер.: Естественные науки. Спец. выпуск «Прикладная математика». 2011. С. 83–95.
[13]. Кузьмина К.С., Марчевский И.К. Оценка трудоемкости быстрого метода расчета вихревого влияния в методе вихревых элементов // Наука и образование: электрон. науч.-техн. изд. 2013. № 10. С. 399–414.
[14]. A. Grama, V. Sarin, and A. Sameh, Improving Error Bounds for Multipole-Based Treecodes, SIAM J. Sci. Comp., 21 (2000), pp. 1790–1803.
[15]. J. K. Salmon, and M. S. Warren, Skeletons from the treecode closet, J. Comput. Phys., 111 (1994), pp. 136–155.

On the estimations of efficiency and error of fast algorithm in vortex element method

K.S. Kuzmina <kuz-ksen-serg@yandex.ru >
I.K. Marchevsky <iliamarchevsky@mail.ru >
Bauman Moscow State Technical University,
105005, Russia, Moscow, 2-nd Baumanskaya st., 5.

Abstract. The main computational complexity of vortex methods is concentrated in the calculation of the convective and diffusive velocities of all vortex elements. The analogue of the Barnes – Hut algorithm is considered as one of the most efficient ways to acceleration of the velocities computation in vortex element method. This method is based on the tree (hierarchical structure of regions) construction. When calculating the convective velocities this algorithm makes it possible to take into account the influence of vortex elements, which are located far enough from each other, approximately by using Taylor approximation of expression for convective velocity. The influence of vortex elements, which are close to each other is calculated directly using Biot —Savart law. There are two parameters of algorithm that affect the computational complexity of the algorithm and the problem solving accuracy: $\boldsymbol{k}$ is the maximum tree level, $\boldsymbol{\theta}$ is the parameter which determines the radius of a near-field zone. When calculating diffusive velocities the influence of the vortex elements to each other decays exponentially with increasing distance between them. Therefore, constructed algorithm of diffusive velocities calculation allows finding vortex elements from near-field zone using tree structure and calculating diffusive velocities using only vortex elements from this zone.

The estimations of computational complexity, which depends on algorithm parameters and number of vortex elements, are obtained for the algorithms for convective and diffusive velocities calculation. Also estimations for the errors of the vortex elements velocities computation, which depend on the algorithm parameters, are constructed. These estimates allow in practice to choice the optimal parameters of the whole algorithm and achieve the maximum possible acceleration of the computations for the given maximum error level.

Keywords. Vortex element method, Biot – Savart law, viscous flow, diffusive velocity, Barnes – Hut algorithm, fast multipole method, computational complexity, error estimation

DOI: 10.15514/ISPRAS-2016-28(1)-15

For citation: Kuzmina K.S., Marchevsky I.K. On the estimations of efficiency and error of fast algorithm in vortex element method. Trudy ISP RAN/Proc. ISP RAS, 2014, vol. 28, issue 1, pp. 259-274 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-15

References

[1]. A.J. Chorin, Numerical study of slightly viscous flow, J. Fluid. Mech., 57 (1973), pp. 785–796.

[2]. P. Degond, and S. Mas-Gallic, The weighted particle method for convection-diffusion equations. Part I: The case of an isotropic viscosity, Math. Comp., 53 (1989), pp. 485–507.

[3]. Y. Ogami, and T. Akamatsu, Viscous flow simulation using the discrete vortex model — the diffusion velocity method, Computers & Fluids, 19 (1991), pp. 433–441.
[4]. G. Ya. Dynnikova, Lagrange method for Navier – Stokes equations solving, Doklady Akademii Nauk, 399 (2004), pp. 42–46.
[5]. S. Guvernyuk, and G. Dynnikova, Modeling the flow past an oscillating airfoil by the method of viscous vortex domains, Fluid Dynamics, 42 (2007), pp. 1–11.
[6]. I. K. Lifanov, and S. M. Belotserkovskii, Methods of Discrete Vortices. CRC Press, 1993.
[7]. S. N. Kempka, M. W. Glass, J. S. Peery, and J. H. Strickland, Accuracy Considerations for Implementing Velocity Boundary Conditions in Vorticity Formulations. SANDIA Report SAND96-0583, 1996.
[8]. K. S. Kuzmina, and I. K. Marchevsky, The Modified Numerical Scheme for 2D Flow-Structure Interaction Simulation Using Meshless Vortex Element Method, in Proc. IV Int. Conference on Particle-based Methods — Fundamentals and Applications (PARTICLES-
2015), Barcelona (2015), pp. 680–691.
[9]. J. Barnes, and P. Hut, A hierarchical O($N \log N$) force-calculation algorithm, Nature, 324 (1986), pp. 446–449.
[10]. G. Ya. Dynnikova, Fast technique for solving the N-body problem in flow simulation by vortex methods, Computational Mathematics and Mathematical Physics, 49 (2009), pp. 1389–1396.
[11]. A. I. Gircha, Fast Algorithm for N-body Problem Solving with Regard to Numerical Method of Viscous Vortex Domains, Informatial technologies in Simulating and Control, 1 (2008), pp. 47–52.
[12]. V. S. Moreva, On the Ways of Computations Acceleration when Solving 2D Aerodynamic Problems by Using Vortex Element Method, Heralds of the Bauman Moscow State University. Natural Sciences. Sp. Issue `Applied Mathematics' (2011), pp. 83–95.
[13]. K. S. Kuzmina, I. K. Marchevsky, Estimation of computational complexity of the fast numerical algorithm for calculating vortex influence in the vortex element method, Science & Education (electronic journal), 10 (2013), pp. 399–414 (URL: http://technomag.bmstu.ru/en/doc/604030.html)
[14]. A. Grama, V. Sarin, and A. Sameh, Improving Error Bounds for Multipole-Based Treecodes, SIAM J. Sci. Comp., 21 (2000), pp. 1790–1803.
[15]. J. K. Salmon, and M. S. Warren, Skeletons from the treecode closet, J. Comput. Phys., 111 (1994), pp. 136–155.

Применение спектральных методов обработки данных к результатам численного моделирования аттракторов внутренних волн

[1]М. Провидухина <itimasha@gmail.com>
[1,2]И. Сибгатуллин <sibgat@imec.msu.ru>
[1]Механико-математический факультет МГУ им. М.В. Ломоносова, 19991, Москва, ГСП-1, Ленинские горы, д. 1
[2]ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. Проведено прямое численное моделирование распространения внутренних волн и образования волновых аттракторов в трапецеидальном контейнере, заполненном устойчиво стратифицированным раствором соли с постоянной **частотой** плавучести. Левая вертикальная стенка контейнера совершает монохроматические колебания в форме половины косинусоиды на высоту контейнера, правая стенка расположена под углом к вертикали и обеспечивает фокусировку внутренних волн, две другие границы горизонтальны. На верхней границе задано условие отсутствия касательных напряжений, на остальных условие прилипания. Рассчитываются уравнения Навье-Стокса в приближении Буссинеска в трёхмерной и двумерной постановке. Прямое численное моделирование проведено с помощью метода спектральных элементов 8-го порядка и модифицированного кода nek5000. С помощью применения преобразований Гильберта и частотно-временных диаграмм к результатам численного моделирования аттракторов внутренних волн удалось получить временные и пространственные волновые характеристики, в частности волновые векторы, соответствующие временным частотам, полученным с помощью частотно-временных диаграмм. При этом используется преобразование Гильберта с фильтрацией по узкому диапазону частот. Частотно временные диаграммы для режимов со слабой надкритичностью показывают возникновение дочерних волн на частотах, соответствующих триадному волновому резонансу. Выполнение триадного резонанса для дочерних частот демонстрируется с помощью биспектров, на которых произведение амплитуд показано на декартовом произведении частотных диапазонов. После фильтрации пространственных полей горизонтальной компоненты скорости по полученным частотам получены соответствующие волновые векторы, также удовлетворяющие условиям триадного резонанса волновых векторов. Результаты хорошо соответствуют данным экспериментов, проводимых в ENS de Lyon.

Ключевые слова: волновые аттракторы, внутренние волны, преобразование Гильберта, преобразование Фурье, частотно-временные диаграммы

DOI: 10.15514/ISPRAS-2016-28(1)-16

Для цитирования: Провидухина М., Сибгатуллин И. Применение спектральных методов обработки данных к результатам численного моделирования аттракторов внутренних волн. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 275-282. DOI: 10.15514/ISPRAS-2016-28(1)-16

1. Введение

В отличие от атмосферы Земли, мировой океан не является классической тепловой машиной, и тепловые процессы вертикального обмена энергией, наряду с ветровыми воздействиями, играют существенную роль лишь у поверхности Океана. В тоже время существенное влияние на глобальную динамику Океана оказывают глубоководные процессы перемешивания, которые в настоящее время изучены в гораздо меньшей степени, как экспериментально так и фундаментально. Существенное влияние на глобальную динамику Океана оказывают глубоководные процессы перемешивания, которые в настоящее время как экспериментально так и фундаментально изучены в гораздо меньшей степени, чем приповерхностные явления, связанные с ветровым воздействием и теплообменом. В глубинах океана внутренние волны, возникающие из-за приливных течений и обтекания донного рельефа, могут сильно влиять на процессы вертикального перемешивания.

Открытые в 1995 году Лео Маасом [1, 2] аттракторы внутренних волн показали, что в определенных геометрических конфигурациях (которые как оказалось могут быть достаточно простыми) энергия внутренних волн может эффективно аккумулироваться вдоль определенных путей. После работ Лео Мааса аттракторам внутренних волн было посвящено большое количество исследований, в результате в настоящее время имеется достаточно развитая теория формирования аттракторов внутренних волн небольшой амплитуды.

Поэтому сейчас основной интерес исследователей вызывает поведение аттракторов внутренних волн большой амплитуды. При этом линеаризованные уравнения уже не описают поведение аттракторов, которые могут становится неустойчивыми, турбулентными, менять свою структуру с течением времени, оказывать существенное влияние на фоновую стратификацию и проявлять другие нелинейные свойства.

В Высшей нормальной школе Лиона в последние годы развиваются методики анализа данных экспериментов, связанных с внутренними волнами. Они позволили качественно и количественно описать ряд важных закономерностей при распространении внутренних волн, образовании и разрушении аттракторов [3].

Несмотря на успехи в экспериментальном изучении волновых аттракторах, в методиках определения параметров течений и гидродинамических полей имеется ряд существенных ограничений. Эти ограничения снимаются прямым численным моделированием. Но рассматриваемые явления обладают очень большим интервалом пространственных масштабов из-за большого числа Прандтля-Шмидта, поэтому такие исследования вплоть до последнего времени не были возможны и численное моделирование ограничивалось лишь линейными режимами, которые лишь подтвердили уже существующую теорию [4, 5]. Лишь недавно были проведено трехмерное прямое численное моделирование, полностью соответствующие экспериментам, и впервые гидродинамические поля экспериментов и расчетов соответствовали в пределах десяти процентов как для линейных, так и для нелинейных режимов [6, 7]. В настоящей работе мы применяем спектральные и статистические методы анализа гидродинамических полей к результатам численного моделирования к слабонелинейным режимам. В дальнейшем они позволят исследовать более сложные конфигурации и сильно нелинейные (турбулентные) режимы с перемешиванием.

2. Математическая модель и прямое численное моделирование

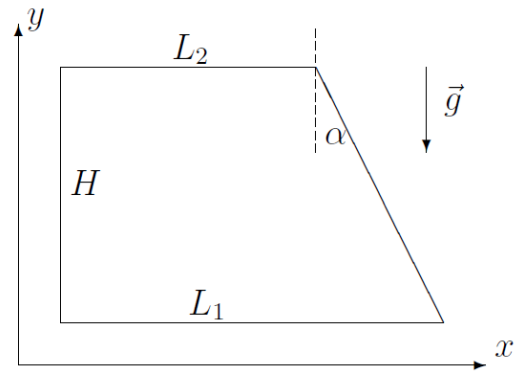


Рис. 1: 2D проекция расчетной области.

Fig. 1. 2D projection of the computational domain

Схема экспериментальной установки в двумерной проекции показана на рисунке 1.

В контейнере трапецеидальной формы находится раствор соли с постоянной по высоте частотой плавучести $N(z) = \sqrt{\frac{-g}{\rho(z)}\frac{d\rho(z)}{dz}}$. Соответственно соленость распределена экспоненциально по высоте, но в условиях эксперимента она практически не отличается от линейного распределения. Можно показать, что условия эксперимента соответствуют предпосылкам использования приближения Буссинеска даже лучше, чем традиционные задачи конвекции. Поэтому полную систему уравнений можно записать в виде:

$$\frac{\partial \vec{v}}{\partial t} + v^k \nabla_k \vec{v} = -\nabla \frac{\tilde{p}}{\rho_m} + \nu \Delta \vec{v} + \rho_s \vec{g} \qquad (1)$$

$$\frac{\partial \rho_s}{\partial t} + v^k \nabla_k \rho_s = \lambda \Delta \rho_s, \quad \mathrm{div}(\vec{v}) = 0 \qquad (2)$$

Граничные условия для скорости на боковых и нижней границах соответствуют условиям прилипания. На верхней границе ставится условие отсутствия вязких касательных напряжений. Для солености на всех границах ставится условие изоляции.

Левая граница совершает колебания следующего вида:

$$x_b(0, y, t) = a \cos(\pi y / H) \cos(\omega_0 t). \qquad (3)$$

Прямое численное моделирование проводилось с помощью спектрально-элементного метода [8] и кода nek5000.

На рисунке 2 показаны примеры устойчивого аттрактора (слева) и неустойчивого (справа). Динамика горизонтальной компоненты скорости показана на рисунках 3, 4.

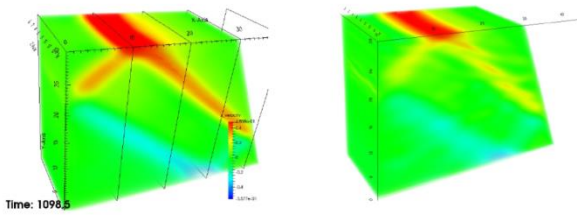


Рис. 2: Устойчивый и неустойчивый аттракторы. Цветом показана горизонтальная составляющая поля скорости.

Fig. 2. Stability and instability attractors. The colors show the horizontal component of the velocity field

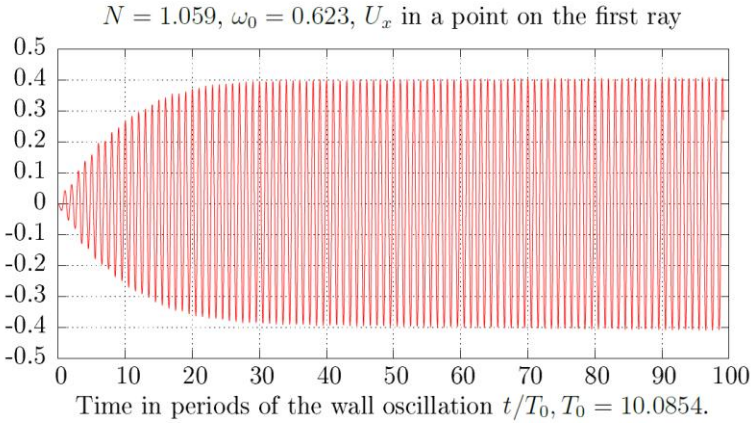


Рис. 3: Зависимость v_x от времени в точке на первом луче аттрактора.

Fig. 3. Dependence v_x on time at a point on the first ray of the attractor

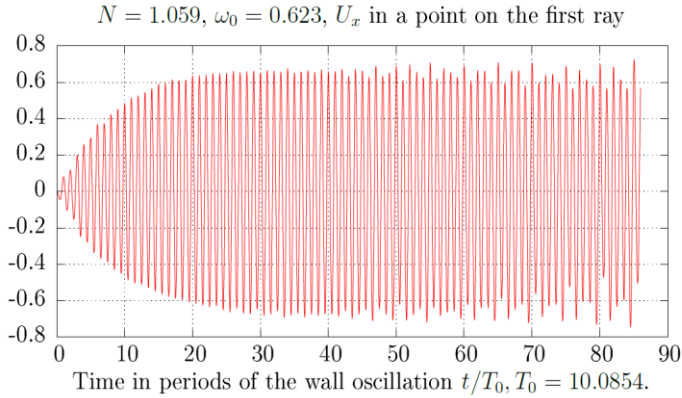


Рис. 4: Зависимость v_x от времени в точке на первом луче аттрактора. Слабая неустойчивость аттрактора.

Fig. 4. Dependence v_x on time at a point on the first ray of the attractor. Weak attractor instability.

3. Применение спектральных методов дляописание неустойчивости и процессов перемешивания

Для выявления имеющихся частот в системе построим частотно-временную диаграмму, а для проверки выполнения условий триадного резонанса на дочерние волны построим так называемый биспектр, отложив по

горизонтальной и вертикальной осям частоты, при этом цвет на диаграмме пропорционален произведению соответствующих амплитуд.

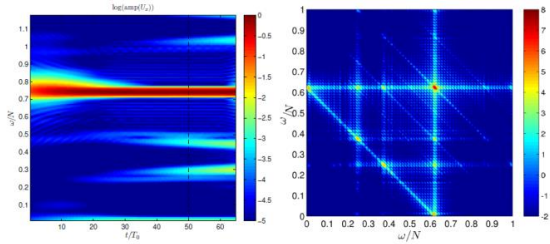


Рис. 5: Частотно-временная диаграмма и биспектр для слабонеустойчивого аттрактора

Fig. 5.Time-frequency diagram and bispectrum for weakly unstable attractor

Частотно-временная диаграмма позволяет увидеть появление вторичных частот ω_1, ω_2, а биспектр доказывает выполнение соотношений $\omega_0 = \omega_1 + \omega_2$, поскольку дочерние частоты лежат на антидиагонали квадрата, с основанием, равным родительской частоте. Применение подобной техники к результатам DNS также позволит исследовать каскады триадных резонансов процессы перемешивания и динамику спектра на длительных временах.

Список литературы

[1]. L. R. M. Maas and F.-P. A. Lam, “Geometric focusing of internal waves”, Journal of Fluid Mechanics, vol. 300, pp. 1-41, 1995.

[2]. L. R. M. Maas, D. Benielli, J. Sommeria, and F.-P. A. Lam, “Observation of an internal wave attractor in a confined, stably stratified fluid”, Nature, vol. 388, pp. 557-561, Aug. 1997.

[3]. H. Scolan, E. Ermanyuk, and T. Dauxois, “Nonlinear Fate of Internal Wave Attractors”, *Physical Review Letters*, vol. 110, p. 234501, June 2013.

[4]. N. Grisouard, C. Staquet, and I. Pairaud, “Numerical simulation of a two-dimensional internal wave attractor”, *Journal of Fluid Mechanics*, vol. 614, p. 1, Oct. 2008.

[5]. J. Hazewinkel, N.Grisouard, and S. B. Dalziel, “Comparison of laboratory and numerically observed scalar fields of an internal wave attractor”, *European Journal of Mechanics B Fluids*, vol. 30, pp. 51-56, Jan. 2011.

[6]. C. Brouzet, S. Joubaud, E. Ermanyuk, I. Sibgatullin, and T. Dauxois, “Energy cascade in internal-wave attractors”, *Europhysics Letters*, vol. 113, issue 4, pp. 44001, 2016.

[7]. C. Brouzet, Sibgatullin, E. Ermanyuk, H. Scolan, and T. Dauxois, “Internal wave attractors: experiments and numerical simulations”, *Journal of Fluid Mechanics*, vol. 793, pp. 109-131, 2016.

[8]. M. Deville, P. Fischer, E. Mund, and D. Gartling, “High-Order Methods for Incompressible Fluid Flow”, *Applied Mechanics Reviews*, vol. 56, p. B43, 2003.

Application of statistical and spectral methods to computational modeling of internal wave attractors

[1]*M.Providukhina <itimasha@gmail.com>*
[1,2]*I. Sibgatullin <sibgat@imec.msu.ru>*
[1]*MSU, Faculty of Mechanics and Mathematics, Russia, 119991, Moscow, GSP-1, 1 Leninskiye Gory, Main Building,*
[2]*ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation.*

Abstract. Direct numerical simulation of internal gravity waves propagation and formation of internal waves attractors in trapezoidal tank, which was filled with stably stratified salt solution of constant buoyancy frequency. The left vertical boundary oscillates with constant time frequency and has the form of semi-cosine of the tank height. The right wall is inclined to the vertical and performs focusing of waves, the other two boundaries are horizontal. The upper wall has stress free boundary conditions, on the other boundaries no-slip condition is imposed. Navier—Stokes equations in Boussinesq approximation and diffusive transport of salt are used as mathematical model. Direct numerical simulation is performed with the help of spectral element approach of 8-th order and modified code nek5000. Hilbert transforms and time-frequency diagrams were applied to the results of direct numerical simulation of internal wave attractors. In particular, with the help of phase reconstruction we obtained wave vectors, corresponding to time frequencies from time-frequency diagrams. To obtain phase images Hilbert transforms with filtration over narrow intervals of frequencies were used. Time-frequency diagrams for moderate forcing amplitudes show appearance of daughter waves of frequencies, corresponding to triadic resonance, which is also demonstrated with the help of bispectra: product of amplitudes on the background of cross product of frequency intervals. The results are close to data of experiments, being carried out at ENS de Lyon.

Keywords: internal waves, wave attractor, Hilbert transform.

DOI: 10.15514/ISPRAS-2016-28(1)-16

For citation: Providukhina M., Sibgatullin I. Application of statistical and spectral methods to computational modeling of internal wave attractors. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 275-282 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-16

References

[1]. L. R. M. Maas and F.-P. A. Lam, “Geometric focusing of internal waves”, Journal of Fluid Mechanics, vol. 300, pp. 1-41, 1995.
[2]. L. R. M. Maas, D. Benielli, J. Sommeria, and F.-P. A. Lam, “Observation of an internal wave attractor in a confined, stably stratified fluid”, Nature, vol. 388, pp. 557-561, Aug. 1997.
[3]. H. Scolan, E. Ermanyuk, and T. Dauxois, “Nonlinear Fate of Internal Wave Attractors”, Physical Review Letters, vol. 110, p. 234501, June 2013.
[4]. N. Grisouard, C. Staquet, and I. Pairaud, “Numerical simulation of a two-dimensional internal wave attractor”, Journal of Fluid Mechanics, vol. 614, p. 1, Oct. 2008.

[5]. J. Hazewinkel, N.Grisouard, and S. B. Dalziel, “Comparison of laboratory and numerically observed scalar fields of an internal wave attractor”, European Journal of Mechanics B Fluids, vol. 30, pp. 51-56, Jan. 2011.
[6]. C. Brouzet, S. Joubaud, E. Ermanyuk, I. Sibgatullin, and T. Dauxois, “Energy cascade in internal-wave attractors”, Europhysics Letters, vol. 113, issue 4, pp. 44001, 2016.
[7]. C. Brouzet, Sibgatullin, E. Ermanyuk, H. Scolan, and T. Dauxois, “Internal wave attractors: experiments and numerical simulations”, Journal of Fluid Mechanics, vol. 793, pp. 109-131, 2016.
[8]. M. Deville, P. Fischer, E. Mund, and D. Gartling, “High-Order Methods for Incompressible Fluid Flow”, Applied Mechanics Reviews, vol. 56, p. B43, 2003.