ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 8
N 8
2017

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 338.364.2 DOI: 10.17587/prin.8.339-344

Д. С. Пащенко, канд. техн. наук, MBA, независимый консультант в области разработки программного обеспечения, e-mail: denpas@rambler.ru, Москва

Отражение в российской практике мировых тенденций в технологиях, средствах и подходах в разработке программного обеспечения

Технологии и подходы к разработке программного обеспечения (ПО) стремительно меняются, а в высококонкурентной среде зрелость производственных процессов является существенным фактором коммерческого успеха. Россия является частью глобального рынка разработки ПО, а российские команды участвуют во многих международных программных проектах. В данной статье представлена часть результатов авторского исследования, проведенного весной 2017 г. и охватившего 79 опытных инженеров из всех федеральных округов России. В рамках этого исследования была рассмотрена востребованность новейших и значимых мировых тенденциях в технологиях разработки программного обеспечения. К общим выводам можно отнести высокую востребованность популярных инструментальных средств разработки ПО в российской практике: от системы хранения версий GIT до JavaScript как полноценного языка разработки. Эксперты также отметили относительное и запаздывающее соответствие российского опыта в подходах к разработке ПО текущим мировым тенденциям и довольно оптимистичные оценки востребованности новейших трендов.

Ключевые слова: *технологии разработки ПО, инструментарий разработки ПО, перспективные производственные практики, российский опыт*

Введение

Разработка программного обеспечения (ПО) — это одна из отраслей "новой экономики", в которой ожидания потребителей, технологии и организационные подходы меняются очень стремительно [1]. Вместе с тем в мире высоких технологий политические и культурные границы очень призрачны, и благодаря глобализации удачные инновации очень быстро становятся популярными трендами, находят многочисленных сторонников и практические внедрения в разных уголках мира. Изменения в самих технологиях и средствах разработки ПО также существенны, они активно влияют на конкурентные преимущества софтверных вендоров (производителей ПО). Российский рынок разработки ПО полностью интегрирован в мировой рынок, что следует и из структуры доходов крупнейших российских частных IT-компаний [2] и из наличия в России многочисленных центров разработки международных софтверных гигантов, расположенных в российских городах [3]. Понимание и адаптированное использование мировых трендов в разработке ПО позволяет российским компаниям, командам разработки и их продуктам сохранять конкурентные возможности на общем рынке, оптимизировать свой бизнес в условиях российских экономических проблем последних лет. Технологии производства ПО, языки, инструментарий и среды разработки оказывают существенное влияние на уровень качества и быстроту создания новых релизов ПО в ситуации высокой конкуренции. Отмеченные факторы подчеркивают актуальность исследования степени востребованности новейших трендов в российских командах разработки и необходимость оценки общего уровня развития отечественной IT-отрасли на технологическом уровне.

В настоящей статье приведено обобщение результатов авторского исследования, цель которого — определение степени востребованности в России мировых трендов в области технологий и подходов к проектированию информационных систем, актуальных в 2016—2017 гг. Исследование путем анкетирования с отложенной обратной связью было проведено во всех федеральных округах России в марте—апреле 2017 г. и охватило 79 опытных инженеров, руководителей проектов, архитекторов ПО. В исследовании были поставлены следующие задачи:

1) определить степень востребованности в российских регионах текущих мировых трендов в разработке и проектировании информационных систем, подходов к организации производства;

2) получить квалифицированное мнение экспертов о локальных российских трендах, связанных с регулирующей ролью государственных органов и ожиданиями заказчиков с долей капитала, принадлежащего государственных органам, в области импортозамещения и обработки персональных данных.

В данной статье освещены вопросы только первой задачи исследования, посвященные технологиям и подходам к проектированию информационных систем.

В предложенной экспертам анкете тренды были разделены на группы (по области применения в разработке ПО) и на подгруппы по степени их актуальности (тренды "на пике" и "восходящие" тренды). Для каждого тренда были предложены техническое описание и набор мнений, одно из которых выбирал каждый эксперт. Эксперты выражали свое мнение, основанное на опыте последних 2...4 лет практической работы в командах разработки ПО, частично или полностью расположенных в определенном регионе России. Обобщенные результаты исследования были отправлены экспертам. Часть экспертов дала по ним обратную связь, которая, впрочем, во всех случаях была положительной и не внесла никаких изменений в итоговые результаты.

Популярность технологий и инструментальных средств

Разработка ПО, как и любые другие высокотехнологические отрасли, использует сложные средства производства — другие программные продукты и технологии. Их эволюция очень стремительна. В данном разделе приведены результаты исследования применительно к оценке степени востребованности в российских регионах самых передовых мировых технологий и инструментальных средств создания ПО.

Среди технологических тенденций "на пике" следует выделить значительную распространенность языков программирования C++ и Java как основных языков разработки прикладного и системного ПО. Большинство новых создаваемых информационных систем (не веб-сайтов) разрабатываются на данных языках программирования, а используемые популярные среды разработки поддерживают данные языки. Эксперты в целом согласились с данными высказываниями (рис. 1, см. вторую сторону обложки).

Использование JavaScript как полноценного языка разработки становится реальностью. Довольно много проектов и разработчиков сейчас создают решения, используя JavaScript на всех уровнях разработки, а не только в части веб-приложения. Эксперты также подтвердили значительный рост популярности JavaScript как полноценного языка разработки, применяемого на всех этапах разработки информационной системы вне зависимости от ее архитектуры (рис. 2, см. вторую сторону обложки).

Инструментарий для обеспечения версионности программных продуктов GIT (*Distributed version control system*) был одним из самых мощных проектов в истоках движения "свободного ПО" [4]. В настоящее время в мире наблюдается лавинообразный рост его популярности в коммерческих компаниях, которые предпочитают использование этого инструментария вместо проприетарных решений коммерческих поставщиков. Почти 90 % экспертов также отмечают значительный рост популярности GIT в отечественных командах разработки.

Состояние подходов к разработке и проектированию ПО

Текущее состояние подходов к разработке и проектированию программных продуктов в России можно описать следующим образом: почти все тренды, находящиеся в мире на самом пике популярности, нашли свое отражение в российских командах разработки. Так, очевидна популярность "гибких" подходов (*Agile*) и организации непрерывной поставки и внедрения (*continues improvement*). "Гибкие" и "гибридные" подходы в разработке в мире занимают доминирующее положение. Большинство мировых вендоров ПО сделали свой выбор в пользу Agile-подходов. В России данный подход также занимает доминирующее положение. Следует отметить, что российские команды дольше шли к этому положению дел. В более ранних исследованиях автора в России не наблюдалось такого доминирования "гибкой" разработки [5]. На рис. 3 (см. вторую сторону обложки) приведено мнение экспертов об использовании "гибких" и "гибридных" методов в разработке ПО в отечественной практике.

Непрерывные поставка ПО и интеграция (релизы каждый день) — это одно из следствий "гибких" методологий разработки, набирающее популярность во всем мире. Однако практическая реализация непрерывной поставки и интеграции требует кардинального перестроения производственных процессов. В основе непрерывной поставки и внедрения лежит не только организованный выпуск регулярных обновлений ПО и их стремительная интеграция в продуктивную среду, но и очень точное управление требованиями. Ожидается, что в прогрессивных командах, занимающихся разработкой для внутренних нужд компании, данный подход станет доминирующим в течение двух—трех следующих лет. Эксперты отмечают стремительный рост популярности "непрерывной поставки и интеграции ПО": около половины экспертов наблюдают следование такому подходу среди знакомых команд и проектов, еще треть отмечает его применение в единичных и редких случаях.

В исследовании были выделены мировые тенденции в проектировании информационных систем, только набирающие популярность за последний год. По мнению автора, российская экспертная группа была излишне оптимистична в оценке прогресса отечественных команд в следовании новейшим трендам этого пула. Однако в обратной связи эксперты

в данной части подчеркивали согласованность результатов исследования с наблюдаемым положением дел в отрасли.

Набирающий популярность подход BaaS (*Backend As A Service*) — это унификация повторяющихся функций (хранение и сбор информации, стандартные функции и запросы и т. п.) [6], которая позволяет разработчикам современных систем передавать их исполнение компонентам/сервисам третьей стороны — другому поставщику ПО. Этот подход серьезно меняет архитектуру проектируемых приложений (модульность, стандартность API). Подход BaaS позволяет не создавать однотипные функции с нуля в своих проектах, а отдавать их исполнение другим решениям. Это экономит время и трудозатраты команды, однако, вносит некоторые ограничения в функциональные возможности систем. Эксперты в целом не наблюдают эту тенденцию в России, но общий процент "наблюдавших примеры таких IT-решений в России" очень высок — около 41 % экспертов.

Еще один прогрессивный подход — это DevOps — новая попытка на ранних этапах совместить интересы разработчиков, инженеров сопровождения и бизнес-заказчиков [7]. В основе этого подхода — набор бизнес-процессов, позволяющих всем участникам проекта "говорить на одном языке", совместно принимать участие в изменении всех ключевых параметров проекта. Подход DevOps стремительно набирает популярность как у поставщиков ПО, так и в компаниях, занимающихся внутренней разработкой (*in-house*). Эксперты, среди которых только 15 % (как будет показано далее) работают в in-house-разработке, однозначно указывают на рост популярности практик DevOps среди их знакомых команд разработки.

Отметим, что 57 % экспертов указывают на элементы применения DevOps в российских компаниях. По мнению автора, такой ответ также довольно оптимистичен.

Использование микросервисов в интеграции систем — это довольно новый тренд, набирающий популярность в мире [8]. Интеграция систем на уровне предприятия все чаще требует гибкого подхода, экономящего ресурсы заинтересованных сторон. Один из современных подходов — это интеграция систем через микросервисы вместо использования шины данных или еще более устаревающих вариантов архитектуры. Тем временем, в ведущих системных интеграторах и крупных европейских поставщиках ПО микросервисы стали де-факто стандартом отрасли. В отечественной практике эксперты в целом не видят реального появления данного подхода — менее трети экспертов отмечали реальное использование микросервисов в своих проектах.

Реальное использование концепции Internet of Things (IoT) в проектировании программных решений экспертам кажется скорее близким будущим, чем настоящим [9]. Сама концепция IoT набирает популярность уже много лет, однако, как правило, о ней можно больше услышать на конференциях производителей бытовой техники и роботов-автоматов, чем на мероприятиях, посвященных разработке прикладного ПО. Вместе с тем в последнее время появляются реальные проекты, в которых обмен информацией на высоких уровнях между устройствами и получение бизнес-результата без взаимодействия с человеком становится конкурентным преимуществом. И даже 20 % экспертов отмечают элементы IoT в проектах и IT-решениях знакомых команд разработки ПО. Данный результат также кажется автору излишне оптимистичным.

Такие различные варианты реализации мобильных приложений, как нативные приложения (*Native Apps*) и приложения, открываемые в браузерах (*Web Apps*), ведут конкурентный спор много лет. Браузеры улучшаются, а число скачиваемых в США и Европе нативных приложений падает от месяца к месяцу. Приложение Web Apps не зависит ни от модели смартфона, ни от операционной системы — оно просто откроется в популярном браузере на любом устройстве. Многие инновационные американские стартапы вообще не делают нативных версий своего ПО [10]. Участвующим в исследованиях экспертам было предложено оценить популярность тенденции разработки Web Apps вместо мобильных нативных приложений. В целом они не отметили тенденцию отказа от нативных приложений в России, что свидетельствует о том, что отечественный рынок в отличие от рынков США и Западной Европы, с точки зрения ожидания потребителей, возможно, проходит предыдущую стадию развития мобильного ПО. На рис. 4 (см. вторую сторону обложки) представлено мнение экспертов, которое в целом подтверждает, что нативные и веб-приложения продолжают существовать вместе.

Подробнее об экспертной группе

В настоящем разделе приведены данные, позволяющие оценить состав экспертной группы, которая представила свое интегральное мнение о востребованности мировых тенденций в технологиях разработки ПО в российских регионах. Приведена информация о возрасте, опыте, регионах и направленности работы экспертов.

Данные об опыте экспертов представлены в табл. 1.

Как видно из данных табл. 1, почти половина экспертов имеет опыт в отрасли разработки ПО десять и более лет, а это означает, что они успели застать две и более отраслевые революций, существенно изменившие производство ПО. Очевидно, что эта группа экспертов начинала карьеру еще при господстве в российском производстве "водопадной модели создания ПО". Участвующие в исследовании эксперты также успели пройти фазу итерационных подходов и пришли к современному доминированию "гибридных" и "гибких" методологий.

Данные о возрастных группах экспертов представлены в табл. 2.

Таблица 1

Сколько лет Вы занимаетесь профессиональной разработкой ПО, соответствующими проектами и командами?

Вариант ответа	Доля экспертов в группе, %
1...3 года	2,5
3...6 лет	20,3
6...10 лет	32,9
Более 10 лет	44,3

Таблица 2

Определите свою возрастную группу

Вариант ответа	Доля экспертов в группе, %
Старше 40 лет	5
30...39 лет	53,2
20...29 лет	41,8

Из данных табл. 2 следует, что более половины экспертов находятся условно в самом продуктивном для отрасли возрасте — от 30 до 39 лет. Это этап наиболее стремительного профессионального и карьерного роста для большей части профессионалов разработки ПО.

С точки зрения направления разработки ПО важно понимать отраслевую специфику — разработка для внешних потребителей очень сильно отличается по организации от разработки для внутренних нужд компании. Эти различия настолько объемны, что заслуживают отдельного исследования. Однако инженеры обоих направлений создают уникальные программные продукты, реализуют в России и мире "цифровую революцию", используют схожие технологии и инструментальные средства в разработке своего ПО.

Данные об экспертах в части направленности их разработки ПО представлены в табл. 3.

Таблица 3

Представленный Вами опыт последних 2...4 лет

Вариант ответа	Доля экспертов в группе, %
Проекты системной интеграции (system integrator)	11,4
Разработка ПО для собственных нужд компании (in-house development)	15,2
Разработка ПО на заказ (включая out-sourcing)	36,7
Разработка ПО (сервисов, технологий) независимым поставщиком (вендор ПО)	36,7

Таблица 4

Определите регион проживания (столицу, Федеральный округ), в котором получен представленный вами опыт

Вариант ответа	Доля экспертов в группе, %
Москва	34,1
Сибирский ФО	21,5
Приволжский ФО	12,7
Северо-Западный ФО (включая Санкт-Петербург)	10,1
Южный и Северо-Кавказский ФО	7,6
Центральный ФО (без Москвы)	6,3
Уральский ФО	5,2
Дальневосточный ФО	2,5

Разработка ПО на заказ считается наиболее технологически продвинутой, так как в сложных условиях конкурентной борьбы необходимо каждый год оказываться "более выгодными", чем in-house-разработка и уже готовые программные продукты от независимых поставщиков. Опыт группы приглашенных экспертов по направлениям деятельности кажется очень сбалансированным и с правильным превалированием разработки на заказ и разработки ПО независимым поставщиком над другими категориями.

Данные о географической принадлежности экспертов представлены в табл. 4.

Представленная разбивка опыта группы экспертов в целом отражает авторское представление о распределении трудовых ресурсов на IT-рынке в России. Москва традиционно является центром информационных технологий в России, а Новосибирск, используя образовательный и научный потенциал, в последние годы становится значительным центром разработки ПО в России, оттесняя Санкт-Петербург. Столичные компании создают свои центры разработки в центральных регионах России, опираясь на технологические вузы. Следует также отметить возрастающее влияние китайских IT-производителей на рынок высокотехнологичного труда Дальневосточного и Сибирского Федеральных округов: все больше российских специалистов работает в китайских и китайско-российских IT-компаниях.

Заключение

Востребованность в России трендов по технологиям и подходам к разработке ПО, рассмотренная в авторском исследовании и данной статье, подчеркивает интегрированность российских команд и компаний в мировую IT-отрасль. Практически все тренды, находящиеся на пике популярности

в странах — мировых лидерах рынка разработки программного обеспечения, находят свое отражение в России. Использование "гибких" и "гибридных" методик производства становится доминирующим, хотя этот путь и занял некоторое дополнительное время. Российские команды используют те же среды разработки, языки, инструментарий, что и лучшие мировые компании, отмечается все большая востребованность GIT и JavaScript (как единственного языка разработки на всех стадиях проекта). Как и во всем мире, в России языки разработки Java и C++ с некоторыми оговорками занимают доминирующее положение, как в части их популярности в изучении, так и в практическом применении при создании информационных систем. Участвующая в исследовании группа экспертов отмечает, что тренды "новой волны" довольно быстро находят отражение в российской практике производства ПО. Так подход DevOpps "становится все более популярен и востребован". Это позволяет использующим его компаниям получить дополнительные конкурентные преимущества — сделать поставки релизов ПО более быстрыми, точнее удовлетворять требованиями потребителей. Процесс обеспечения непрерывной поставки и интеграции ПО в российских компаниях, по мнению экспертной группы, невысок. Такое очевидное отставание от общемирового уровня оставляет потенциал для развития отечественного бизнеса, использующего прежде всего разработку ПО для внутренних целей компании (in-house).

Такие "нарастающие тренды", как подход BaaS или использование микросервисов в интеграции, пока не нашли достаточного воплощения на российском рынке, хотя значительная часть экспертов (прежде всего в Москве) отметили некоторые признаки роста их популярности. Микросервисы в интеграции на уровне предприятия уже стали доминирующим подходом в США и Европе в проектах для финансовых, транспортных и телекоммуникационных компаний. В России этот тренд, по мнению автора, также станет доминирующим в течение 2...3 следующих лет. Подход BaaS, безусловно, станет новым стандартом в проектировании сложных информационных систем в мире. Степень его использования в России определить пока сложно, однако некоторые элементы этого подхода используются в реальных проектах уже сейчас.

Еще менее популярными в России представляются тенденции отказа от нативных приложений в пользу Web App и реальное применение концепции IoT в IT-проектах. Данные тренды, по мнению автора, еще должны будут пройти "свой тернистый путь" на более зрелых рынках создания ПО — в США, Японии и Западной Европе, найти коммерчески успешные области применения, получить значительное количество последователей из числа IT-экспертов и практиков рынка. Возможно, что их реализация на других рынках, например, в России или Африке будет менее яркой и пройдет с отставанием в десять—пятнадцать лет.

Полученные в результате исследования данные в целом согласуются с текущей ролью России в мировой экономике производства программного обеспечения. Ее можно оценить как роль страны "второго эшелона" с мощным человеческим потенциалом и полной интегрированностью в мировые технологии производства ПО [11]. Автор считает, что выявленная востребованность в самых новых технологиях и подходах к производству ПО демонстрирует конкурентоспособность отечественных команд и компаний, разрабатывающих программные продукты. Она может быть (при разумной государственной политике) одним из составляющих компонентов в повышении экономической значимости информационных технологий как в структуре экспорта российской экономики, так и в возможностях удовлетворения внутреннего спроса.

Список литературы

1. **Авдокушин Е.** "Новая экономика": сущность и структура: Экономическая теория на пороге XXI века — 5. Неоэкономика / Под ред. Ю. М. Осипова и др. М.: Юристъ, 2001. 624 с.
2. **Russian** software developing industry and software exports, 9-th survey, 2012. P. 61—62. URL: http://www.russoft.ru/files/RUSSOFT_Survey_12_en.pdf
3. **Прохоров Н.** Экспорт ПО из России и перспективы его роста // Компьютер Пресс. 2011. № 11. URL: http://compress.ru/article.aspx?id=22566#05
4. **Chacon S.** Pro Git (2nd ed.). New York, NY: Apress, 2014.
5. **Pashchenko D., Blinov O.** Standardization in software production at the corporate level: results of research in CIS // Business Informatics Journal. 2014. № 4. С. 63—70.
6. **Silicon** India Review, Understanding the Basics of Backend as a Service (BaaS) // Mobile City. 2012. URL: http://mobile.siliconindia.com/news/Understanding-the-Basics-of-Backend-as-a-Service-BaaS-nid-126045.html
7. **Walls M.** Building a DevOps Culture. OReilly Media, 2013.
8. **Newman S.** Building Microservices, Designing Systems, 2015.
9. **Kranenburg R.** The Internet of Things: A critique of ambient technology and the all-seeing network of RFID. Pijnacker: Telstar Media, 2008. 62 p.
10. **Будущее** рядом: инновационные технологии для ресторанов // HoReCa.ru. 2016. № 06. URL: http://www.horeca-magazine.ru/article/3666/
11. **Russian** software developing industry and software exports, 13-th survey, 2016. P. 4—38. URL: http://www.russoft.ru/files/RUSSOFT_Survey_13_en.pdf

Reflection in the Russian Practice of World Trends in Technologies, Tools and Approaches to Software Development

D. S. Pashchenko, denpas@rambler.ru, SlavaSoft, Moscow, 125368, Russian Federation

Corresponding author:

Pashchenko Denis S., CEO, SlavaSoft, Moscow, 125368, Russian Federation,
E-mail: denpas@rambler.ru

Received on May 26, 2017
Accepted on June 09, 2017

Technologies and approaches to software development (SD) are changing rapidly, and in a highly competitive environment the maturity of production processes is an essential factor of commercial success. Russia is part of the global software development market, and Russian teams are involved in a large number of international projects. This article contains some of the results of the author's research finished in the spring of 2017 and covered 79 experienced engineers from all Federal Districts of Russia. The study demonstrated the relevance of the latest and significant global trends in technologies and approaches to software development in Russian regions. There is a strong demand for popular software development tools in Russian practice: from Git (version storage system) to JavaScript as a language for full-stack development. The panel of experts also noted the relative and lagging correspondence of Russian experience in software development approaches to current global trends and quite optimistic estimation of the demand for new trends.

Keywords: *software development technologies, software development tools, promising manufacturing practices, Russian experience*

For citation:

Pashchenko D. S. Reflection in the Russian Practice of World Trends in Technologies, Tools and Approaches to Software Development, *Programmnaya Ingeneria,* 2017, vol. 8, no. 8, pp. 339—344.

DOI: 10.17587/prin.8.339-344

References

1. **Avdokushin E.** "*Novaja jekonomika": sushnost' i struktura: Jekonomicheskaja teorija na poroge XXI veka-5. Neojekonomika* ("New economy": essence and structure: The economic theory on the dawn of the 21st century-5. Neoeconomy) / Eds. Ju. M. Osipov i dr., Moscow, Jurist, 2001, 624 p. (in Russian).

2. **Russian** software developing industry and software exports, 9-th survey, 2012, pp. 61—26, available at: http://www.russoft.ru/files/RUSSOFT_Survey_12_en.pdf (in Russian).

3. **Prohorov N.** Eksport PO iz Rossii i perspektivy ego rosta (Software export from Russia and its prospects), *Komp'yuter Press*, 2011, vol. 11, available at: http://compress.ru/article.aspx?id=22566#05 (in Russian).

4. **Chacon S**. *Pro Git* (2nd ed.), New York, Apress., 2014.

5. **Pashchenko D., Blinov O.** Standardization in software production at the corporate level: results of research in CIS, *Business Informatics Journal*, 2014, no. 4, pp. 63—70.

6. **Silicon** India Review, Understanding the Basics of Backend as a Service (BaaS), *Mobile City*, 2012, available at: http://mobile.siliconindia.com/news/Understanding-the-Basics-of-Backend-as-a-Service-BaaS-nid-126045.html

7. **Walls M.** *Building a DevOps Culture,* OReilly Media, 2013.

8. **Newman S.** *Building Microservices*, Designing Systems, 2015.

9. **Kranenburg R.** *The Internet of Things: A critique of ambient technology and the all-seeing network of RFID*, Pijnacker, Telstar Media, 2008, 62 p.

10. **Budushchee** ryadom: innovacionnye tekhnologii dlya restoranov (Future is near: innovative technologies for restaurants), *HoReCa.ru,* 2016, no. 6, available at: http://www.horeca-magazine.ru/article/3666/

11. **Russian** software developing industry and software exports, 13-th survey, 2016, pp. 4—38, available at: http://www.russoft.ru/files/RUSSOFT_Survey_13_en.pdf

УДК 004.43 DOI: 10.17587/prin.8.345-353

В. А. Васенин, д-р физ.-мат. наук, проф., **М. А. Кривчиков**, канд. физ.-мат. наук, ст. науч. сотр., e-mail: maxim.krivchikov@gmail.com, МГУ имени М. В. Ломоносова

Методы промежуточного представления программ

Представлена классификация промежуточных представлений программ, написанных на различных языках программирования. Такие представления используются на практике в задачах трансляции программ, разработанных на нескольких языках. Основными критериями классификации являются синтаксические характеристики представления (являются ли частью представления нетривиальные синтаксические конструкции), способ описания потока исполнения (синтаксические конструкции, графы или байт-код), а также типизация элементов представления (сохраняются ли типы выражений исходного языка программирования в промежуточном представлении; какова степень соответствия типов в промежуточном представлении и типов в исходном языке программирования). С позиций предлагаемой классификации представлено современное состояние исследований в области промежуточных представлений программ.

Ключевые слова: *языки программирования, промежуточное представление, трансляция программ, формальная семантика программ, классификация, обзор*

Введение

Понятие промежуточного представления программы используется, в первую очередь, в рамках теории трансляции программ. При этом известны примеры применения промежуточных представлений как в задачах совместной трансляции программ, написанных с использованием нескольких языков программирования, так и для более удобного представления программ, написанных на одном языке, на различных стадиях их трансляции.

В настоящей статье представлена классификация и дана характеристика современному состоянию исследований в области промежуточных представлений программ на высоком уровне их абстракции (независимости) от аппаратной платформы, на которой они могут исполняться. Представленный в статье краткий обзор выполнен в контексте решения задачи разработки промежуточного представления для описания высокоуровневой формальной семантики и верификации функциональных свойств программ, написанных на нескольких языках программирования, в первую очередь, предметно-ориентированных. Основными предметами настоящего обзора и анализа являются подходы к формальному (математическому) описанию промежуточных представлений программ, а также к описанию в терминах представлений различных особенностей семантики языков программирования. В статье не рассмотрены промежуточные представления с узкой областью применения, в том числе ориентированные исключительно на оптимизацию кода программ, распараллеливание и на низкоуровневое описание асинхронных систем. Кроме того, не ставится цель получения исчерпывающего обзора, поскольку в настоящее время в той или иной степени промежуточные представления используются практически в любой реализации языка программирования. Целью настоящей статьи является систематизация и анализ уже существующих решений на рассматриваемом направлении и выделение особенностей промежуточного представления для различных категорий языков программирования.

Определения основных терминов, которые встречаются в тексте настоящей статьи, даны далее.

Язык промежуточного представления программ — частный случай языка программирования, который используется в качестве интерфейса взаимодействия между определенными стадиями трансляции программы. Промежуточное представление программ не предназначено для использования человеком. Как правило, синтаксическое описание промежуточного представления ограничивается абстрактным синтаксическим деревом или байт-кодом. Далее для краткости вместо термина "язык программирования" будем использовать "язык", а в качестве сокращения для термина "язык промежуточного представления программ" — "представление".

Байт-код — форма промежуточного представления программ, в которой код программы задан в виде линейной последовательности инструкций. Такая линейная последовательность кодируется в двоичном виде, аналогично машинному коду.

Базовый блок — последовательность инструкций программы, которая имеет одну точку входа, одну точку выхода и не содержит инструкций передачи управления ранее, чем в точке выхода.

Симуляция — рефлексивное транзитивное отношение на парах моделей формальной семантики языка программирования, согласно которому каждый шаг вычислений в первой модели соответствует некоторому шагу вычислений во второй модели с сохранением отношения следования.

Изложение материала в статье построено следующим образом. В последующих разделах, соответству-

ющих предлагаемой классификации, перечислены основные особенности промежуточных представлений, которые, на взгляд авторов, наиболее полно характеризуют современное состояние исследований в этой области. Заключение содержит обобщение и анализ результатов, которые приведены в основной части статьи.

Низкоуровневые промежуточные представления, используемые в компиляторах в машинный код

Промежуточные представления программы на низком уровне абстракции от аппаратной платформы, на которой предполагается ее использование, характеризуются линейной структурой кода функций, возможностью описания типов данных, преимущественно с позиций их хранения в памяти, а также близостью набора инструкций к инструкциям машинного кода. Основным представителем данного класса является байт-код LLVM (*Low Level Virtual Machine*), основанный на абстрактном представлении SSA (*Static Single Assignment Form*). Как правило, подобные промежуточные представления слишком близки к машинному коду для адекватного описания формальной семантики. Однако три примера, приведенные далее, показывают, что принципиальных ограничений по описанию формальной семантики таких языков нет.

В работе [1] в рамках проекта Vellvm определена в терминах среды Coq формальная спецификация подмножества инструкций LLVM. Определяется статическая семантика байт-кода и следующие пять моделей операционной семантики: недетерминированная мелкошаговая; детерминированная мелкошаговая (уточняет ряд аспектов семантики, которые не зафиксированы в спецификации байт-кода); крупношаговая с выполнением вызовов функций за один шаг; крупношаговая с выполнением базовых блоков за один шаг; семантика в терминах интерпретатора.

Типизированный язык ассемблера (TAL) [2] предназначен для использования в виде промежуточного представления нижнего уровня абстракции, из которого возможна непосредственная генерация машинного кода. Блоки кода задаются в представлении TAL с помощью тройки (H, R, I), где H — спецификация кучи (частичное отображение из меток в значения на куче; среди значений на куче могут находиться блоки кода), R — спецификация регистров (типы значений, которые находятся в каждом из регистров на момент начала выполнения блока) и I — последовательность инструкций. Переходы между блоками могут происходить только с использованием меток (а не адресов в явном виде), что гарантирует безопасность типов.

Понятие фундаментального кода, несущего доказательство свойств программы (*foundational proof-carrying code*) было предложено А. Аппелем в работе [3]. В рамках демонстрации реализации этого подхода логика второго порядка с аксиомами арифметики в рамках среды Twelf используется для описания операционной семантики машинного кода. Затем в терминах этого представления определяются требуемые свойства кода и выполняется построение доказательства свойств для заданного кода программы. Следует отметить, что аппарат логики второго порядка по сравнению с исчислением конструкций имеет ограниченные выразительные средства. В частности, описание свойств, зависящих от значений переменных, выполняется в ней с помощью предикатов. Кроме того, низкий уровень абстракции машинного кода от оборудования требует формулировать свойства программы в терминах участков памяти.

Виртуальные машины уровня приложений на основе байт-кода, ориентированные на императивные языки программирования

Отличительной особенностью представлений, составляющих основу виртуальных машин уровня приложений для императивных языков программирования (в том числе объектно-ориентированных языков), является более высокий, по сравнению с низкоуровневыми представлениями, уровень абстракции от аппаратной платформы. Эта особенность выражается в наличии инструкций, которые не имеют прямых аналогов в машинном коде распространенных архитектур. В отличие от низкоуровневых промежуточных представлений, более типичным является использование стековых, а не регистровых машин. В таких промежуточных представлениях допускается описание типов данных и функций, поддерживающих параметрический полиморфизм первого порядка. Промежуточные представления, ориентированные на объектно-ориентированные императивные языки, как правило, позволяют в том или ином виде определять типы данных с позиций иерархии наследования.

Стандарт *Common Language Infrastructure* (CLI) [4] определяет систему типов, формат метаданных, виртуальную машину и систему команд для нее, а также набор библиотек, которые в совокупности составляют инфраструктуру взаимодействия произвольных языков программирования, удовлетворяющих этому стандарту. Отличительной особенностью представлений CLI и JVM (*Java Virtual Machine*, в настоящий обзор не входит, так как в основных особенностях аналогична представлению CLI, которое представляет больший интерес с позиций настоящей статьи) является хранение информации об используемых в программах типах данных на высоком уровне абстракции от аппаратной платформы. Информация о типах в представлении CLI хранится в реляционной форме — в виде набора таблиц, в которых содержится информация о составе и типах полей классов, о функциях, дополнительных атрибутах и т. д. Код программы представлен в CLI в виде байт-кода — линейной последовательности инструкций виртуальной машины с переходами по смещению. Набор инструкций виртуальной машины поддерживает:

- низкоуровневые инструкции, аналогичные инструкциям распространенных архитектур про-

цессоров (арифметические инструкции, операции сравнения, условные и безусловные переходы, инструкции вызова);

- распространенные инструкции, предоставляющие абстракцию от аппаратной платформы (чтение и запись заданного поля структуры данных);
- специальный набор инструкций для поддержки объектно-ориентированного программирования (проверка наличия типа в иерархии наследования для заданного значения, виртуальный вызов).

Следует отметить, что система типов CLI поддерживает параметрический полиморфизм — обобщенные типы, при определении которых может быть задан ряд типов-параметров с ограничениями. Ограничения могут иметь вид "данный тип должен наследоваться от заданного типа", "данный тип должен предоставлять реализацию заданного интерфейса" или "данный тип должен иметь конструктор по умолчанию". Еще одним важным аспектом представления CLI (с позиций настоящей статьи) является возможность присваивать записям в таблицах типов произвольные типизированные данные в форме пользовательских атрибутов. Такая расширяемость позволяет хранить в стандартном формате дополнительную информацию, структура которой не определена в стандарте.

Необходимо обратить внимание на то обстоятельство, что пользовательские атрибуты CLI имеют ограничения. В реализации средства Code Contracts вместо пользовательских атрибутов данные записываются непосредственно в код верифицируемой функции в виде инструкций виртуальной машины. Это может объясняться тем фактом, что механизм пользовательских атрибутов не позволяет сохранять данные с требуемой для средств такого рода гранулярностью, а именно не поддерживает присваивание атрибутов базовым блокам. Набор инструкций и модель исполнения виртуальной машины допускают определение обобщенных функций без подстановки параметров типов, а также анализ типов выражений. Работы по описанию формальной статической семантики CLI выполнялись, в частности, и авторами настоящей статьи [5].

Промежуточное представление Vortex IL [6] предназначено для компилятора программ, написанных на разных объектно-ориентированных языках программирования (таких, например, как C++, Java и Modula-3). Для единообразного представления статических процедур, функций-членов классов и мультиметодов в Vortex IL используется понятие обобщенной функции, под которым понимается множество функций с атрибутами, ассоциирующими их с классами — узлами в дереве наследования. Статическая перегрузка имен функций не поддерживается, предполагается ее реализация в интерфейсе компилятора с помощью техники *name mangling*, которая кодирует во внутреннем идентификаторе функции типы ее аргументов. В представлении Vortex IL выделены инструкции для отправки сообщений (вызова методов-членов для экземпляра некоторого класса), доступа к переменным-членам, выделения памяти для объектов, динамической проверки типов объектов (как в форме логических выражений, так и в форме предположений *assertions*). Для поддержки более широкого набора оптимизирующих преобразований авторы отказались от единообразной реализации исключений и остановились на параллельном использовании двух реализаций — высокоуровневой (для языков C++, Java) со специальным соглашением вызова функций и низкоуровневой на основе команд дальнего перехода Setjmp, Longjmp.

Виртуальные машины уровня приложений на основе байт-кода, ориентированные на функциональные языки программирования

В отличие от виртуальных машин уровня приложений, ориентированных на императивные языки программирования, машины, ориентированные на функциональные языки программирования, как правило, поддерживают на уровне байт-кода создание замыканий — структур, которые описывают функции со значениями свободных переменных из внешнего контекста. Следует отметить, что в настоящий раздел входит также описание абстрактной машины Уоррена (WAM), предназначенной для языка Prolog. Строго говоря, язык Prolog относится к логическим языкам программирования. Однако определенные особенности, а именно описание замыканий с использованием перманентных переменных, наличие инструкций, сходных с языком сборки для языка Рефал, а также влияние этого представления на машину BEAM языка Erlang, позволяют рассматривать представление WAM в качестве промежуточного представления для функциональных языков программирования.

В работе [7] описана реализация ZINC для языков программирования ML-семейства. Представление поддерживает такую особенность языка ML, как модули. Абстрактное синтаксическое дерево преобразуется в промежуточное представление на основе безтипового лямбда-исчисления, расширенного константами и следующими конструкциями: локальные объявления (обычные и рекурсивные); примитивные операции (например, арифметические); конструкция множественного выбора switch (с отдельной формой для констант примитивных типов); конструкции обработки исключений, последовательного выполнения операций и цикла while. На следующем этапе это представление преобразуется в граф потока исполнения, узлами которого являются инструкции абстрактной машины ZINC с продолжениями. Листьями графа могут являться инструкции stop (завершение предложения программы верхнего уровня), return (конец функции), termapply (хвостовой вызов функции). Проверка типов была отложена автором [7] в качестве дальшейшей работы, поэтому промежуточные представления не хранят информацию о типах.

Примером промышленного промежуточного представления для программ, написанных на динамическом языке программирования, является байт-код эталонной реализации CPython языка Python (официальной опубликованной документации реа-

лизации CPython не существует, основная информация представлена в работе [8], перечень инструкций с описаниями доступен в работе [9]). В качестве особенности байт-кода CPython следует выделить наличие инструкций для импорта функций и переменных из внешних программных модулей, а также расширенных операций со стеком исполнения (копирование двух верхних элементов стека, подъем второго и третьего элементов стека), инструкции для описания генераторов.

С позиций строгого формального описания представлений для функциональных языков программирования, в работе [10] описан процесс получения интерпретатора абстрактной машины KAM на языке Agda для лямбда-исчисления с простыми типами путем логического вывода из набора правил исчисления с явными подстановками. Интерпретатор является корректным по построению, т. е., во-первых, на его вход может быть подан только корректно типизированный терм, и, во-вторых, для любого корректного входного терма процедура его нормализации (вычисление значения) завершается значением, которое имеет такой же тип. Доказательства корректности проходят проверку средствами языка Agda. Автор статьи [10] отмечает, что аналогичным образом может быть получен корректный по построению интерпретатор и для других разновидностей абстрактных машин, выполняющих бета-редукцию.

В работе [11] рассмотрено промежуточное представление (язык сборки) для языка Рефал. Это представление ориентировано на сопоставление с образцом и переписывание термов. Язык сборки представляет собой линейный байт-код, без нетривиального синтаксиса и нетривиального описания потока исполнения. Поток исполнения задается с использованием стека откатов и инструкций, модифицирующих этот стек. Стек откатов представляет собой последовательность обработчиков исключений.

Промежуточное представление BEAM [12] разработано для языка Erlang на основе абстрактной машины Уоррена и машины Janus Virtual Machine. Особенностями виртуальной машины BEAM являются инструкции управления памятью, работы со списками и кортежами, а также, учитывая специфику языка Erlang, инструкции межпроцессного взаимодействия. Инструкции содержат указатели на начало тела той функции, частью которой они являются. Так как Erlang является языком программирования с динамической типизацией, информация о типах хранится в значениях в процессе выполнения программы, а в наборе инструкций содержатся инструкции динамической проверки типов.

Абстрактная машина Уоррена [13], предназначенная для языка Prolog, существенно отличается как от промежуточных представлений императивных и объектно-ориентированных языков программирования, так и от представлений функциональных языков программирования. Основная часть инструкций машины связана с записью и чтением из памяти структур данных языка Prolog (констант, формул логики первого порядка, переменных и списков), унификацией структур и определением процедур поиска с откатами. Представление имеет вид последовательности инструкций, поток исполнения задается в форме откатов, не имеет абстрактного синтаксиса. Как и для других промежуточных представлений языков с динамической типизацией, предусмотрены инструкции для динамической проверки типов.

Внутренние промежуточные представления на основе графа потока исполнения, сохраняющие информацию о типах значений

Представления на основе байт-кодов, описанные в предыдущих двух разделах, в общем случае, без дополнительного динамического контроля, допускают с помощью инструкций перехода по меткам описание потока исполнения, выделение структуры которого является непростой для решения задачей. Представления, описанные в настоящем разделе, определяют поток исполнения в структурированном виде, в форме графа или, в частном случае, дерева.

Представление на основе продолжений (*continuation-passing style, CPS*) является классическим промежуточным представлением, используемым для трансляции программ, написанных с использованием функциональных языков программирования, в машинные коды. В этом представлении поток исполнения описывается путем передачи в функцию дополнительного аргумента-продолжения, который можно считать функциональным аналогом оператора return из императивных языков программирования. Таким образом, функция в представлении CPS возвращает не само значение, а результат применения этого значения к функции-продолжению. В работе [14] Кеннеди сравнивает продолжения с A-нормальной формой, которая является распространенной альтернативой представления CPS, и с языками на основе монад, которые могут считаться развитием A-нормальной формы. A-нормальная форма представляет собой вложенную последовательность определений именованных значений с помощью конструкции вида let <имя> = <значение> in <выражение>. В результате сравнения делается вывод, что с позиций оптимизирующих преобразований программы продолжения значительно удобнее в использовании. Следует отметить, что представление CPS можно считать частным случаем термов лямбда-исчисления, что позволяет сохранять типизацию. Для поддержки обработки исключений авторы работы [14] предлагают использовать типизированные термы с двумя функциями продолжения: для штатного возврата из функции и для обработки исключительных ситуаций. Такая схема может быть адаптирована и для общего случая эффектов.

В качестве примера использования A-нормальных форм на практике следует отметить компилятор TIL языка ML [15]. В этом компиляторе используется последовательность типизированных промежуточных представлений, начиная от представления Lmli на

основе полиморфного лямба-исчисления на верхнем уровне, через представление Bform, которое представляет собой А-нормальную форму, и заканчивая представлением Lmli-Closure, которое содержит конструкции для явного выделения замыканий. На последующих этапах используется бестиповое промежуточное представление.

Промежуточное представление Thorin [16] предназначено для описания программ, в которых используются функции высшего порядка. По сравнению с представлением SSA, Thorin допускает более компактное представление кода таких программ, а по сравнению с представлением CPS — дополнительный класс оптимизаций под названием *lambda mangling* (декорирование лямбда-термов). Как и в CPS, последовательность вычислений описывается в виде вызовов функций-продолжений. Имена функций и переменных используются в качестве вершин ориентированного графа, направленными ребрами которого, в свою очередь, связаны место использования и определение имени. Такая структура, как и в случае с индексами де Брёйна, позволяет не хранить имена в явном виде и исключить тем самым необходимость коррекции термов при подстановке. При этом, в отличие от индексов де Брёйна, графовое представление допускает ссылки как на ранее определенные имена, так и на имена, определяемые далее по коду, что позволяет задавать взаимно-рекурсивные функции в явном виде.

В недавно вышедшей работе [17] предложено промежуточное представление на основе исчисления секвенций, дополненного лямбда-термами. Одним из итогов исследования, представленного в этой статье, является установление непосредственной связи между промежуточным представлением CPS и предлагаемым авторами работы [17] промежуточным представлением. При этом секвенции описывают продолжения в явном виде, отделенном от обычных лямбда-термов. Такое разделение целесообразно, поскольку позволяет отделить места "служебного" использования продолжений для описания потока исполнения от использования продолжений непосредственно в коде программы. В программах на функциональных языках программирования такая ситуация возникает достаточно часто. Частным случаем последнего являются функции обратного вызова при асинхронном потоке исполнения.

Промежуточное представление среднего уровня абстракции (MIR) в эталонной реализации языка программирования Rust было предложено в работе [18]. Одной из задач, для решения которой было разработано представление, было упрощение реализации проверки типов с возможностью последующего математического обоснования безопасности типов. Представление является графовым, направленным на описание тела отдельной функции. Граф потока исполнения составлен из базовых блоков, которые завершаются одной из специальных инструкций-терминаторов (goto, panic, if, switch, call, diverge, return). Кроме того, имеются "инструкции"-конструкторы сложных выражений (структур, кортежей и массивов). Информация о типах сохраняется, однако в настоящее время не используется при проверке типов. Существенным отличием от SSA-подобных представлений является наличие типов-ссылок. Статический контроль за корректностью передачи и использования ссылок является основной особенностью языка Rust. В явном виде (в виде инструкций-терминаторов или выделенного вида предложений) описываются порождение исключений и момент выхода выражения за пределы области видимости (*drop*). Авторы спецификации отмечают, что некоторые аспекты представления (например, проверка на выход за границы массива по месту обращения) в настоящее время не подлежат статической проверке типов, что ограничивает набор свойств, которые может гарантировать система типов на основе такого представления. После обработки на среднем уровне абстракции, представление MIR транслируется в набор инструкций LLVM.

Высокоуровневые, специфичные для языка программирования промежуточные представления на основе канонических форм синтаксического дерева

Последний класс промежуточных представлений в классификации, предлагаемой авторами настоящей статьи, в значительной степени пересекается с понятием языка программирования и отличается от него единственным аспектом, который относится скорее к прагматике языка. А именно, ряд синтаксических конструкций языка, семантика которых может быть выражена через более простые синтаксические конструкции, заменяются на такие более простые конструкции. Для обозначения удаляемых таким образом синтаксических конструкций используются термины "синтаксический сахар" (*syntactic sugar*) или, в других случаях, "макрос", а удаление таких конструкций называется "раскрытием сахара" (*desugaring*) или "раскрытием макросов" соответственно.

Классическая работа [19] представляет машину STG (*Spineless Tagless G-machine*), которая предназначена для описания программ, использующих нестрогий порядок вычислений (ленивые вычисления). Это промежуточное представление было разработано для реализации языка программирования Haskell. Такое представление можно отнести к классу промежуточных, имеющих нетривиальный абстрактный синтаксис, который в целом аналогичен подмножеству исходного языка Haskell. Следует выделить следующие его особенности:

- спецификация замыкания (перечисление используемых свободных переменных) в определении функции;
- операторы последовательного и рекурсивного определения (let, letrec);
- оператор разбора альтернатив case (среди литералов или по конструкторам алгебраических типов данных);
- работа как с упакованными (находящимися в динамической памяти), так и с неупакованными (передаваемыми по значению) данными.

Промежуточное представление CIL [20] предназначено для анализа, преобразования и последующего восстановления исходного кода на языке программирования C. Оно относится к высокоуровневым промежуточным предоставлениям с нетривиальным абстрактным синтаксисом. В значительной степени представление CIL повторяет основные конструкции языка C (и даже поддерживает такие расширения компиляторов, как атрибуты). Авторы уделяют внимание возможности отображения конструкций промежуточного представления на строки исходной программы. Однако ряд конструкций ("синтаксический сахар") раскрывается в более простые команды. К таким конструкциям относятся циклы (типично для промежуточных представлений императивных языков) и левые части операторов присваивания (*lvalue*).

Язык промежуточного представления данных IDL (*Interface Descriptor Language*) предназначен для описания структур данных и их свойств в форме базовой спецификации структур данных. Эта спецификация задает: состав полей данных и общую схему их взаимодействия; последовательности уточнений (*refinements*), которые определяют конкретные типы данных и их свойства. На представлении IDL основано представление DIANA (*Descriptive Intermediate Attributed Notation for Ada*) программ на языке Ada [21]. Это представление описывает результаты лексического анализа, синтаксического анализа и анализа статической семантики. Основной абстрактной моделью, которой оперирует представление DIANA, являются абстрактные синтаксические деревья и атрибутные деревья, дополняющие узлы синтаксического дерева результатами синтаксического и семантического анализа. Атрибуты могут содержать ссылки на другие узлы атрибутного дерева. Представление DIANA может быть транслировано обратно в исходный текст программы на языке Ada. Однако в отличие от остальных промежуточных представлений, рассматриваемых в настоящем разделе, оно не имеет конкретного синтаксиса.

Представление Henk [22] основано на термах чистых систем типов (PTS), описанных Хэнком Барендрегтом в виде лямбда-куба. Такой выбор базовой модели допускает гибкую настройку выразительных возможностей систем типов с помощью включения тех или иных правил лямбда-куба. Представление Henk основано на лямбда-исчислении второго порядка. Авторы предпочли не вводить правило, допускающее зависимые типы, так как это существенно осложнило бы доказательство завершимости проверки типов. В рамках постановки задачи о разработке промежуточного представления авторы упоминают необходимость сохранения информации о типах на всех этапах трансляции, в частности, в промежуточном представлении. Представление Henk имеет нетривиальный синтаксис. Его операционная семантика в явном виде не задана. Объявления типов отделены от кода функций. Авторы отмечают практическую необходимость в конкретной синтаксической записи представления (с сокращениями и "сахаром") и в многоместных абстракциях и перечислениях (суммах). Верхний уровень представления взаимно рекурсивен, т. е. возможно описание рекурсивных функций в явном виде.

В верифицированном компиляторе языка C CompCert используется последовательность промежуточных представлений [23], каждое из которых имеет заданную формальную семантику, описанную в терминах исчисления конструкций в среде Coq. Входным для компилятора является представление Clight, аналогичное абстрактному синтаксису языка C с раскрытием сложных синтаксических конструкций ("синтаксического сахара"), без побочных эффектов внутри приложений и без объявлений переменных на уровне блоков. Первые две стадии компиляции переводят представление Clight в представление C#minor. В последнем удаляется перегрузка арифметических операторов, а также приводятся к единому виду циклы. Затем выполняется трансляция в представление Cminor, удаляющее оператор получения указателя (&) и заменяющее локальные переменные, адреса которых используются в теле функции, на области памяти, выделенные на стеке. На следующем шаге выполняется специфичная для целевой аппаратной платформы подстановка специализированных арифметических инструкций процессора (представление CminorSel). После этого код транслируется в граф потока управления, узлы которого содержат наборы инструкций в представлении RTL (*Register transfer language*). На нескольких последующих стадиях на основе графа потока исполнения происходит генерация машинного кода, однако эти вопросы выходят за пределы настоящего обзора. Следует отметить, что для каждой стадии компиляции в среде Coq получено доказательство сохранения семантики на этой стадии с использованием формальной семантики исходного и целевого промежуточных представлений.

Исходным языком для средства формальной верификации Boogie [24] является язык Spec# — расширение языка C# конструкциями для описания инвариантов, пред- и постусловий, а также их динамическими проверками. Средство Boogie на уровне промежуточного представления CIL распознает такие динамические проверки и выполняет статическую верификацию на предмет выполнения таких условий. Представление CIL транслируется во внутреннее представление BoogiePL, которое для верификации преобразуется в последовательность предложений логики первого порядка. Представление BoogiePL имеет нетривиальный синтаксис и состоит из "теории" — набора объявлений типов, имен объектов и логических аксиом, которые в совокупности определяют семантику исходного языка, — и императивной части, в которой содержатся объявления переменных, объявления процедур и реализации процедур. По сравнению с обычным императивным языком программирования представление BoogiePL содержит специальные команды assert (условие, которое должно быть верифицировано, при невыполнении порождается ошибка), assume (условие, которое

подразумевается в качестве данного для процесса верификации, а при невыполнении которого считается, что дальнейшее доказательство корректности процедуры можно получить по принципу ex falso) и команды havoc (присвоить переменной произвольное корректно типизированное значение, удовлетворяющее ограничениям, которые задаются опционально при объявлении переменной). Контроль потока исполнения в представлении ограничивается командой недетерминированного перехода на один из базовых блоков, перечисленных в аргументах команды. Это позволяет, с одной стороны, генерировать условия верификации для всех возможных путей исполнения тела процедуры, но с другой стороны, ограничивает возможность по выводу проверок по потоку исполнения в ходе выполнения процедуры (вместо этого используется команда assume). Набор базовых блоков представления CIL преобразуется в плоское абстрактное синтаксическое дерево, в котором побочные эффекты вычисления аргументов задаются как отдельные слоты стека вычислений. Стек вычисления является обобщением понятия стека исполнения CIL: на стек допускается помещение выражений (а не только значений).

Заключение

В настоящей статье представлена классификация методов промежуточного представления программ с позиций их формальной спецификации. Для каждого из классов приведены примеры, в общей сложности 22 промежуточных представления, которые позволяют охарактеризовать современное состояние исследований на этом направлении.

Если упорядочить рассмотренные промежуточные представления в хронологическом порядке (заметим, что в настоящей статье могут использоваться ссылки на более поздние источники), можно выделить следующие изменения в задачах, которые ставятся перед промежуточным представлением.

1. Ранние промежуточные представления ([4, 6, 7, 11—14, 19, 21], а также классические представления — SSA и трехадресный код) используются, в первую очередь, в качестве фиксированной структуры данных, другими словами, интерфейса между различными стадиями трансляции программы. Интерфейс промежуточных представлений оказался удобен также для декомпозиции трансляторов и при создании внешних средств анализа и трансформации кода (например, оптимизаторов): подобные этапы являются последовательностью отображений промежуточного представления в себя, в общем случае частичных отображений (например, стадии анализа могут прерывать процесс трансляции с ошибкой). Кроме того, естественным образом такие отображения использовались при реализации взаимодействия между фрагментами кода программы, написанными с использованием нескольких разных языков программирования ([4, 6], в настоящее время — [12] в реализации языка Elixir).

2. В более поздних промежуточных представлениях ([2, 15—18, 20, 22, 23], в меньшей степени — в упомянутом в предыдущем пункте [4]) большее внимание уделяется системе типов. В дополнение к роли интерфейса, такие промежуточные представления используются для упрощения реализации процедур проверки типов.

3. Отдельно следует отметить промежуточные представления, на которых реализуются процедуры формальной верификации программ (в настоящем обзоре к таким представлениям относятся представления, описанные в работах [1, 10, 24] и упомянутые ранее [2, 23]).

Промежуточные представления, направленные на решение задач практического характера, по сравнению с результатами академических исследований, как правило, имеют большое количество примитивов (инструкций, команд), многие из которых специфичны для задач, на решение которых нацелено то или иное промежуточное представление. Достаточно большой вклад в число команд вносят константы примитивных типов, операции над ними и определяющие их отношения, такие как, например, равенство и порядок на типе целочисленной арифметики аппаратной платформы.

Следует отметить также определенное сходство между промежуточными представлениями объектно-ориентированных языков программирования (например, [4, 6]) и промежуточными представлениями языков с динамической типизацией [8, 12]: и в том и в другом случае в набор инструкций представления входят инструкции динамической (в процессе выполнения) проверки типов.

С позиций математического описания промежуточных представлений следует обратить внимание на тот факт, что исполнители проекта Vellvm разработали несколько крупношаговых (менее детализированных) моделей семантики для того, чтобы на этих моделях можно было обосновать ряд преобразований, проводимых компилятором в рамках оптимизации. Этот факт показывает, что для адекватного описания функциональных свойств программ их промежуточное представление должно иметь более высокий уровень абстракции от среды исполнения, допускающий определенную свободу реализации. Мелкошаговая операционная семантика существенно ограничивает набор преобразований, сохраняющих семантику.

Систематический обзор работ по построению промежуточных представлений программ представлен в статье [25]. Отчет [26] содержит более детальное по сравнению с настоящей работой сравнение типизированных промежуточных представлений, но содержит только три представления, а именно: виртуальную машину Java, во многом эквивалентную представлению CLI, вошедшему в настоящий обзор; типизированный язык ассемблера TAL, упоминаемый и в настоящем обзоре; код, несущий доказательство, развитием которого является упомянутый выше фундаментальный код, несущий доказательство.

Языки LISP и Forth, которые также традиционно используются в качестве промежуточных представлений, выходят за рамки настоящего обзора, однако

их можно было бы охарактеризовать как представление класса "высокоуровневое представление с нетривиальным синтаксисом на основе канонических форм синтаксического дерева" (LISP) и виртуальную машину уровня байт-кода, ориентированные на императивные языки программирования (Forth).

Представленные в настоящей статье результаты анализа различных технических решений, использованных в известных промежуточных представлениях, могут быть применены при разработке новых языков программирования и промежуточных представлений.

Работа выполнена при поддержке РФФИ, проект № 16-07-01178а.

Список литературы

1. **Zhao J., Nagarakatte S., Martin M. M. K., Zdancewic S.** Formalizing the LLVM Intermediate Representation for Verified Program Transformations // Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, NY, USA: ACM, 2012. P. 427—440.

2. **Morrisett G., Walker D., Crary K., Glew N.** From System F to Typed Assembly Language // ACM Trans. Program. Lang. Syst. 1999. Vol. 21, N 3. P. 527—568.

3. **Appel A.** Foundational proof-carrying code // Proceedings 16th Annual IEEE Symposium on Logic in Computer Science. IEEE Comput. Soc., 2001. P. 247—256.

4. **ISO** ISO/IEC 23271:2003: Information technology — Common Language Infrastructure. Geneva, Switzerland: International Organization for Standardization, 2005. xi + 99 (Part. I), ix + 164 (Part. II), vi + 125 (Part. III), iii + 16 (Part. IV), iv + 79 (Part. V) c.

5. **Васенин В. А., Кривчиков М. А.** Статическая семантика стандарта ECMA-335 // Программирование. 2012. № 4. С. 3—16.

6. **Dean J., DeFouw G., Grove D., Litvinov V., Chambers C.** Vortex: An Optimizing Compiler for Object-oriented Languages // Proceedings of the 11th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. New York, NY, USA: ACM, 1996. P. 83—100.

7. **Leroy X.** The ZINC experiment: an economical implementation of the ML language: Technical report. INRIA, 1990.

8. **Portnoy A., Santiago A.-R.** Reverse Engineering Python Applications // Proceedings of the 2Nd Conference on USENIX Workshop on Offensive Technologies. Berkeley, CA, USA: USENIX Association, 2008. P. 6:1—6:5.

9. **Van der Laan W. J.** Python Bytecode Archeology. 2011. URL: https://laanwj.github.io/2011/5/4/python-bytecode-archeology (дата обращения 01.02.2017).

10. **Swierstra W.** From Mathematics to Abstract Machine: A formal derivation of an executable Krivine machine // Electronic Proceedings in Theoretical Computer Science. 2012. Vol. 76. P. 163—177.

11. **Романенко С. А**. Машинно независимый компилятор с языка рекурсивных функций: дис. ... канд. физ.-мат. наук. М.: ИПМ АН СССР, 1978. 148 с.

12. **Hausman B.** Turbo Erlang: Approaching the Speed of C // Implementations of Logic Programming Systems / Eds. E. Tick, G. Succi. Springer US, 1994. P. 119—135.

13. **Aït-Kaci H.** Warren's Abstract Machine: A Tutorial Reconstruction. Cambridge, Mass: The MIT Press, 1991. 114 p.

14. **Kennedy A.** Compiling with continuations, continued // ACM SIGPLAN Notices. 2007. Vol. 42, N 9. P. 177—190.

15. **Tarditi D., Morrisett G., Cheng P. et al**. TIL: A Type-directed Optimizing Compiler for ML // Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 1996. P. 181—192.

16. **Leißa R., Köster M., Hack S.** A graph-based higher-order intermediate representation // 2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). 2015. P. 202—212.

17. **Downen P., Maurer L., Ariola Z. M., Jones S. P.** Sequent Calculus As a Compiler Intermediate Language // Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming. New York, NY, USA: ACM, 2016. P. 74—88.

18. **Matsakis N.** *rust-lang/rfcs/1211-mir.md. GitHub. 2016. URL: https://github.com/rust-lang/rfcs

19. **Jones S. P.** Implementing Lazy Functional Languages on Stock Hardware: The Spineless Tagless G-machine // Journal of Functional Programming. 1992. Vol. 2, P. 127—202.

20. **Necula G. C., McPeak S., Rahul S. P., Weimer W** CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs // Compiler Construction. Berlin, Heidelberg, Springer, 2002. P. 213—228.

21. **DIANA** An Intermediate Language for Ada / Eds. G. Goos et al. Berlin, Heidelberg: Springer, 1983.

22. **Jones S. P., Meijer E.** Henk: A Typed Intermediate Language // In Proc. First Int'l Workshop on Types in Compilation. 1997. URL: http://rnyingma.synrc.com/publications/cat/Lambda%20Calculus/Henk.pdf

23. **Leroy X**. Formal Verification of a Realistic Compiler // Communications of the ACM. 2009. Vol. 52, N 7. P. 107—115.

24. **Barnett M., Chang B. E., DeLine R. et al.** Boogie: A Modular Reusable Verifier for Object-Oriented Programs // Formal Methods for Components and Objects. Berlin, Heidelberg, Springer, 2005. P. 364—387.

25. **Diehl S., Hartel P., Sestoft P.** Abstract machines for programming language implementation // Future Generation Computer Systems. 2000. Vol. 16, N 7. P. 739—751.

26. **Tse S.** Typed Intermediate Languages: Technical Report MS-CIS-04-17. University of Pennsylvania Department of Computer and Informational Scinece, 2004.

Program Intermediate Representation Techniques

V. A. Vasenin, vasenin@msu.ru, **M. A. Krivchikov,** maxim.krivchikov@gmail.com, Lomonosov Moscow State University, Moscow, 119234, Russian Federation

Corresponding author:

Krivchikov Maxim A., Senior Researcher, Lomonosov Moscow State University, Moscow, 119234, Russian Federation,
E-mail: maxim.krivchikov@gmail.com

Received on April 21, 2017
Accepted on May 30, 2017

The concept of an intermediate representation of a program originates from software translation theory. In the present paper the classification of high-level intermediate representations is proposed, together with the review of current state-of-the-art research in intermediate representations. The paper presents a part of a research project aimed at development of intermediate representation for high-level specification of formal semantics for domain-

specific programming languages. The review is mostly concerned with mathematical representation of intermediate representations. The five proposed intermediate representation classes are:

- *low-level intermediate representations used mostly in compilers;*
- *bytecode-based application virtual machines for imperative programming languages;*
- *bytecode-based application virtual machines for functional programming langauges;*
- *internal graph-based intermediate representations;*
- *high-level, programming language-specific intermediate representations based on canonical forms for the AST.*

The review contains 22 intermediate representations. Looking at them in chronological order we can observe the following shift in problem statements. Earlier intermediate represntations ([4, 6, 7, 11—14, 19, 21] and classical examples of SSA, RTL and three-address code) are used primarily as a fixed data structure, in other words, an interface between different stages of compilation pipeline. Recent intermediate representations ([2, 15—18, 20, 22, 23] and, to lesser degree, [4], mentioned previously) are more concerned with type system. These representations are used to simplify implementation of the typechecker. In addition, we shall mention intermediate representations aimed at formal verification procedures ([1, 10, 24] and [22, 23] from previous group).

Industrial intermediate representations (in comparison with academic research) usually have more primitive instructions or commands. The increase is mostly due to primitive type constants, operations and relations. We shall also note the similarity between intermediate representations for the object-oriented programming languages (e.g. [4, 6]) and dynamically-typed programming languages ([8, 12]): in both cases instructions of the dynamical (run-time) type checking are present.

Related work includes the article [25], which contains systematic review of intermediate representations, and report [26] with rather detailed comparison of type system expressiveness for three typed intermediate representations.

Keywords: *programming languages, intermediate representation, program translation, formal semantics, software engineering, review, classification*

For citation:

Vasenin V. A., Krivchikov M. A. Program Intermediate Representation Techniques, *Programmnaya Ingeneria*, 2017, vol. 8, no. 8, pp. 345—353.

DOI: 10.17587/prin.8.345-353

References

1. **Zhao J., Nagarakatte S., Martin M. M. K., Zdancewic S.** Formalizing the LLVM Intermediate Representation for Verified Program Transformations, *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2012, pp. 427—440, doi:10.1145/2103656.2103709.

2. **Morrisett G., Walker D., Crary K., Glew N.** From System F to Typed Assembly Language, *ACM Trans. Program. Lang. Syst.,* 1999, vol. 21, pp. 527—568.

3. **Appel A. W.** Foundational proof-carrying code, *In Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*, 2001, pp. 247—256, doi:10.1109/LICS.2001.932501.

4. **ISO.** ISO/IEC 23271:2003: Information technology — Common Language Infrastructure, 2005.

5. **Vasenin V. A., Krivchikov M. A.** Staticheskaja semantika standarta ECMA-335 (ECMA-335 Static Formal Semantics), *Programming and Computer Software*, 2012, vol. 38, no. 4, pp. 183—188 (in Russian).

6. **Dean J., DeFouw G., Grove D., Litvinov V., Chambers C.** Vortex: An Optimizing Compiler for Object-oriented Languages, *Proceedings of the 11th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, 1996, pp. 83—100, doi:10.1145/236337.236344.

7. **Leroy X.** The ZINC experiment: an economical implementation of the ML language. Technical report, INRIA, 1990.

8. **Portnoy A., Santiago A.-R.** Reverse Engineering Python Applications, *Proceedings of the 2nd Conference on USENIX Workshop on Offensive Technologies,* 2000, vol. 6, pp. 1—5.

9. **Van der Laan W. J.** *Python Bytecode Archeology*, 2011, available at: https://laanwj.github.io/2011/5/4/python-bytecode-archeology.

10. **Swierstra W.** From Mathematics to Abstract Machine: A formal derivation of an executable Krivine machine, *Electronic Proceedings in Theoretical Computer Science*, 2012, vol. 76, pp. 163—177.

11. **Romanenko S. A.** *Mashinno nezavisimii kompilyator s yazika rekursivnikh funkcii* (Machine-independent compiler for recursive functions language), PhD thesis (Keldysh Institute of Applied Mathematics), 1978, 148 p. (in Russian).

12. **Hausman B.** Turbo Erlang: Approaching the Speed of C, *Implementations of Logic Programming Systems* / Eds. E. Tick, G. Succi. Springer US, 1994, pp. 119—135.

13. **Aït-Kaci H.** *Warren's Abstract Machine: A Tutorial Reconstruction,* Cambridge, Mass, The MIT Press, 1991, 114 p.

14. **Kennedy A.** Compiling with continuations, continued, *ACM SIGPLAN Notices*, 2007, vol. 42, no. 9, pp. 177—190.

15. **Tarditi D., Morrisett G., Cheng P., Stone C., Harper R., Lee P.** TIL: A Type-Directed Optimizing Compiler for ML. *Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, 1996, pp. 181—192, doi:10.1145/231379.231414.

16. **Leissa R., Koster M., Hack S.** A graph-based higher-order intermediate representation, *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO),* 2015, pp. 202—212.

17. **Downen P., Maurer L., Ariola Z. M., Jones S. P.** Sequent Calculus As a Compiler Intermediate Language, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016, pp. 74—88, doi:10.1145/2951913.2951931.

18. **Matsakis N.** *rust-lang/rfcs/1211-mir.md, 2016, available at: https://github.com/rust-lang/rfcs

19. **Jones S. P.** Implementing Lazy Functional Languages on Stock Hardware: The Spineless Tagless G-machine, *Journal of Functional Programming,* 1992, vol. 2, pp. 127—202.

20. **Necula G. C., McPeak S., Rahul S. P., Weimer W.** CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs, *Compiler Construction,* 2002, pp. 213—228, doi:10.1007/3-540-45937-5_16.

21. **DIANA** An Intermediate Language for Ada. / Eds. **G. Goos,** Berlin, Heidelberg: Springer Berlin Heidelberg, 1983.

22. **Jones S. P., Meijer E.** Henk: A Typed Intermediate Language, *Proc. First Int'l Workshop on Types in Compilation,* 1997, available at: http://rnyingma.synrc.com/publications/cat/Lambda%20Calculus/Henk.pdf

23. **Leroy X.** Formal Verification of a Realistic Compiler, *Communications of the ACM,* 2009, vol. 52, no. 7, pp. 107—115.

24. **Barnett M., Chang B. E., DeLine R., Jacobs B., Leino K. R. M.** Boogie: A Modular Reusable Verifier for Object-Oriented Programs, *Formal Methods for Components and Objects,* 2005, pp. 364—387, doi:10.1007/11804192_17.

25. **Diehl S., Hartel P., Sestoft P.** Abstract machines for programming language implementation, *Future Generation Computer Systems,* 2000, vol. 16, no. 7, pp. 739—751.

26. **Tse S.** Typed Intermediate Languages: Technical Report MS-CIS-04-17, University of Pennsylvania Department of Computer and Informational Science, 2004.

УДК 004.921 DOI: 10.17587/prin.8.354-358

М. В. Михайлюк, д-р физ.-мат. наук, зав. отделом, e-mail: mix@niisi.ras.ru,
А. М. Трушин, науч. сотр., ФГУ "ФНЦ Научно-исследовательский институт системных исследований РАН", Москва

Алгоритмы определения коллизий сфер на GPU

Рассмотрена задача определения попарных коллизий сфер между собой и сфер с капсулами в системах расчета динамики виртуальных объектов. Предложены алгоритмы решения этих задач на GPU (Graphics Processing Unit) и проведено сравнение времени расчетов с реализацией на CPU (Central Processing Unit).

Ключевые слова: *виртуальные объекты, коллизии, сфера, капсула, GPU, CUDA, параллельные вычисления, системы виртуального окружения*

Введение

Одной из важных задач при расчете динамики системы твердых тел является моделирование их столкновений (коллизий). Так как коллизии объектов произвольной формы определить достаточно сложно, общепринятым подходом является использование ограничивающих контейнеров простой формы (сфер, капсул, прямоугольных параллелепипедов и т. д.). Каждый объект окружается одним или несколькими ограничивающими контейнерами, и коллизия объектов сводится к определению коллизий пар этих контейнеров. Если имеется N пар контейнеров, то в общем случае требуется провести около N^2 проверок (т. е. каждый с каждым). Для уменьшения этого числа задачу разбивают на две фазы: широкую и узкую [1]. В широкой фазе выполняется грубый (но быстрый) отбор тех пар, которые потенциально могут пересекаться. Таким образом, число пар, необходимых для дальнейшего анализа, может сильно уменьшиться. В узкой фазе проводится точное определение коллизии и вычисление ее параметров для отобранных пар.

Тем не менее обработка коллизий даже в узкой фазе для систем динамики, содержащих тысячи и миллионы объектов, является задачей, не реализуемой на CPU в масштабе реального времени. Одним из выходов является решение этой задачи с помощью параллельных алгоритмов на процессорах графической карты персонального компьютера с использованием архитектуры CUDA (*Compute Unified Device Architecture*).

Некоторые коммерческие системы моделирования динамики (например, PhysX) пошли по этому пути, однако используемые в них алгоритмы являются коммерческой тайной и недоступны научной общественности. В работе [2] для решения задачи коллизий тел сложной формы предложено приближать их поверхность множеством маленьких сфер, а затем находить попарное пересечение этих сфер на GPU. Таким образом, задача определения коллизий сфер является актуальной.

В работах [3, 4] описаны алгоритмы определения коллизий сфер с контейнерами других типов. В настоящей работе рассматривается задача определения коллизий N пар сфер и N пар сфера/капсула с использованием архитектуры CUDA. В разд. 1 рассмотрена общая технология работы с архитектурой CUDA, в разд. 2 приведено описание алгоритма определения коллизий сфер между собой, а в разд. 3 — коллизий сфер с капсулами. Также представлены результаты расчетов для разных значений N.

1. Технология работы с архитектурой CUDA

Опишем кратко архитектуру CUDA и особенности технологии программирования с ее помощью [5, 6]. В графической карте имеется N процессоров, каждый из которых имеет свою регистровую (быструю) память и доступ (достаточно медленный) к глобальной памяти видеокарты (*global memory*). Все процессоры разбиты на группы по M штук, каждая из которых образует мультипроцессор, имеющий разделяемую память (*shared memory*), константный и текстурный кэши (рис. 1). Логическая структура задачи описывается в виде массива (сетки) нитей (потоков), разбитого на блоки, содержащие одинаковое число нитей. Эта структура может быть одномерной, двумерной или трехмерной в зависимости от типа задачи. На рис. 2 показан пример двумерной сетки размера $m \times n$ блоков, каждый из которых может обрабатывать $(s + 1) \times (k + 1)$ потоков. Потоки разбиваются на группы по 32 потока, называемые варпами (*warps*). Только нити в пределах одного варпа выполняются физически одновременно. Один или несколько (например, четыре) блоков выполняются на одном мультипроцессоре. Любой обмен информацией между потоками возможен только внутри блока через разделяемую память.

Для работы с архитектурой CUDA необходимо создать функцию-ядро, которая обычно запускается из контекста CPU и параллельно, в синхронном режиме, выполняется множеством потоков на GPU.

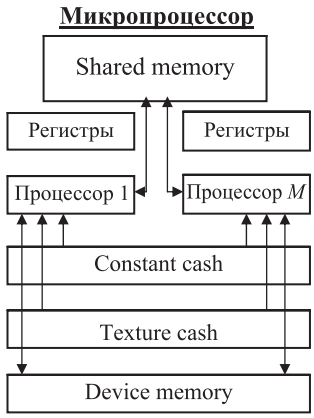

Рис. 1. Схема мультипроцессора

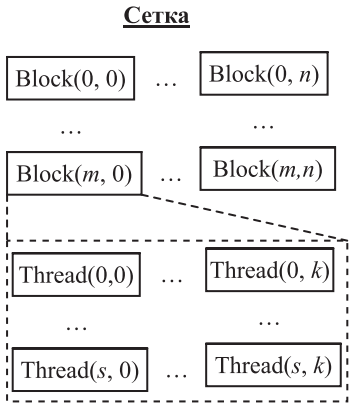

Рис. 2. Схема сетки (*grid*)

Каждый поток узнает свой индекс, что позволяет ему определить области предназначенных для него обрабатываемых данных. Программа запуска, в которой задаются размеры массивов потоков и блоков, выделяет нужные объемы памяти на GPU, копирует в них исходные данные, запускает функцию-ядро и копирует в память CPU полученные результаты по окончании счета. Функция-ядро, таким образом, будет выполняться параллельно на всех графических ядрах CUDA, но с разными исходными данными. Необходимо отметить особый способ обработки в GPU условных операторов типа if *flag*, то *A*, иначе *B*. Независимо от значения условия *flag* на процессоре будут выполнены оба действия, т. е. и *A*, и *B*, а затем выбран нужный результат. Поэтому для повышения эффективности вычислений надо избегать условных операторов или маскировать их.

Очень важным аспектом при работе с архитектурой CUDA является способ обращения к глобальной памяти. Для эффективной работы адрес обращения к ней должен быть выравнен по границе слова (т. е. кратен 4 байтам). Так как все обращения мультипроцессора к памяти происходят независимо для каждой половины варпа, то следует располагать данные для 16 варпов последовательно и считывать их за один шаг. Кроме того, с точки зрения объединения запросов к памяти эффективнее использование не массивов структур, а структуры массивов.

Предложенные в данной работе алгоритмы выполнялись на персональном компьютере под управлением операционной системы Windows с графической картой GTX750 архитектуры Maxwell, имеющей 512 процессоров, разделенных на четыре мультипроцессора. В каждом мультипроцессоре содержится 65536 регистров (т. е. 512 регистров в одном процессоре). Для N пар объектов число блоков в сетке было выбрано равным $[(N + 128 - 1)/128]$, а число нитей в блоке — 128. Следуя устоявшейся терминологии, CPU будем называть хостом.

2. Алгоритм пересечения сфер между собой

Пусть имеется N пар сфер, для которых надо определить их пересечение и, в случае если они пересекаются, получить информацию о пересечении (точка, нормаль и глубина). Каждая пара задается центрами сфер P_1 и P_2 в мировой системе координат и радиусами r_1 и r_2. Для каждой пары алгоритм возвращает флаг пересечения *isColl* (равен 1, если сферы пересекаются и 0 в противном случае), точку контакта P, нормаль **n** и глубину проникновения d (рис. 3).

Для обеспечения эффективного доступа процессоров (нитей варпа) к глобальной памяти данные записывают в виде следующих структур массивов:

```
struct InputSpheresData
{
  float r1[N];                          // радиусы первых сфер
  float r2[N];                          // радиусы вторых сфер
  float P1x[N], P1y[N], P1z[N];         // центры первых сфер
  float P2x[N], P2y[N], P2z[N];         // центры вторых сфер
}
и
struct OutputSpheresData
{
  bool isColl [N];                      // флаги пересечений сфер
  float nx[N], ny[N], nz[N];            // нормали сфер
  float Px[N], Py[N], Pz[N];            // точки пересечений сфер
  float d[N];                           // глубины проникновений
}
```

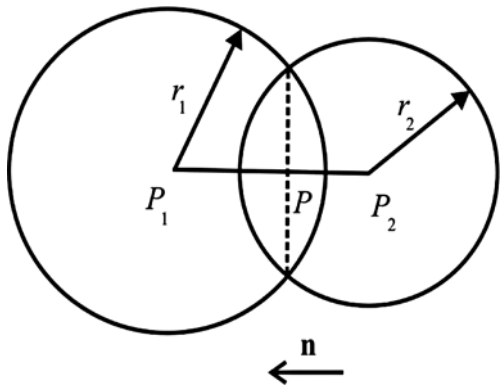

Рис. 3. Пересечение двух сфер

Алгоритм определения пересечения сфер на GPU

1. Подготовка на хосте структуры данных для N пар сфер.

2. Копирование входных данных InputSpheresData с хоста на GPU.

3. Выполнение на GPU функции-ядра, определяющей пересечения N пар сфер и вычисляющей нормали, точки и глубины для пересекающихся пар.

4. Копирование выходных данных OutputSpheresData с GPU на хост.

Для оптимизации передачи данных между хостом и GPU структуры данных записываются в закрепленную память (*pinned-память* [5]) общего виртуального пространства, что позволяет копировать данные между CPU и GPU напрямую без копирования в хост-буфер.

Функция-ядро

1. Вычисление номера нити $tdx = ThreadIdx.x + blockIdx.x \cdot blockDim.x$.

2. Определение вектора $\mathbf{d} = (d_x, d_y, d_z)^{\mathrm{T}}$, соединяющего центры сфер:

$$\mathbf{d} = P_1[tdx] - P_2[tdx].$$

3. Вычисление расстояния между сферами $d = \sqrt{d_x^2 + d_y^2 + d_z^2}$ и суммы радиусов сфер: $r_s = r_1[tdx] + r_2[tdx]$.

4. Вычисление флага пересечения: $isColl = (d \leqslant r_s)$.

5. Если $isColl = 1$, то

5.1. Проверка на совпадение центров сфер: $isNull = (d \leqslant \varepsilon)$.

5.2. Вычисление величины, обратной к d:

$$d' = \varepsilon isNull + d(1 - isNull);\quad d^{-1} = (1 - isNull)/d'.$$

5.3. Вычисление глубины проникновения:

$$deep[tdx] = r_s - d.$$

5.4. Вычисление координат нормали:

$$\mathbf{n}[tdx] = (isNull, 0, 0)^{\mathrm{T}} + d^{-1}\mathbf{d}.$$

5.5. Вычисление смещения:

$$\Delta p = \frac{1}{2}(1 - isNull)\left(r_2[tdx] - r_1[tdx] - d\right).$$

5.6. Вычисление точки контакта:

$$P[tdx] = P_1[tdx] + \Delta p\mathbf{n}[tdx].$$

Здесь $ThreadIdx.x$, $blockIdx.x$, $blockDim.x$ — встроенные переменные CUDA-ядра, а $\varepsilon = 10^{-6}$. В пп. 5.1 этой функции $isNull = 1$, если центры сфер совпадают. В этом случае $d' = \varepsilon$, $d^{-1} = 0$ и нормаль коллизии равна $\mathbf{n}[tdx] = (1, 0, 0)^{\mathrm{T}}$, а точка контакта равна $P_1[tdx]$. Если же $isNull = 0$, то нормаль равна нормализованному вектору $\mathbf{d}$ (см. пп. 5.4). Фактически формулы пп. 5.2 заменяют условный оператор if с целью избежать нежелательного ветвления потоков.

В табл. 1 приведены данные апробации этого алгоритма на компьютере указанной выше конфигурации. Из данных табл. 1 видно (столбцы 2 и 3), что время выполнения функции-ядра на GPU в 50...100 раз меньше, чем на CPU. Однако копирование данных в глобальную память и обратно занимает довольно большое время, и выигрыш получается лишь для числа пар, начиная с 17 тыс.

Таблица 1

N, пары	Время на CPU, мс	Время выполнения ядра, мс	Время выполнения с учетом копирования данных, мс
10 000	0,62	0,017	0,93
17 000	1,11	0,022	1,11
100 000	9,97	0,091	4,3
1 000 000	85,25	0,902	38,44

3. Алгоритм пересечения сферы и капсулы

Пусть имеется N пар сфера-капсула, для которых надо определить их пересечение. Каждая пара задается центрами P_1 и P_2, радиусами r_1 и r_2 и половиной L длины отрезка капсулы (рис. 4). Пусть $\mathbf{v} = (v_x, v_y, v_z)^{\mathrm{T}}$ — единичный вектор вдоль оси капсулы. Для каждой пары алгоритм возвращает флаг пересечения $isColl$, точку контакта P, нормаль $\mathbf{n}$ и глубину проникновения $deep$.

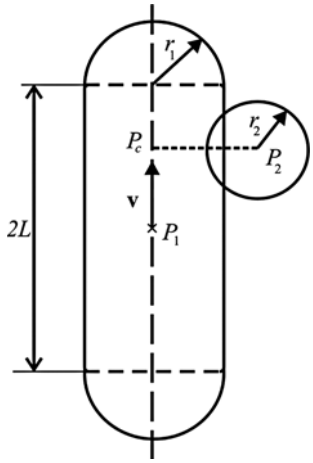

Рис. 4. Пересечение капсулы и сферы

Для обеспечения эффективного доступа процессоров (нитей варпа) к глобальной памяти данные записываются в виде структур InputCapsSpheresData и OutputCapsSpheresData массивов аналогично случаю двух сфер. Алгоритм пересечения также аналогичен описанному выше алгоритму, поэтому разберем только его функцию-ядро.

Алгоритм определения коллизии капсулы со сферой

1. Подготовка структуры данных для N пар капсул со сферами.

2. Копирование входных данных InputCapsSpheresData с хоста на GPU.

3. Запуск функции-ядра, выполняющей проверку пересечения N капсул со сферами и вычисляющей нормали, точки и глубины для пересекающихся пар.

4. Копирование выходных данных OutputCapsSpheresData с GPU на хост.

В описанной далее функции-ядре для простоты записи опущен индекс *tdx*. Его позиции легко понять исходя из функции-ядра пересечения сфер.

Функция-ядро

1. Вычисление номера нити $tdx = ThreadIdx.x + blockIdx.x \cdot blockDim.x$.

2. Вычисление проекции α вектора $\mathbf{P_1P_2}$ на ось капсулы:

$$\alpha = (P_{2x} - P_{1x})\nu_x + (P_{2y} - P_{1y})\nu_y + (P_{2z} - P_{1z})\nu_z.$$

3. Вычисление флагов для проверки принадлежности проекции центра сферы отрезку капсулы:

$$isMax = (\alpha > L),\ \ isMin = (\alpha < -L).$$

4. Обрезка α по границам оси капсулы:

$$\alpha = isMaxL + isMin(-L) + (1 - isMax - isMin)\alpha.$$

5. Вычисление точки проекции центра сферы на капсулу:

$$P_c = P_1 + \alpha \cdot \mathbf{v}.$$

6. Вычисление расстояния между точкой P_c и центром сферы P_2:

$$\mathbf{d} = P_c - P_2;\ \ d = \sqrt{d_x^2 + d_y^2 + d_z^2}.$$

7. Вычисление суммы радиусов капсулы и сферы: $r_s = r_1 + r_2$.

8. Вычисление флага пересечения:

$$isColl = (dist \leqslant r_s).$$

9. Если $isColl = 1$, то

9.1. Проверка на совпадение центров сферы и капсулы:

$$isNull = (d \leqslant \varepsilon).$$

9.2. Вычисление величины, обратной к d:

$$d' = \varepsilon isNull + d(1 - isNull);\ \ d^{-1} = (1 - isNull)/d'.$$

9.3. Вычисление глубины проникновения:

$$deep = r_s - d.$$

9.4. Вычисление координат нормали:

$$\mathbf{n} = (isNull, 0, 0)^{\mathrm{T}} + d^{-1}\mathbf{d}.$$

9.5. Вычисление смещения:

$$\Delta p = \frac{1}{2}(1 - isNull)(r_2 - r_1 - d).$$

9.6. Вычисление точки контакта:

$$P = P_c + \Delta p\mathbf{n}.$$

В этой функции пп. 9.1 и 9.2 также заменяют условный оператор.

В табл. 2 приведены данные апробации этого алгоритма на компьютере приведенной выше конфигурации. Из данных табл. 2 видно (столбцы 2 и 3), что время выполнения функции на GPU в 50...100 раз меньше, чем на CPU. Однако копирование данных в глобальную память и обратно и здесь занимает довольно большое время, и выигрыш с учетом времени копирования получается для числа пар, начиная с 8,5 тысяч.

Таблица 2

N, пары	Время на CPU, мс	Время выполнения ядра, мс	Время выполнения с учетом копирования данных, мс
8400	0,85	0,017	0,85
10 000	1,01	0,02	0,92
100 000	14,02	0,12	5,05
1 000 000	126,62	1,13	46,66

На рис. 5 показан пример падения 3 тыс. шариков в ящик. Для моделирования динамики этого процесса обрабатывается около 4,5 млн пар коллизий сфер.

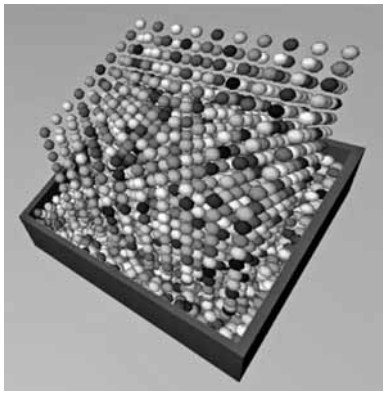

Рис. 5. Обработка коллизий 4,5 млн пар сфер

Заключение

В данной работе предложены параллельные алгоритмы определения коллизий N пар сфер и N пар контейнеров типа сфера/капсула на архитектуре CUDA. Проведенные исследования и апробации алгоритмов показали, что узким местом (по времени) в работе на этой архитектуре является пересылка данных из оперативной памяти в память графической карты и обратно. В связи с этим реальный выигрыш во времени работы алгоритма начинается с 17 тысяч пар сфер и 8,5 тысяч пар сфера/капсула.

Однако в ряде случаев подобное копирование не является необходимым, поскольку динамика движения объектов может также быть рассчитана средствами GPU. Например, многие современные компью-

терные игры используют GPU как для реализации 3D-графики, так и для расчета движения объектов.

Таким образом, результаты имеют практическое значение для специалистов, занимающихся разработкой систем моделирования динамики многих тел.

Работа выполняется при поддержке РФФИ, проект 16-37-00218-мол_а.

Список литературы

1. **Watt A., Policarpo F.** 3D Games. Real-time Rendering and Software Technology. Addison-Wesley, Reading, 2002.

2. **Mazhar H., Heyn T., Negrut D.** A scalable parallel method for large collision detection problems // Multibody System Dynamics. 2011, Vol. 26, Issue 1. P. 37—55.

3. **Трушин А. М., Михайлюк М. В.** Определение коллизий аппроксимирующих капсул и сфер трехмерных виртуальных динамических объектов // Труды НИИСИ РАН. 2016. Т. 6, № 2. С. 42—45.

4. **Трушин А. М.** Определение коллизий аппроксимирующих сфер и прямоугольных параллелепипедов в системах трехмерного моделирования // Программные продукты и системы. 2015. № 4. С. 105—109.

5. **Боресков А. В., Харламов А. А., Марковский Н. Д.** и др. Параллельные вычисления на GPU. М.: Изд-во МГУ, 2012. 978 с.

6. **Сандерс Д., Кэндрот Э.** Технология CUDA в примерах: введение в программирование графических процессоров. М.: ДМК Пресс, 2011.

Spheres Collision Detection Algorithms on GPU

M. V. Mikhayluk, mix@niisi.ras.ru, D. Sc., Head of Department, **A. M. Trushin,** Researcher, Scientific Research Institute of System Analysis Russian Academy of Sciences, Moscow, 117218, Russian Federation

Corresponding author:

Mikhayluk Mikhail V., Professor, D. Sc., Head of Department, Scientific Research Institute of System Analysis Russian Academy of Sciences, Moscow, 117218, Russian Federation
E-mail: mix@niisi.ras.ru

Received on April 05, 2017
Accepted on April 14, 2017

In the paper we consider the problem of pairwise collision detection of spheres and spheres with capsules in virtual object dynamics calculation systems. Performing this task in real-time on CPU (Central Processing Unit) for hundreds of thousands pairs of spheres and capsules is impossible. The algorithms to solve these tasks on GPU (Graphics Processing Unit) are proposed here. First, the technology and features of the CUDA (Compute Unified Device Architecture) architecture and features of work are described, in particular, memory organization, data transferring from CPU to GPU and back, the implementation of conditional statements, etc. Then, the algorithms of kernels to solve raised tasks are presented. In these algorithms conditional statements are replaced with simple expressions, involving condition flags. The proposed algorithms were tested on virtual scenes with different number of spheres and capsules (from several thousand to several million), and their working time on CPU was compared with the one computed with CUDA. It was shown that the tasks are solved on the GPU 50—100 times faster, starting from 17 thousand pairs for spheres and from 8,5 thousand sphere/capsule pairs. The developed software modules are implemented in the simulator for complex dynamic systems control, developed in Scientific Research Institute for System Analysis Russian Academy of Sciences. They can be used in multibody dynamics calculation systems, virtual environment systems, virtual laboratories, etc.

Keywords: *virtual objects, collision detection, sphere, capsule, GPU, CUDA, parallel computing, virtual environment systems*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project nos. 16-37-00218-мол_а.*

For citation:

Mikhayluk M. V., **Trushin A. M.** Spheres Collision Detection Algorithms on GPU, *Programmnaya Ingeneria*, 2017, vol. 8, no. 7, pp. 354—358.

DOI: 10.17587/prin.8.354-358

References

1. **Watt A., Policarpo F.** *3D Games. Real-time Rendering and Software Technology*, Addison-Wesley, Reading (2002).

2. **Mazhar H., Heyn T., Negrut D.** A scalable parallel method for large collision detection problems, *Multibody System Dynamics*, 2011, vol. 26, issue 1, pp. 37—55.

3. **Trushin A. M., Mikhaylyuk M. V.** Opredelenie kollizij approksimiruyushchih kapsul i sfer trekhmernyh virtualnyh dinamicheskih obektov (Collision detection for bounding capsules and spheres for 3D virtual dynamic objects), *Trudy NIISI RAN*, 2016, vol. 6, no. 2, pp. 42—45 (in Russian).

4. **Trushin A. M.** Opredelenie kollizij approksimiruyushchih sfer i pryamougolnyh parallelepipedov v sistemah trekhmernogo modelirovaniya (Collision detection for bounding spheres and rectangular parallelepipeds in 3D modeling systems), *Programmnye produkty i sistemy*, 2015, no. 4, pp. 105—109 (in Russian).

5. **Boreskov A. V., Harlamov A. A., Markovskij N. D., Mikushin D. N., Mortikov E. V., Myl'cev A. A., Saharnyh N. A., Frolov V. A.** *Parallelnye vychisleniya na GPU* (Parallel Computing on GPU), Moscow, Izdatel'stvo MGU, 2012, 978 p. (in Russian).

6. **Sanders J., Kendrot E.** *CUDA by Example. An Introduction to General-Purpose GPU Programming,* Addison-Wesley, 2010, 290 p.

УДК 004.056 DOI: 10.17587/prin.8.359-368

А. Н. Шниперов, канд. техн. наук, зав. лаб., e-mail: ashniperov@sfu-kras.ru,
А. П. Чистяков, магистрант, e-mail: Acella93@mail.ru,
ФГАОУ ВО "Сибирский федеральный университет", г. Красноярск

Способ и информационная система для конфиденциального обмена информацией в открытых компьютерных сетях

Рассмотрена проблема защиты конфиденциальной информации в открытых компьютерных сетях. Разработан и предложен способ безопасного информационного обмена в открытых сетях, а также архитектура соответствующей информационной системы. В статье также отражены технологические аспекты разработки и надежности.

Ключевые слова: *защита информации, защита телекоммуникаций, оконечное шифрование, распределение ключей шифрования*

Введение

Повсеместное распространение абонентских устройств подключения к сети Интернет и практическое применение информационно-коммуникационных технологий стали неотъемлемой частью жизни современного общества. Как следствие, программные системы мгновенного информационного обмена сообщениями (*Instant messaging*, *IM*) сделались очень популярны, обрели большое число пользователей и огромный объем передаваемых данных. Так, например, только приложение-мессенджер *WhatsApp* используют более одного миллиарда человек в 180 странах [1]. Столь широкое распространение *IM*-систем ставит задачу безопасного обмена конфиденциальной информацией в открытых сетях в ряд наиболее важных. Особенно актуально решение этой задачи, например, для ряда таких категорий пользователей, как государственные служащие.

Практически все распространенные *IM*-системы построены на основе централизованной архитектуры и несмотря на серьезные механизмы защиты информации, применяемые разработчиками, они имеют общий и существенный недостаток — отсутствие доверия к центральной части (сервис-провайдеру) *IM*-систем. При этом центральная сторона может быть скомпрометирована как злоумышленниками, так и владельцем системы по тем или иным причинам. Другими словами, может быть реализована атака "человек посередине" (*Man-in-the-middel*, *MITM*) [2], которая, в свою очередь, несет ряд угроз нарушения конфиденциальности и целостности информации пользователей в процессе информационного обмена. Наличие таких угроз, особенно в контексте того, что большая часть распространенных *IM*-систем принадлежит иностранным компаниям, существенно ограничивает или делает невозможным их использование для обмена конфиденциальной информацией.

В настоящей статье предложен способ передачи конфиденциальной информации в компьютерных сетях общего доступа, учитывающий возможную компрометацию центральной части, и практически нивелирующий угрозу конфиденциальности передаваемой информации. Кроме того, способ предполагает подход к контролю целостности информационного обмена. В работе приведена архитектура разработанной информационной системы для конфиденциального обмена информацией, реализованной в виде программного комплекса *ruMessanger* [3].

Обзор существующих решений и постановка задачи

Подавляющее большинство современных *IM*-систем являются централизованными, в том числе *WhatsApp*, *Viber*, *Telegram*, *Threema* и другие системы, построенные на клиент-серверной архитектуре. Кратко рассмотрим их с позиции обеспечения конфиденциальности передаваемой информации.

Обеспечение конфиденциальности передаваемой информации в системе *WhatsApp* [4] между абонентом (клиентское устройство) и центральным узлом базируется на протоколе *Noise Pipes* [5]. Этот протокол включает в себя протокол Диффи-Хеллмана на эллиптических кривых (*Elliptic Curve Diffie-Hellman*, *ECDH*) для выработки ключей шифрования, а также алгоритм шифрования *AES* в режиме счетчика с аутентификацией Галуа (*Galois Counter Mode*, *GCM*) [6]. Начиная с версии 2.16 для передачи конфиденциальной информации между абонентами используется *Signal Protocol* [4, 7], позволяющий организовать между ними оконечное шифрование (*End-to-end encryption*, *E2EE*) [2]. Такое шифрование обеспечивается путем выработки общего долгосрочного секретного ключа

между абонентами — мастер-ключа — с помощью нескольких пар различных ключей (идентификационные, подписывающие, эфемерные, одноразовые) абонентов, хранящихся на сервере. С использованием мастер-ключа и алгоритма *Double ratchet* [8] формируются сеансовые ключи шифрования. Контроль целостности и подлинности сообщений достигается генерированием кода аутентичности сообщения (*Message Authentication Code*, *MAC*) [2]. Для защиты от *MITM*-атак абонентам предлагается по сторонним каналам связи, либо при личной встрече верифицировать свои *QR*-коды, которые формируются с использованием идентификационных открытых ключей обоих абонентов.

В системе *Telegram* используется аналогичная архитектура. Защита клиент-серверного канала связи основана на подходе *Pinning* [9] типа *Public key pinning* с *RSA* (2048-бит). Протокол *MTProto* [10] в режиме секретных чатов использует концепцию *E2EE* для обеспечения конфиденциальности информации, передаваемой между абонентами. Шифрование сообщения осуществляется посредством алгоритма *AES* с расширением неопределенного искажения (*Infinite Garble Extension*, *IGE*) [11] по 256-битному сеансовому ключу. Такой ключ вырабатывается на основе общего секрета абонентов, генерируемого по протоколу Диффи-Хелмана, а также кода целостности. Сеансовый ключ одновременно является и *MAC*-кодом для контроля целостности сообщений. Защита от *MITM*-атак построена на системе, аналогичной по принципам защиты системы *WhatsApp*, а именно — на сравнении визуализируемых ключей шифрования.

Начиная с версии 6.0 *IM*-система *Viber* также стала использовать концепцию *E2EE* для защиты информации, передаваемой абонентами. Протокол связи системы *Viber* [12] основан на идеях и принципах протокола *Signal Protocol* [7]. Каждый абонент генерирует идентификационный асимметричный ключ, а также набор вспомогательных ключей под конкретное устройство, необходимых для создания защищенного сеанса связи. Вспомогательные ключи шифрования состоят из двух асимметричных 256-битных ключей *ratchet* и *handshake*. При этом открытые ключи хранятся на серверах системы, а закрытые — на оконечном устройстве. Для создания сеанса связи между двумя абонентами, отправитель запрашивает открытые ключи (идентификационный и вспомогательный) получателя у сервера. На основе таких ключей вырабатывается сеансовый ключ, необходимый для шифрования одноразовых ключей шифрования сообщений посредством поточного шифра *Salsa20* [13]. Защита от *MITM*-атак построена на визуальном/аудиальном сравнении общего секрета посредством аудио/видеозвонка, выработанного по *ECDH* из закрытого идентификационного ключа отправителя и открытого идентификационного ключа получателя.

Система *Threema* [14] для защиты клиент-серверного канала использует протокол *TLS* версии 1.2 наряду с подходом *Certificate Pinning* [9]. Информационный обмен между абонентами также основан на концепции *E2EE*. Для этого на основе открытого ключа получателя и закрытого ключа отправителя по протоколу *ECDH* вырабатывается 256-битный симметричный ключ. Открытые ключи абонентов хранятся на сервере, а закрытые ключи — на абонентских устройствах. В качестве алгоритма шифрования сообщений используется поточный шифр *XSalsa20* [15], а на основе шифротекста сообщения вычисляется код аутентичности сообщения (*MAC*-код) [2]. Защита от *MITM*-атак построена на генерации абонентами *QR*-кода, получаемого из открытого ключа абонента и его идентификатора. *QR*-коды абонентов предлагается верифицировать при личной встрече или посредством иного способа связи, который абоненты считают безопасным.

Несомненно, рассмотренные *IM*-системы имеют достаточно эффективные механизмы обеспечения конфиденциальности информационного обмена между абонентами, потому что базируются на концепции оконечного (абонентского) шифрования. Однако, по мнению авторов, их объединяет и один весомый недостаток — потенциальная возможность осуществления *MITM*-атаки между любыми абонентами системы посредством компрометации ее центрального узла. При этом компрометация может быть осуществлена как злоумышленниками, так и непосредственными разработчиками, мотивированными экономическими, политическими или иными соображениями.

На рис. 1 представлена диаграмма классической *MITM*-атаки, исходящей со стороны сервера *S* на ключевую информацию. В случае компрометации сервера злоумышленник может скомпрометировать передаваемые значения открытых ключевых параметров своими значениями, выработать с каждым участником информационного обмена общий ключ, а затем получить полный доступ к конфиденциальной информации. Данная *MITM*-атака носит вполне вероятный характер и уже была продемонстрирована в работе [16] на примере системы *Signal* (*TextSecure*), а также в аналитическом обзоре [17] протокола *Telegram*.

Схожая атака может иметь место и в случае компрометации базы данных (БД) открытых ключей пользователей, хранящихся на серверах *IM*-системы. Злоумышленник может временно подменить откры-

Рис. 1. Общая схема *MITM*-атаки на ключевую информацию

тые ключи абонентов, хранящиеся в БД. Подмена ключей и знание структуры передаваемых данных и идентификаторов абонентов также позволят злоумышленнику создать сеанс связи от чужого имени и скомпрометировать весь информационный обмен.

С учетом изложенного выше, актуальна научно-техническая задача по разработке способа безопасного обмена конфиденциальной информацией в компьютерных сетях общего доступа, который бы позволил устранить обозначенные недостатки *IM*-систем. Решение данной задачи включает в себя разработку архитектуры информационной системы, протоколов сетевого взаимодействия и различных алгоритмов.

Предлагаемый подход и модель информационной системы

Использование только одного канала связи "клиент-сервер" при распределении и выработке ключей шифрования для организации безопасного обмена конфиденциальной информацией в *IM*-системах ставит угрозу компрометации ключевой информации со стороны центрального узла в разряд актуальных даже при использовании оконечного шифрования. В целях более эффективного противодействия реализации такой угрозы предлагается использовать идею разделения общего секрета (ключей шифрования и сопутствующих данных) абонентов системы на части и последующей их передачи по нескольким независимым каналам связи. Такой подход, по мнению авторов, позволит децентрализовать процесс распределения и выработки ключей шифрования.

Модель предлагаемой информационной системы (ИС) Σ можно представить с помощью следующих теоретико-множественных отношений:

$$\Sigma = \{A, S, M, K, N, P, Q, CA, SA\},$$

где A — множество абонентов системы; S — центральный узел (сервер); M — множество передаваемых сообщений; K — множество ключей шифрования; N — каналы связи; P — протоколы информационного обмена; Q — множество состояний системы; CA — криптографические алгоритмы; SA — подсистема управления доступом.

Множество сообщений M между участниками информационного обмена $M = \{M_{AS}, M_{AA}\}$, где $M_{AS} = \{ms_1, ..., ms_x\}$ — множество сообщений типа "абонент—сервер"; $M_{AA} = \{ma_1, ..., ma_y\}$ — множество сообщений типа "абонент—абонент", при этом $M_{AA} \cap M_{AS} = \varnothing$.

Ключи шифрования K, используемые для криптографической защиты информации, $K = \{k_{ses}, k_S, k_A, K_S, K_A, K_T\}$, где k_{ses} — симметричный сеансовый ключ, k_S — симметричный ключ сервера; k_A — симметричный ключ абонента; $K_S = \{k_S^{pub}, k_S^{pri}\}$ — открытый и закрытый ключи сервера; $K_A = \{k_A^{pub}, k_A^{pri}\}$ — открытый и закрытый ключи абонентов; $K_T = \{k_T^{pub}, k_T^{pri}\}$ — разовый открытый и закрытый ключи.

Архитектура ИС при этом подразумевает использование трех различных и независимых каналов связи $N = \{n_1, n_2, n_3\}$, где n_1 — канал "абонент—сервер"; n_2 — канал "абонент—абонент", являющийся каналом оверлейной пиринговой сети (*Peer-to-Peer, p2p*); n_3 — сотовая связь. Введением избыточности каналов связи достигается снижение зависимости от центральной части ИС при обмене ключевой информации. Информационный обмен в ИС осуществляется по протоколам $P = \{p_1, p_2\}$, где p_1 — протокол взаимодействия "абонент-сервер", а p_2 — "абонент—абонент".

Информационная система может находиться в одном из множества состояний $Q = \{q_i\}$, где q_0 — начальное состояние ИС, определяемое ее конфигурацией при вводе в эксплуатацию; q_i — состояние ИС в некоторый момент времени *t*. В процессе функционирования ИС под воздействием различных событий может сохранять или изменять свое состояние. Переход системы из одного состояния в другое определяется функцией перехода

$$T : \begin{bmatrix} (q_i \cup a) \\ (q_i \cup t) \end{bmatrix} \rightarrow q_{i+1},$$

основанием к переходу ИС в новое состояние являются события $D = \{d_a, d_k\}$, где d_a — добавление абонента a в ИС; d_k — обновление элементов множества ключей K, определяемых функцией времени *t*. Так, например, регистрация нового абонента $A = A \cup a$, $a \notin A$ в ИС, которая находится в начальном состоянии, порождает переход в состояние $q_1 = T(q_0, a)$. Кроме изменения числа абонентов в ИС ее состояние меняется и во времени, что связано с периодическим обновлением элементов множества ключей шифрования K.

В ИС используются криптографические алгоритмы $CA = \{SE, AE, H, AK\}$, где SE — алгоритм симметричного шифрования (ГОСТ Р 34.12—2015 [18] в режиме OFB [19]); AE — алгоритм шифрования с открытым ключом (основанный на SE и AK); H — криптографическая хеш-функция (ГОСТ Р 34.11—2012 [20]); AK — криптографический протокол выработки ключей (Диффи-Хеллмана на эллиптических кривых, $ECDH$).

Выбор режима шифрования с обратной связью по выходу (OFB) обусловлен следующими факторами, положительными в контексте разрабатываемой ИС:

- отсутствие необходимости в дополнении последнего блока незначащими, обратимыми байтами, которое сокращает расходы на пересылку, что особенно важно для большого количества коротких сообщений;
- возможность организации потоковой передачи данных видео/аудиозвонка, что является одной из функций ИС;
- локализация распространения ошибок передачи в пределах одного блока, что актуально в условиях

Рис. 2. Структурно-функциональная схема ИС

потоковой передачи данных видео/аудиозвонка в ИС при плохой мобильной связи, обеспечивающей соединение с сетью Интернет.

Выбор в качестве хеш-функции H стандарта ГОСТ Р 34.11—2012 обусловлен тем обстоятельством, что такая хеш-функция удовлетворяет современным требованиям к криптографической стойкости. При необходимости усиления защиты ИС от *HMAC*-атак, которые являются маловероятными и не рассматриваются в предлагаемой модели ИС, хеш-функция H может быть переведена в ключевую *HMAC*-функцию.

Подсистема управления доступом $SA = \{I, V, J\}$ в ИС состоит из трех механизмов: I — аутентификация; V — авторизация; J — контроль доступа. Аутентификация в ИС является многофакторной, т. е. $I = \{k_A, OTC\}$, где OTC — одноразовые коды подтверждения, передаваемые по каналу связи n_3.

Структурно-функциональная схема ИС представлена на рис. 2. Взаимодействие между абонентами A и центральным узлом S осуществляется по сети Интернет с использованием канала связи n_1 и протокола p_1. Коммуникации между абонентами также выполняются посредством центрального узла ИС и канала n_1, но с использованием протокола p_2. При этом отметим, что в процессе обмена ключами шифрования совместно с n_1 используется и канал n_2. Для передачи одноразовых кодов OTC центральный узел S использует любой сторонний сервис передачи *sms*-сообщений на абонентские устройства по каналу n_3 оператором сотовой связи.

Инфраструктура и распределение ключей шифрования

Как уже было отмечено ранее, разрабатываемая ИС должна иметь такую архитектуру, в основе которой лежит распределение общего секрета между абонентами и центральным узлом таким образом, чтобы исключить компрометацию ключей шифрования, даже в случае, если злоумышленник имеет непосредственный доступ к информации, хранящейся на серверах ИС. Для $\forall k \in K$ определен жизненный цикл, который включает в себя такие стадии, как генерация, распределение, использование, смена и хранение. Рассмотрим эти этапы более подробно.

Ключевая пара центрального узла $K_S = \left\{k_S^{pub}, k_S^{pri}\right\}$ состоит из двух ключей: закрытого k_S^{pri}, открытого k_S^{pub} размерностью 512 и 1024 бит соответственно. Время жизни K_S определяется временем выпуска новой версии программного комплекса. Открытый ключ $k_S^{pub} \in K_S$ встраивается в клиентскую часть ИС для ассоциации с сервером S, что реализует подход *Public key pinning* [9]. Ключевая пара абонентов $K_A = \left\{k_A^{pub}, k_A^{pri}\right\}$ является эфемерной, с временем жизни 24 часа. После истечения времени действия абонент передает на хранение серверу шифротекст закрытого ключа k_A^{pri}, полученного по алгоритму SE и ключу k_A, а также генерирует новую ключевую пару. Открытый ключ $k_A^{pub} \in K_A$ прекратившей существование пары ключей, безвозвратно удаляется. Временная ключевая пара $K_T = \left\{k_T^{pub}, k_T^{pri}\right\}$ определяется в рамках работы протокола p_2 в целях защиты от прослушивания открытой информации, передаваемой по каналу связи n_2.

Симметричные 256-битные ключи $k_S \in K$ и $k_A \in K$ вырабатываются из многоразового пароля (секретной фразы), задаваемого абонентами A и центральным узлом S. Для выработки симметричного ключа применяют стандарт формирования ключа на основе пароля (*Password-Based Key Derivation Function*, *PBKDF*) версии *PBKDF2*, рекомендуемый в информационном документе *RFC* 2898. В качестве псевдослучайной функции используется хэш-функция ГОСТ Р 34 11—2012 с длиной выхода 512 бит. Такие ключи решают две задачи в ИС. Во-первых, они являются одним из факторов аутентификации пользователя, а во-вторых, они используются в качестве мастер-ключа для защиты k_S и k_A при хранении на центральном узле. Симметричный 256-битный ключ сессии $k_{ses} \in K$ вырабатывается каждый раз при создании сеанса связи (сессии) между абонентами A и центральным узлом S. Время жизни сессии, равно как и сеансового ключа, определяется временем взаимодействия абонента с центральным узлом. В случае продолжительного отсутствия связи сессия завершается.

Распределение ключей в ИС осуществляется по сети Интернет с использованием двух независимых каналов связи n_1 и n_2 посредством протоколов p_1 и p_2 соответственно. Процесс обмена ключевой информацией между абонентом и центральным узлом (сервером) в рамках протокола p_1 в общем виде можно описать следующим алгоритмом.

Шаг 1. Абонент генерирует ключ сессии k_{ses} с использованием генератора псевдослучайных последовательностей.

Шаг 2. Абонент зашифровывает k_{ses} с помощью открытого ключа сервера $k_S^{pub} \in K_S$, т. е. $\hat{k}_{ses} = AE\left(k_{ses}, k_S^{pub}\right)$. Затем по алгоритму H вычисляет код целостности от k_{ses} и отправляет по каналу n_1 на сервер запрос на создание сессии.

Шаг 3. Сервер посредством своего закрытого ключа $k_S^{pri} \in K_S$ расшифровывает полученный за-

прос, т. е. $k_{ses} = AE^{-1}\left(\hat{k}_{ses}, k_S^{pri}\right)$. Затем сервер проверяет целостность данных. В случае неудачи алгоритм завершает работу, а абоненту возвращается ошибка.

Шаг 4. Сервер генерирует идентификатор сессии id_{ses} с использованием генератора псевдослучайных последовательностей, ассоциирует его с ключом сессии и заносит ассоциацию в список активных сессий.

Шаг 5. Сервер зашифровывает идентификатор сессии абонента с помощью k_{ses}, т. е. $\widehat{id}_{ses} = SE\left(id_{ses}, k_{ses}\right)$. Далее сервер формирует код целостности и отправляет ответ абоненту.

Шаг 6. Абонент расшифровывает ответ сервера, т. е. $id_{ses} = SE^{-1}\left(\widehat{id}_{ses}, k_{ses}\right)$ и проверяет его целостность.

Шаг 7. Обмен ключами считается состоявшимся. Сервер и абонент осуществляют конфиденциальный обмен сообщениями, которые зашифровываются по алгоритму *SE* с использованием сессионного ключа k_{ses}.

В свою очередь, процесс обмена ключевой информацией для реализации оконечного абонентского шифрования между абонентами в рамках протокола p_2 в общем виде можно описать следующим алгоритмом.

Шаг 1. Абонент *A*1 генерирует разовую пару ключей $K_T = \left\{k_T^{pub}, k_T^{pri}\right\}$.

Шаг 2. Абонент *A*1, передает абоненту *A*2 по каналу n_2 разовый открытый ключ k_T^{pub} и запрос на отправку открытого ключа k_{A2}^{pub} абонента *A*2. В качестве идентификатора узла в сети *p2p* при этом выступает идентификатор абонента в ИС.

Шаг 3. Получив запрос от абонента *A*1, абонент *A*2 разделяет свой открытый ключ k_{A2}^{pub} на две равные части k_1 и k_2, т. е. $k_{A2}^{pub} = k_1 \bullet k_2$, где • — операция конкатенации.

Шаг 4. Абонент *A*2 зашифровывает k_1 и k_2 с помощью разового открытого ключа k_T^{pub}: $\hat{k}_1 = AE\left(k_1, k_T^{pub}\right)$ и $\hat{k}_2 = AE\left(k_2, k_T^{pub}\right)$ и передает их абоненту *A*1 по каналам n_1 и n_2 соответственно.

Шаг 5. Прием $\hat{k}_1$ и $\hat{k}_2$ абонентом *A*1 осуществляется в два этапа. На первом этапе осуществляется прием $\hat{k}_2$ по каналу n_2. На втором этапе абонент *A*1 получает $\hat{k}_1$ по каналу n_1, путем периодического опрашивания сервера на наличие событий "новый ключ".

Шаг 6. Получив $\hat{k}_1$ и $\hat{k}_2$ абонент *A*1 расшифровывает их по разовому закрытому ключу k_T^{pri}: $k_1 = AE^{-1}\left(\hat{k}_1, k_T^{pri}\right)$ и $k_2 = AE^{-1}\left(\hat{k}_2, k_T^{pri}\right)$ и проверяет их целостность. В случае подтверждения целостности k_1 и k_2 абонент *A*1 восстанавливает открытый ключ абонента *A*2: $k_{A2}^{pub} = k_1 \bullet k_2$, в противном случае алгоритм завершает свою работу, а абоненту *A*2 возвращается ошибка.

Шаг 7. Абонент *A*1 зашифровывает свой открытый ключ k_{A1}^{pub} открытым ключом абонента *A*2: $\hat{k}_{A1}^{pub} = AE\left(k_{A1}^{pub}, k_{A2}^{pub}\right)$. Затем вычисляет код целостности $h = H\left(k_{A1}^{pub}\right)$. Полученные $\hat{k}_{A1}^{pub}$ и h передаются абоненту *A*2 по каналу n_2.

Шаг 8. Обмен ключами считается состоявшимся. Далее абоненты осуществляют обмен конфиденциальной информацией, которая зашифровывается с использованием публичных ключей абонентов k_{A1}^{pub} и k_{A2}^{pub}.

Необходимо отметить, что алгоритм шифрования с открытым ключом *AE* не является таковым в обычном понимании. Он построен на основе симметричного алгоритма шифрования *SE* и протокола *ECDH*. В общем виде алгоритм *AE* можно описать следующим образом.

Шаг 1. Абонент *A*1 генерирует случайный симметричный ключ k_m и зашифровывает сообщение m, т. е. $\widehat{m} = SE\left(m, k_m\right)$.

Шаг 2. Абонент *A*1 генерирует разовую пару ключей $K = \left\{k^{pub}, k^{pri}\right\}$.

Шаг 3. Используя k^{pri} и k_{A2}^{pub} по алгоритму *ECDH*, абонент *A*1 вырабатывает мастер-ключ, т. е. $k_{master} = ECDH\left(k^{pri}, k_{A2}^{pub}\right)$.

Шаг 4. Абонент *A*1 зашифровывает k_m с помощью ключа k_{master}, т. е. $\hat{k}_m = SE\left(k_m, k_{master}\right)$.

Шаг 5. Абонент *A*1 передает абоненту *A*2 криптоконтейнер, включающий в себя $\widehat{m}$, $\hat{k}_m$, k^{pub} и иную служебную информацию.

Таким образом можно определить обратный алгоритм AE^{-1}.

Шаг 1. Абонент *A*2 получает криптоконтейнер, включающий в себя $\widehat{m}$, $\hat{k}_m$, k_T^{pub} и иную служебную информацию.

Шаг 2. Используя свой закрытый ключ k_{A2}^{pri} и k^{pub}, абонент *A*2 вырабатывает мастер-ключ, т. е. $k_{master} = ECDH\left(k_{A2}^{pri}, k^{pub}\right)$.

Шаг 3. Абонент *A*2 расшифровывает ключ шифрования сообщения $\hat{k}_m$ с помощью ключа k_{master}, т. е. $k_m = SE^{-1}\left(\hat{k}_m, k_{master}\right)$.

Шаг 4. Абонент *A*2 расшифровывает сообщение m с помощью ключа k_m, т. е. $m = SE^{-1}\left(\widehat{m}, k_m\right)$.

Отметим, что для генерации случайных последовательностей (ГСП), включая ключевую информацию, необходимо использовать криптографически стойкий генератор случайных чисел (ГСЧ). Технологическая реализация ГСЧ в предлагаемой ИС основана на библиотеке криптографических алгоритмов *WebCrypto GOST* [21], которая включает в себя криптографически стойкий ГСЧ.

Протоколы сетевого взаимодействия

Как уже было отмечено ранее, информационный обмен в разработанной ИС осуществляется по протоколам $P = \{p_1, p_2\}$. Протокол p_1 предназначен для передачи сообщений M_{AS} между сервером S и абонентами A по каналу n_1. На рис. 3 представлена схема преобразования сообщения $ms_i \in M_{AS}$ в закодированное сообщение $\widehat{ms}_i \in \widehat{M}_{AS}$ для последующей передачи по протоколу p_1.

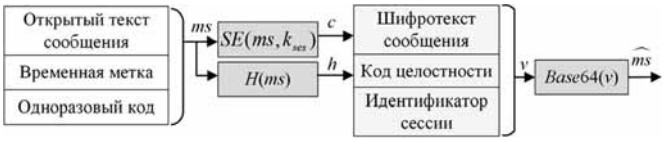

Рис. 3. Схема преобразования сообщений в протоколе p_1

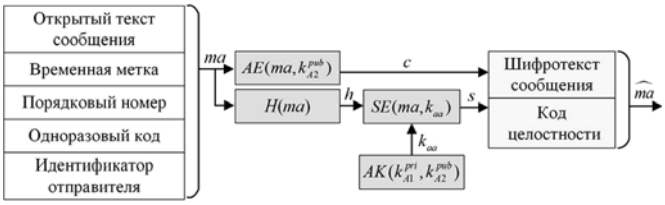

Рис. 4. Схема преобразования сообщений в протоколе p_2

В свою очередь, передача сообщений M_{AA} между абонентами ИС осуществляется с использованием протокола p_2 и канала n_2. На рис. 4 представлена схема преобразования сообщения $ma_i \in M_{AA}$ в закодированное сообщение $\widehat{ma_i} \in \widehat{M}_{AA}$ для последующей передачи от одного абонента другому. Заметим, что контроль целостности информации между абонентами осуществляется посредством кода (метки) целостности с использованием счетчика сообщений.

Для экономии вычислительных ресурсов мобильных устройств предлагается осуществлять проверку целостности сообщений только для части передаваемых сообщений, включая первое. Другую часть сообщений предлагается проверять на целостность посредством счетчика сообщений. Поскольку весь объем передаваемой информации защищен посредством оконечного шифрования, только абонентам будет известно общее количество переданной/полученной информации в ходе информационного обмена. В качестве начального значения для счетчика задается большое случайное число.

Оценка безопасности предлагаемого подхода

Исходя из постановки задачи, предлагаемый подход для безопасного обмена информацией в компьютерных сетях общего пользования основывается на следующей неформальной модели нарушителя:

- нарушитель может быть внешним и оказывает воздействие на абонентское устройство из компьютерной сети посредством сниффинга, в том числе и активного;
- нарушитель может быть внутренним и оказывает воздействие на центральный узел ИС (сервер), например, имеет доступ к базе данных ИС;
- нарушитель обладает всей информацией, необходимой для подготовки и проведения атак, за исключением информации, доступ к которой со стороны нарушителя исключается, например, информация о закрытых ключах абонента;
- нарушитель обладает необходимыми знаниями и средствами для проведения атак.

С учетом представленной модели нарушителя в рамках описываемого подхода рассматриваются следующие виды атак на ИС:

- пассивная и активная формы атаки "человек посередине" (*MITM*-атака), включая перехват, искажение и вставку;
- подмена (*impersonation*) абонента;
- повторное навязывание сообщения (*replay attack*), включая
  - ◇ задержку передачи сообщений (*forced delay*);
  - ◇ отражение (*reflection attack*);
- комбинированная атака (*interleaving attack*).

В рамках данной неформальной модели нарушителя и рассматриваемых потенциальных атак определим, что под понятием безопасного обмена информацией понимается комплекс требований и мер, реализованных в предлагаемой ИС в целях обеспечения санкционированного доступа к информации уполномоченным на это пользователям, предотвращения несанкционированного доступа к ИС, обеспечения конфиденциальности, доступности и целостности передаваемой информации.

Оценим предпринятые меры защиты информации от описанного класса атак с позиции рассматриваемой модели нарушителя.

○ Защита от *MITM*-атаки со стороны внешнего нарушителя на множество сообщений типа "абонент—сервер" достигается путем применения известного подхода к распределению ключей — *Pinning*, типа *Public key*. Такой подход позволяет неявным образом аутентифицировать сервер и предотвратить возможность злоумышленника завладеть сессионным ключом k_{ses}, потому что для этого ему необходимо обладать серверной частью закрытого ключа k_S^{pri}. Даже если нарушитель гипотетически сможет выступить в качестве посредника в *MITM*-атаке и подменить сообщение абонента, то он не сможет выдать принимающему абоненту зашифрованный ответ по ожидаемому ключу k_{ses}. Следует также отметить, что во время создания сеанса связи создается ассоциативная связь между идентификатором абонента в ИС, сеансовым ключом k_{ses}, идентификатором сеанса id_{ses} и токеном (маркером) безопасности. Такая ассоциация сохраняется как у абонента, так и на сервере в списке активных сессий, что позволяет серверу разграничивать доступ и предотвращать несанкционированный доступ к информации.

○ Защита от MITM-атаки со стороны внешнего нарушителя на множество сообщений типа "абонент-абонент" основывается на использовании счетчика сообщений и кода целостности. Чтобы сформировать правильный код целостности или узнать значения счетчика нарушителю необходимо знать закрытый ключ k^{pri} подменяемого абонента, либо подменить абонентский открытый ключ k^{pub}, т. е. скомпрометировать все используемые каналы связи в ИС, что является трудновыполнимой задачей. В ином случае злоумышленник будет вынужден использовать свою часть закрытого ключа $k'^{pri} \neq k^{pri}$, что легко выявляет факт подмены абонента.

- Для защиты от повторного навязывания сообщений используются одноразовые коды и метки времени, по которым в том числе можно определять актуальность ("свежесть") сообщений.
- Для защиты передаваемых открытых ключей от *MITM*-атаки внутренним нарушителем используются различные и независимые каналы связи и механизмы обеспечения целостности передаваемых данных. В случае если нарушитель скомпрометировал один из каналов связи, например, подменил часть ключа k_1 на k_1', то абонент-получатель сможет обнаружить подмену. Это осуществляется путем сравнения значения вычисленного хэш-кода открытого ключа со значением, который на сервере опубликовал абонент-отправитель, т. е. $H(k_1 \bullet k_2) \neq H(k_1' \bullet k_2)$.
- Для защиты от внутреннего нарушителя конфиденциальной информации абонента, которая хранится в базе данных ИС, применяется код целостности, а также шифрование с применением закрытого ключа абонента k^{pri} или симметричного ключа абонента k, созданного на основе секретной фразы абонента.
- Для защиты от несанкционированного доступа предлагаемая ИС реализует функции аутентификации (в том числе и многофакторной), авторизации и управления доступом.

Предлагаемая ИС в полном объеме реализует перечисленные меры по защите информации, что позволяет говорить о безопасности предлагаемого способа обмена конфиденциальной информацией в компьютерных сетях общего доступа в рамках рассматриваемых модели нарушителя и класса потенциальных атак.

Техническая реализация и нагрузочное тестирование информационной системы

Предлагаемая модель ИС для безопасного обмена информацией в компьютерных сетях общего доступа реализована в виде программного комплекса *ruMessanger* [3]. Этот комплекс базируется на открытых и платформо-независимых технологиях *Apache Cordova* и *Node.js*. Это обстоятельство позволяет снизить вероятность наличия недекларируемых возможностей, которые могут присутствовать в проприетарных решениях. В качестве криптографических алгоритмов были применены государственные стандарты России в области криптографии, которые реализованы в библиотеке алгоритмов *WebCrypto GOST* [21]. Для организации децентрализованного канала связи посредством оверлейной пиринговой сети использовалась библиотека алгоритмов *PeerJS* [22]. Программный комплекс реализован с использованием языка программирования высокого уровня *JavaScript*. При разработке клиентской части использовались также языки *HTML* и *CSS*. В качестве системы управления базами данных (СУБД) выбрана *MongoDB*, поставляемая в открытых исходных кодах и имеющая хорошие возможности по масштабированию [23] при последующей кластеризации ИС. Взаимодействие абонентов с ИС осуществляется посредством клиентского приложения для мобильного устройства или с помощью браузера. Коммуникация между клиентской частью и серверной частью осуществляется посредством протокола *http*. Безопасность информационного обмена обеспечивается авторскими механизмами защиты информации, рассмотренными ранее.

Для повышения качества разработки программного комплекса был использован принцип "пусть падает" (*Let it crash*). Этот принцип основан на том, что исключительные ситуации не обрабатывались, а в случае их возникновения программа перезапускалась с выводом ошибки. Такой подход позволил оперативно выявлять места синтаксических, лексических и логических ошибок. Кроме того, уменьшение числа возможных ошибок осуществлялось путем применения инструментов автоматизации тестирования. Для модульного тестирования использовался инструментарий *QUnit*, а для функционального тестирования — *Selenium*.

Интерфейс клиентской части разработанного программного комплекса представлен на рис. 5, см. третью сторону обложки.

Очевидно, что с ростом числа абонентов ИС возрастает нагрузка на ее центральный узел (серверную часть), растет число обрабатываемых данных, увеличивается количество входящих и исходящих потоков данных и др. В связи с этим обстоятельством возрастает время обслуживания запроса и увеличивается вероятность потенциальных сбоев в работе ИС. Чтобы оценить потенциальную нагрузочную способность серверной части разработанной ИС, было осуществлено ее нагрузочное тестирование. Для этого были определены следующие тестовые сценарии.

- Сценарий *1* имитирует аутентификацию и авторизацию пользователя ИС с получением 10 входящих сообщений. Экспериментальным путем было определено, что данный сценарий является наиболее вычислительно затратным, так как задействует основные ресурсоемкие операции, а именно — обращение к базе данных и криптографические преобразования.
- Сценарий *2* имитирует отправку сообщений в ИС авторизовавшимся пользователем, что обеспечивает основные функции ИС.
- Сценарий *3* имитирует поиск конкретного абонента в ИС по телефонному номеру и отображение расширенных сведений о нем.
- Сценарий *4* содержит все описанные выше сценарии, которые запускаются одновременно и в равном соотношении.

В качестве инструментария для проведения тестовых испытаний был использован программный продукт *Apache jMater* [24]. Методика нагрузочного тестирования базировалась на варьировании числа экземпляров нагрузочного сценария в целях регистрации ключевых показателей производительности серверной части ИС [25]. Серверная часть программного комплекса была развернута на персональном компьютере с двухъядерным процессором *Intel Core i*3-4170, позволяющем реализовать четыре вычислительных потока

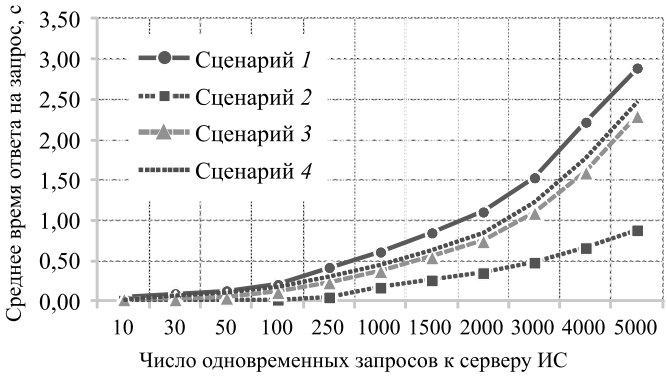

Рис. 6. Результаты нагрузочного тестирования

(ОЗУ емкостью 8 Гб). В качестве операционной системы была использована *Windows* 8. Результаты нагрузочного тестирования представлены на рис. 6.

Как следует из представленных графиков, превышение отметки в ~2000 одновременно обрабатываемых запросов порождает время отклика системы более 1 с, что можно принять за критическое. В реальных условиях эксплуатации ИС пороговому времени отклика будет соответствовать еще меньшее значение одновременно исполняемых запросов. В такой ситуации актуален вопрос о потенциальной возможности масштабирования ИС.

Масштабируемая архитектура информационной системы

Программная платформа *Node.js* потенциально поддерживает масштабирование как вертикальное, так и горизонтальное [26]. Первый подход подразумевает наращивание вычислительных возможностей физического сервера. Второй подход базируется на принципах разделения серверной части на несколько физических серверов и балансировки запросов к ИС между ними.

На рис. 7 (см. третью сторону обложки) представлена архитектура масштабируемой ИС. Клиентское приложение абонентов обращается к балансировщику запросов ИС (*Front-end*), в качестве которого может выступать веб-сервер *Nginx* [27]. Последний, в свою очередь, формирует очереди запросов к серверам приложений *Node.js*, реализующим бизнес-логику ИС.

В зависимости от логики обработки запроса серверы приложений обращаются к БД или к сетевому кластерному хранилищу. При обращении к БД запросы передаются серверу маршрутизации СУБД — *MongoS*, который является интерфейсом для обращения к БД как к единому целому. Сервер маршрутизации СУБД перенаправляет все запросы подходящему сегменту БД — *MongoD*. Каждый сегмент развертывается в виде набора реплик, на которых хранится некоторая часть всего множества данных.

Для снижения нагрузки на серверы приложений предполагается использовать подсистему кэширования представлений, реализованную на базе журналируемого хранилища данных типа "ключ-значение" *Redis* [28]. Для хранения передаваемых между абонентами файлов предполагается использовать кластер файловых серверов с кластерной файловой системой, например, *Oracle Cluster File System* [29].

Заключение

В настоящей статье рассмотрены и проанализированы наиболее распространенные системы мгновенного обмена сообщениями с точки зрения безопасности обработки информации. Отмечено, что всем таким информационным системам присущ общий недостаток — потенциальная возможность компрометации ее центрального узла владельцем ИС (с согласия владельца) или злоумышленником. Такая потенциальная возможность существует в силу особенностей инфраструктуры ключей шифрования, предполагающей участие центрального узла во всех информационных обменах ключевой информации. Компрометацию ключей шифрования могут устранить механизмы защиты передаваемой информации.

В статье описан способ реализации ИС для конфиденциального обмена информацией, который заключается в частичной децентрализации инфраструктуры ключевого обмена, что, в свою очередь, в значительной степени снижает угрозу нарушения конфиденциальности информации даже в случае компрометации центрального узла ИС. В работе предложена модель ИС, подробно описана инфраструктура и алгоритмы распределения ключей шифрования, которые построены на основе российских стандартов в области криптографии, а также предложены протоколы сетевого взаимодействия. Отмечено, что предлагаемый подход и модель ИС реализованы в виде программного комплекса *ruMessanger* [3], технологические особенности реализации которого представлены в работе. Посредством тестирования данного программного комплекса были получены экспериментальные оценки его производительности. Отмечено, что ИС в монолитном исполнении серверной части не способна нести высокую нагрузку, которая присуща системам мгновенного обмена информацией с очень большим числом пользователей. Как следствие, в работе предложен подход к масштабированию архитектуры центрального узла, который в настоящее время находится на стадии реализации.

Дальнейшее направление исследований связано с комплексной оценкой других аспектов безопасности используемых информационных технологий в целях устранения потенциальных угроз, которые могут иметь место в разработанной ИС в контексте иных моделей нарушителя. Кроме того, актуальными представляются вопросы оптимизации архитектуры ИС в контексте задачи по ее масштабированию.

Статья подготовлена при поддержке Красноярского краевого фонда поддержки научной и научно-технической деятельности.

Список литературы

1. **About** WhatsApp. URL: https://www.whatsapp.com/about/
2. **Schneier B.** Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C. John Wiley & Sons, 2and edition, 1996. 784 p.
3. **Шниперов А. Н., Чистяков А. П.** Программный комплекс для конфиденциального обмена информацией в компьютерных сетях общего доступа "ruMessanger". Свидетельство о государственной регистрации программы для ЭВМ. № 2017610542. Зарегистрировано 12.01.2017.
4. **WhatsApp Encryption Overview.** Technical white paper. November 17, 2016. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf
5. **Perrin T.** The Noise Protocol Framework. October 7, 2016. URL: http://noiseprotocol.org/noise.html
6. **McGrew D. A., Viega J.** The Galois/Counter Mode of Operation (GCM). May 2005. URL: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf
7. **Cohn-Gordon K., Cremers C., Dowling B., et al.** Formal Security Analysis of the Signal Messaging Protocol. Cryptology ePrint Archive, Report 2016/1013, 2016. URL: https://eprint.iacr.org/2016/1013.pdf
8. **Perrin T., Marlinspike M.** The Double Ratchet Algorithm. November 20, 2016. URL: https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf
9. **Евдокимов Д.** Безопасность мобильного банкинга: возможность реализации атаки "MITM" // Digital Security — 2014. URL: http://dsec.ru/upload/medialibrary/56e/56e70f90cbcc8c092f036d8005351fd9.pdf
10. **Telegram.** FAQ for the Technically Inclined. URL: https://core.telegram.org/techfaq
11. **Gligor V., Donescu P.** On Message Integrity in Symmetric Encryption// In Proc. 1st NIST Workshop on AES Modes of Operation. November 10, 2000. URL: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ige/ige-spec.pdf
12. **Viber Encryption Overview**. URL: https://www.viber.com/en/security-overview
13. **Bernstein D. J.** Salsa20 specification. URL: https://cr.yp.to/snuffle/spec.pdf
14. **Threema Cryptography Whitepaper**. February 15, 2017. URL: https://threema.ch/press-files/2_documentation/cryptography_whitepaper.pdf
15. **Bernstein D. J.** Extending the Salsa20 nonce// In Workshop record of Symmetric Key Encryption Workshop (2011), vol. 2011. URL: https://cr.yp.to/snuffle/xsalsa-20110204.pdf
16. **Wind D**. Man-in-the-middle attack on TextSecure // University of Applied Science St. Pölten. IT-Security Community Xchange (ITSeCX), 2015. URL: https://www.youtube.com/watch?v=bSap-VI4oh8
17. **Jakobsen J., Orlandi C.** A practical cryptanalysis of the Telegram messaging protocol. PhD thesis, Master Thesis, Aarhus University (Available on request), September 2015. URL: https://cs.au.dk/~jakjak/master-thesis.pdf
18. **ГОСТ Р 34.12—2015.** Информационная технология. Криптографическая защита информации. Блочные шифры. М.: Стандартинформ, 2015. 25 с. URL: https://tc26.ru/standard/gost/GOST_R_3412-2015.pdf
19. **ГОСТ Р 34.13—2015.** Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров. М.: Стандартинформ, 2015., 42 с. URL: https://www.tc26.ru/standard/gost/GOST_R_3413-2015.pdf
20. **ГОСТ Р 34.11—2012.** Информационная технология. Криптографическая защита информации. Функция хэширования. М.: Стандартинформ, 2012. 38 с. URL: http://specremont.su/pdf/gost_34_11_ 2012.pdf
21. **WebCrypto GOST** Library official site. URL: http://gostcrypto.com
22. **The PeerJS** library official site. URL: http://peerjs.com
23. **Banker K., Bakkum P., Verch S., et al.** MongoDB in Action: Covers MongoDB version 3.0, 2nd Edition. AK Peters/CRC Press, 2016. 480 p.
24. **The Apache JMeter** official site. URL: http://jmeter.apache.org
25. **Hare D.** Performance testing and analysis with WebSphere Application Server. August 1, 2012. URL: http://www.ibm.com/developerworks/websphere/techjournal/1208_hare/1208_hare.html?S_TACT=105AGX99&S_CMP=CP
26. **Brown E.** Web Development with Node and Express. O'Reilly Media, July 2014. 332 p.
27. **Nginx** official site. URL: https://www.nginx.com
28. **Redis** official site. URL: https://redis.io
29. **Oracle** official site. URL: http://www.oracle.com/us/technologies/linux/025995.htm

The Method and Information System for the Exchange of Confidential Information in Open Computer Networks

A. N. Shniperov, ashniperov@sfu-kras.ru, **A. P. Chistyakov,** acella93@mail.ru, Siberian Federal University, Krasnoyarsk, 660074, Russian Federation

Corresponding author:

Shniperov Alexey N., Head of Information Security Laboratory, Siberian Federal University, Krasnoyarsk, 660074, Russian Federation,
E-mail: ashniperov@sfu-kras.ru

Received on April 26, 2017
Accepted on May 29, 2017

Submitted article considers the problem of protecting confidential information in open computer networks. The paper views the method and information system for instant messaging of confidential information in networks. The method is based on the idea of sharing cryptographic keys and distributing them through various network channels. The article reviews and analyzes the existing instant messaging applications including WhatsApp, Viber, Telegram, Threema in terms of security of information exchange. It is concluded that all the systems are subject to a classic MITM-attack on the key information coming from the server side. For example, this is possible for the government agencies with access to the server. The article proposes a model of the information system which basically solves this

problem by dividing the shared secret via several communication channels. The Infrastructure of the encryption keys is described in detail, including the algorithms for their distribution. We also consider cryptographic primitives, which we used. The article considers the protocols of the network interaction of subscribers in the information system. In addition, the article describes the development technology, as well as the development tools which were used. The article deals with the architecture of the information system in terms of its load capacity and its scalability.

Keywords: *information security, cryptoprotocols, telecommunication security, MITM-attack, shared secret key, end-to-end encryption, instant messaging applications, fault tolerance, security network protocol*

Acknowledgements: *This work was supported by the Krasnoyarsk Regional Foundation for Support of Scientific and Technical Activity.*

For citation:

Shniperov A. N., **Chistyakov A. P.** The Method and Information System for the Exchange of Confidential Information in Open Computer Networks, *Programmnaya Ingeneria,* 2017, vol. 8, no 8, pp. 359—368.

DOI: 10.17587/prin.8.359-368

References

1. **About WhatsApp**, available at: https://www.whatsapp.com/about/

2. **Schneier B.** *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C.,* John Wiley & Sons, 1996, 784 p.

3. **Shniperov A. N., Chistyakov A. P.** Programmnyj kompleks dlja konfidencial'nogo obmena informaciej v komp'juternyh setjah obshhego dostupa "ruMessanger" (The application for confidential information exchange in open computer networks "ruMessanger"), Svidetel'stvo o gosudarstvennoj registracii programmy dlja JeVM. № 2017610542. 12.01.2017 (in Russian).

4. **WhatsApp Encryption Overview**. Technical white paper. November 17, 2016, available at: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.

5. **Perrin T.** The Noise Protocol Framework. October 7, 2016, available at: http://noiseprotocol.org/noise.html

6. **McGrew D. A., Viega J.** The Galois/Counter Mode of Operation (GCM), May 2005, available at: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

7. **Cohn-Gordon K., Cremers C., Dowling B., Garratt L., Stebila D.** A Formal Security Analysis of the Signal Messaging Protocol, Cryptology ePrint Archive, Report 2016/1013, 2016, available at: https://eprint.iacr.org/2016/1013.pdf

8. **Perrin T., Marlinspike M.** The Double Ratchet Algorithm, November 20, 2016, available at: https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf

9. **Evdokimov D.** Bezopasnost' mobil'nogo bankinga: vozmozhnost' realizacii ataki "MITM" (Security of the mobile banking: the possibility of MITM-attack), *Digital Security*, 2014, available at: http://dsec.ru/upload/medialibrary/56e/56e70f90cbcc8c092f036d8005351fd9.pdf

10. **Telegram.** FAQ for the Technically Inclined, available at: https://core.telegram.org/techfaq.

11. **Gligor V., Donescu P.** On Message Integrity in Symmetric Encryption, *Proc. 1st NIST Workshop on AES Modes of Operation*, November 10, 2000, available at: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ige/ige-spec.pdf

12. **Viber Encryption Overview**, available at: https://www.viber.com/en/security-overview

13. **Bernstein D. J.** Salsa20 specification, available at: https://cr.yp.to/snuffle/spec.pdf

14. **Threema Cryptography Whitepaper**, February 15, 2017, available at: https://threema.ch/press-files/2_documentation/cryptography_whitepaper.pdf

15. **Bernstein D. J**. Extending the Salsa20 nonce, *In Workshop record of Symmetric Key Encryption Workshop*, 2011, available at: https://cr.yp.to/snuffle/xsalsa-20110204.pdf

16. **Wind D.** Man-in-the-middle attack on TextSecure, *University of Applied Science St. Polten. IT-Security Community Xchange (IT-SeCX)*, 2015, available at: https://www.youtube.com/watch?v=bSap-VI4oh8.

17. **Jakobsen J., Orlandi C.** A practical cryptanalysis of the Telegram messaging protocol, PhD thesis, Master Thesis, Aarhus University, 2015, available at: https://cs.au.dk/~jakjak/master-thesis.pdf

18. **GOST R 34.12—2015.** Informacionnaya tehnologiya. Kriptograficheskaja zashhita informacii. Blochnye shifry (Information Technology. Cryptographic protection of information. The block ciphers), Moscow, Federal agency on technical regulation and metrology, 2015, available at: https://tc26.ru/standard/gost/GOST_R_3412-2015.pdf (in Russian).

19. **GOST R 34.13—2015.** Informacionnaya tehnologiya. Kriptograficheskaja zashhita informacii. Rezhimy raboty blochnyh shifrov (Information Technology. Cryptographic protection of information. The modes of Block ciphers), Moscow, Federal agency on technical regulation and metrology, 2015, available at: https://www.tc26.ru/standard/gost/GOST_R_3413-2015.pdf (in Russian).

20. **GOST R 34.11—2012.** Informacionnaya tehnologiya. Kriptograficheskaja zashhita informacii. Funkcija hjeshirovanija (Information Technology. Cryptographic protection of information. The Hash Function), Moscow, Federal agency on technical regulation and metrology, 2012, available at: http://specremont.su/pdf/gost_34_11_ 2012.pdf (in Russian).

21. **WebCrypto GOST** Library official site, available at: http://gostcrypto.com

22. **The PeerJS** library official site, available at: http://peerjs.com

23. **Banker K., Bakkum P., Verch S., Garrett D., Hawkins T.** *MongoDB in Action: Covers MongoDB version 3.0*, 2nd Edition, A K Peters/CRC Press, 2016, 480 p.

24. **The Apache JMeter** official site, available at: http://jmeter.apache.org

25. **Hare D.** Performance testing and analysis with WebSphere Application Server, IBM Corporation, August 1, 2012, available at: http://www.ibm.com/developerworks/websphere/techjournal/1208_hare/1208_hare.html?S_TACT=105AGX99&S_CMP=CP.

26. **Brown E.** *Web Development with Node and Express*, O'Reilly Media, July 2014, 332 p.

27. **Nginx** official site, available at: https://www.nginx.com

28. **Redis** official site, available at: https://redis.io

29. **Oracle** official site, available at: http://www.oracle.com/us/technologies/linux/025995.htm

УДК 517.98 DOI: 10.17587/prin.8.369-384

П. Н. Бибило, д-р техн. наук, проф., зав. лаб., e-mail: bibilo@newman.bas-net.by,
Ю. Ю. Ланкевич, мл. науч. сотр., Объединенный институт проблем информатики Национальной академии наук Беларуси, г. Минск

Использование полиномов Жегалкина при минимизации многоуровневых представлений систем булевых функций на основе разложения Шеннона

Предложено минимизировать многоуровневые представления систем булевых функций на основе разложения Шеннона с учетом нахождения одинаковых (с точностью до инверсии) подфункций разложения и использовать для этих целей представления функций в виде полиномов Жегалкина. Программа, реализующая предложенные алгоритмы, позволила получать лучшие результаты синтеза функциональных блоков заказных сверхбольших интегральных схем, чем результаты синтеза по минимизированным дизъюнктивным нормальным формам систем функций и минимизированным разложениям Шеннона, выполняемым без нахождения инверсий подфункций.

Ключевые слова: *синтез логических схем, минимизация систем булевых функций, разложение Шеннона, полином Жегалкина*

Введение

Синтез схем комбинационной логики остается попрежнему актуальной задачей автоматизированного проектирования цифровых заказных СБИС (сверхбольших интегральных схем). Традиционно он разбивается на два этапа — технологически независимую оптимизацию и технологическое отображение в заданный базис логических элементов. Решающее значение имеет первый этап, на котором проводится минимизация различных форм представлений булевых функций. Как показала практика проведения синтеза схем в промышленных синтезаторах, например, в *LeonardoSpectrum*, изменение формы задания одной и той же системы булевых функций может значительно изменить результаты синтеза, а именно площадь схемы, ее быстродействие и энергопотребление [1]. Если функции системы заданы в виде ДНФ (дизъюнктивных нормальных форм), то при синтезе схем из библиотечных логических элементов целесообразно перейти к многоуровневым представлениям на основе разложения Шеннона. Графическая форма таких представлений названа в литературе *BDD* (*Binary Decision Diagram*) [2—6]. В отечественной литературе *BDD* называется диаграммой двоичного выбора, диаграммой двоичных решений, двоичной (бинарной) разрешающей диаграммой. Синтез схем по логическим уравнениям, соответствующим минимизированным *BDD*-представлениям, дает, как правило, лучшие результаты по сравнению с синтезом по раздельно либо совместно минимизированным системам ДНФ [7, 8].

В данной работе предлагается минимизировать многоуровневые представления систем булевых функций на основе разложения Шеннона с учетом нахождения одинаковых подфункций (с точностью до инверсии) и использовать для этих целей представления функций в виде полиномов Жегалкина. Полиномы Жегалкина легко сравнивать, при этом просто получать полиномы, реализующие инверсные функции, а это значительно ускоряет время вычислений во многих случаях. Проведенные вычислительные эксперименты показали, что для уменьшения площади и увеличения быстродействия комбинационных логических схем из библиотечных КМОП-элементов лучше вести синтез по минимизированным многоуровневым представлениям систем функций в виде формул разложений Шеннона с использованием инверсий подфункций, чем по многоуровневым представлениям, не использующим инверсии подфункций.

1. Формы задания булевых функций

1.1. Дизъюнктивные нормальные формы

Булевыми называются двоичные (0, 1) функции $f(\mathbf{x}) = f(x_1, x_2, ..., x_n)$ двоичных переменных x_1, x_2, ..., x_n. Широко распространенной формой задания булевой функции является ДНФ, которая обычно представляется троичной матрицей, например, ДНФ D^1 булевой функции $f^1 = x_1x_2x_3\bar{x}_4 \vee \bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1x_2x_4$ может быть представлена троичной матрицей (табл. 1), строки которой соответствуют элементарным конъюнкциям. В табл. 1 третий троичный вектор (0 1—1) задает элементарную конъюнкцию $\bar{x}_1x_2x_4$: положительные литералы x_2, x_4 представляются единицами в троичном векторе, отрицательный литерал $\bar{x}_1$ — нулем, отсутствующий литерал перемен-

Таблица 1

ДНФ D^1 булевой функции f^1

D^1				k_i
x_1	x_2	x_3	x_4	
1	1	1	0	k_1
—	0	0	0	k_2
0	1	—	1	k_3

Таблица 2

Характеристическое множество M^{f^1} булевой функции f^1

M^{f^1}			
x_1	x_2	x_3	x_4
1	1	1	0
0	0	0	0
1	0	0	0
0	1	0	1
0	1	1	1

ной x_3 обозначается символом "—". Характеристическое множество M^{f^1} булевой функции, заданной ДНФ (табл. 1), дано в табл. 2. Элементарные конъюнкции k_i, k_j, $(i \neq j)$ называются *ортогональными*, если $k_i k_j = 0$, если же $k_i k_j \neq 0$, то конъюнкции k_i, k_j называются неортогональными. Будем говорить, что троичные векторы $\mathbf{a} = (a_1, \ldots, a_m)$, $\mathbf{b} = (b_1, \ldots, b_m)$ ортогональны (*ort*), если найдется хотя бы одна компонента $i \in \{1, \ldots, m\}$, для которой a_i, b_i определены и не равны. Например, троичные векторы $\mathbf{a} = (0 - 10)$, $\mathbf{b} = (-100)$ ортогональны, так как для $i = 3$ выполняется условие ортогональности: $a_3 = 1$, $b_3 = 0$. ДНФ называется *ортогонализованной*, если ортогональной является каждая пара k_i, k_j, $(i \neq j)$ элементарных конъюнкций, из которых она состоит. Например, ДНФ (см. табл. 1), является ортогонализованной.

Под векторной булевой функцией $\mathbf{f}(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, будем понимать упорядоченную систему булевых функций $\mathbf{f}(\mathbf{x}) = \left(f^1(\mathbf{x}) \ldots, f^m(\mathbf{x})\right)$. Широко известной в литературе [1, 8, 9] формой представления систем булевых функций является пара матриц $< T^x, B^f >$: троичная матрица T^x задания элементарных конъюнкций в виде троичных векторов и булева матрица B^f вхождений конъюнкций в ДНФ компонентных функций системы. В табл. 3 дан пример задания системы ДНФ векторной полностью определенной функции $\mathbf{f}(\mathbf{x}) = \left(f^1(\mathbf{x}), f^2(\mathbf{x}), f^3(\mathbf{x})\right)$:

$$f^1 = x_1 x_2 x_3 \bar{x}_4 \vee \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 x_4;$$

$$f^2 = \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee x_1 x_2 \bar{x}_3 x_4 \vee \vee \bar{x}_1 x_2 x_4 \vee x_3 x_4 \vee x_2 x_3;$$

$$f^3 = \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee x_1 x_2 \bar{x}_3 x_4 \vee x_1 x_2 x_3 \bar{x}_4 \vee \bar{x}_1 x_2 \bar{x}_4.$$

ДНФ, представляющие функции f^1, f^3, являются ортогонализованными, что нельзя сказать о ДНФ, представляющей функцию f^2 — последние три конъюнкции из левой части табл. 3 являются неортогональными.

1.2. Полиномы Жегалкина

Рассмотрим многоместную операцию $x_1 \oplus x_2 \oplus \oplus \ldots \oplus x_n$ "сумма по модулю 2". Данную операцию называют также "исключающее ИЛИ". Значение этого выражения равно 1 тогда и только тогда, когда

Таблица 3

Пример системы ДНФ трех булевых функций

T^x				B^f		
x_1	x_2	x_3	x_4	f^1	f^2	f^3
0	0	0	1	0	1	1
1	0	0	0	0	1	1
1	1	0	1	0	1	1
1	1	1	0	1	0	1
—	0	0	0	1	0	0
0	1	—	0	0	0	1
0	1	—	1	1	1	0
—	—	1	1	0	1	0
—	1	1	—	0	1	0

в наборе значений переменных имеется нечетное число единиц. Например, $1 \oplus 0 \oplus 0 = 1$, $0 \oplus 0 \oplus 0 = 1$. *Полиномом* называется сумма по модулю 2 произвольных элементарных конъюнкций. *Полином Жегалкина* — это многоместная сумма по модулю 2 попарно различных положительных элементарных конъюнкций, т. е. элементарных конъюнкций, не содержащих инверсных литералов. Длиной полинома называется число входящих в него элементарных конъюнкций. Полином Жегалкина является канонической формой задания булевой функции — он является *единственным* для любой булевой функции [9].

1.3. Многоуровневое разложение Шеннона с нахождением инверсных подфункций

Еще одной используемой в данной работе формой задания систем булевых функций являются алгебраические многоуровневые представления на базе разложения Шеннона с использованием инверсных подфункций.

Разложением Шеннона полностью определенной булевой функции $f = f(\mathbf{x})$ по переменной x_i называется представление

$$f = f(\mathbf{x}) = \bar{x}_i f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \vee \vee x_i f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n). \quad (1)$$

Функции $f_0 = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$, $f_1 = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ в правой части (1) называются коэффициентами разложения по переменной x_i [9], остаточными подфункциями, либо просто *подфункциями*. Они получаются из функции $f = f(x_1, \ldots, x_n)$ подстановкой вместо переменной x_i константы 0 и 1 соответственно. Каждая из подфункций $f_0 = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ и $f_1 = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ может быть разложена по одной из переменных из множества $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$. Процесс разложения подфункций заканчивается, когда все n переменных будут использованы для разложения, либо когда все подфункции выродятся до констант 0, 1. На каждом

шаге разложения выполняется сравнение на равенство полученных подфункций и оставляется одна из нескольких попарно равных (с *точностью до инверсии*) подфункций. Если же подфункции разложения по переменной x_i равны, т. е.

$$f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n),$$

то переменная x_i называется *несущественной* (фиктивной), и $f = f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

Под *BDDI* (***B**inary **D**ecision **D**iagram with **I**nverse cofactors*) в данной работе понимается ориентированный ациклический граф, задающий последовательные разложения Шеннона булевой функции $f(x_1, \ldots, x_n)$ по всем ее переменным $x_1, \ldots, x_n$ при заданном порядке (перестановке) переменных, по которым проводятся разложения. Граф *BDDI* одной полностью определенной булевой функции содержит три вида вершин: функциональные вершины, соответствующие разлагаемым функциям и подфункциям (и их инверсиям); вершины-переменные; листовые вершины, соответствующие константам 0, 1. Ориентация дуг обычно не показывается, так как при изображении *BDDI* все дуги ориентируются сверху вниз.

Функциональная вершина *BDDI* (рис. 1) реализует одну функцию φ (подфункцию) либо две функции (подфункцию φ и ее инверсию $\overline{\varphi}$).

Смысл высказывания "с точностью до инверсии" заключается в том, что любую из пары взаимно инверсных подфункций разложения можно считать неинверсной подфункцией, тогда другая подфункция такой пары будет инверсной к выбранной.

Функциональная вершина, соответствующая функции *f*, называется корнем. Если подфункции разложения (1) равны, то граф *BDDI* упрощается, так как вершина-переменная x_i, из которой исходит одна дуга, удаляется из графа *BDDI*. Граф *BDDI*, представляющий систему *m* полностью определенных булевых функций, имеет *m* корневых и две листовые вершины 0, 1, которые обычно дублируются для упрощения изображения графа. Наиболее близкими к графам *BDDI* для систем функций являются широко известные в литературе сокращенные упорядоченные *BDD* (англ. *Reduced Ordered BDD, ROBDD*) для одной булевой функции, в которых каждой функциональной вершине соответствует одна функция (подфункция разложения Шеннона), при этом функциональные вершины лишь подразумеваются (отождествляются с вершинами-переменными). В *ROBDD* паре взаимно инверсных подфункций соответствует пара вершин-переменных. Подробное описание *OBDD* (упорядоченных *BDD*) дано в работе [4], *ROBDD* — в работе [6]. Далее под *BDD* будут пониматься *ROBDD* для систем функций, если это не оговорено особо. Переход от *BDDI* к *BDD* будет рассмотрен далее. Под *сложностью* S_{BDDI} будем понимать число функциональных вершин *BDDI*. При реализации *BDDI* логическими схемами из мультиплексоров *MS* (это мультиплексоры (рис. 2) с одним управляющим и двумя информационными входами и двумя взаимно инверсными выходами) каждой функциональной вершине *BDDI* соответствует один мультиплексор, а всему графу *BDDI* — каскадная логическая схема.

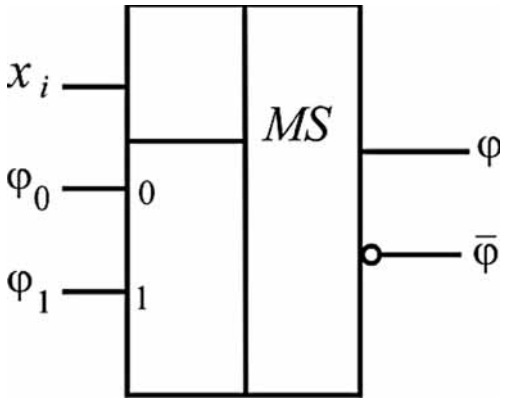

Рис. 2. Мультиплексор *MS* с одним управляющим входом x_i

Отметим, что каждая вершина *BDD* реализуется одним мультиплексором, имеющим только один выход (такой мультиплексор не имеет инверсного выхода). Заметим, что представление многоуровневого разложения системы булевых функций в виде графа *BDDI* в данной статье используется для иллюстрации зависимостей булевых переменных (и функций) в логических выражениях. Целью является получение минимального числа логических выражений, по которым впоследствии осуществляется синтез логической схемы в заданном базисе (библиотеке) логических элементов. Функциональные вершины нижнего (неконстантного) уровня *BDDI* равны переменной либо ее инверсии. В уравнениях вида (1) такие функции не будут записываться, поэтому число уравнений многоуровневого представления обычно меньше сложности *BDDI* на одну либо две функциональные вершины.

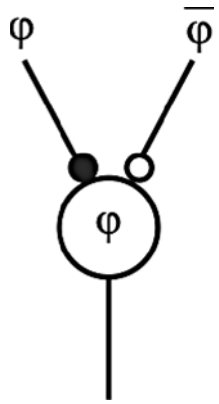

Рис. 1. Функциональная вершина *BDDI*

2. Постановка задачи

Граф *BDDI* задает последовательность разложений Шеннона исходной функции и получаемых подфункций разложения по некоторой перестановке переменных. Минимизация сложности *BDDI*, также как и *BDD*, основана на том, что в процессе разложения могут появляться одинаковые подфункции разложения не только у одной, но и у нескольких (либо даже у всех) компонентных функций. Давно известно, что от перестановки переменных, по которым ведется разложение, зависит сложность *BDD*, поэтому основной задачей компактного представления булевой функции либо системы функций в виде *BDDI* также является нахождение перестановки, дающей минимальное число функциональных вершин *BDDI*.

Задача. Для заданной системы ДНФ булевых функций найти такую перестановку $< x_{i_1}, x_{i_2}, \ldots, x_{i_n} >$ переменных, по которой будет получен граф *BDDI* минимальной сложности.

Сложность *BDD* и *BDDI* зависит от порядка следования (перестановки) переменных разложения. Так как число всех перестановок переменных для булевой функции $f = f(x_1, x_2, \ldots, x_n)$, зависящей от n переменных, равно $n!$ (факториалу числа n), то нахождение лучшей перестановки, т. е. перестановки, обеспечивающей минимальную сложность *BDD* и *BDDI* для больших (сотни и более) значений числа n, представляет собой чрезвычайно трудоемкую задачу. Поэтому основными методами нахождения лучшей перестановки, предложенными в литературе, являются методы "ветвей и границ" и методы генерирования случайных перестановок, дополненные различными эвристиками [10]. Полный перебор всех $n!$ перестановок осуществим для небольших значений n, например, в работе [1] были испытаны все $n!$ перестановок переменных для примеров (*benchmarks*) систем ДНФ с числом переменных $n \leqslant 8$. Для полиномов Жегалкина легко получить инверсию функции $(\overline{f} = f \oplus 1)$, поэтому при решении задачи исходные ДНФ заменяют полиномами Жегалкина, по которым и строят многоуровневые разложения Шеннона — так значительно облегчается поиск взаимно инверсных подфункций. Переход от ДНФ к полиному Жегалкина известен.

3. Алгоритм построения полинома Жегалкина по ортогонализованной ДНФ

Алгоритм состоит из трех шагов.

Шаг 1. В ДНФ заменяем каждый символ "$\vee$" дизъюнкции символом "$\oplus$".

Шаг 2. Заменяем каждый инверсный литерал $\overline{x}_i$ равносильным выражением $\overline{x}_i = x_i \oplus 1$.

Шаг 3. Раскрываем скобки, используя формулы $x_i \oplus x_i = 0$; $x_i \oplus 0 = x_i$; $0 \oplus 0 = 0$; $1 \oplus 1 = 0$, и получаем полином Жегалкина.

Используя данный алгоритм, построим полиномы Жегалкина для ортогонализованных ДНФ, представляющих функции f^1 и f^3:

$$f_1 = x_1x_2x_3\overline{x}_4 \vee \overline{x}_2\overline{x}_3\overline{x}_4 \vee \overline{x}_1x_2x_4 =$$
$$= x_1x_2x_3\overline{x}_4 \oplus \overline{x}_2\overline{x}_3\overline{x}_4 \oplus \overline{x}_1x_2x_4 = x_1x_2x_3(x_4 \oplus 1) \oplus$$
$$\oplus (x_2 \oplus 1)(x_3 \oplus 1)(x_4 \oplus 1) \oplus (x_1 \oplus 1)x_2x_4 =$$
$$= x_1x_2x_3x_4 \oplus x_2x_3x_4 \oplus x_3x_4 \oplus x_1x_2x_4 \oplus x_4 \oplus$$
$$\oplus x_1x_2x_3 \oplus x_2x_3 \oplus x_3 \oplus x_2 \oplus 1.$$

$$f^3 = \overline{x}_1\overline{x}_2\overline{x}_3x_4 \vee x_1\overline{x}_2\overline{x}_3\overline{x}_4 \vee x_1x_2\overline{x}_3x_4 \vee x_1x_2x_3\overline{x}_4 \vee \overline{x}_1x_2\overline{x}_4 =$$
$$= \overline{x}_1\overline{x}_2\overline{x}_3x_4 \oplus x_1\overline{x}_2\overline{x}_3\overline{x}_4 \oplus x_1x_2\overline{x}_3x_4 \oplus x_1x_2x_3\overline{x}_4 \oplus \overline{x}_1x_2\overline{x}_4 =$$
$$= (x_1 \oplus 1)(x_2 \oplus 1)(x_3 \oplus 1)x_4 \oplus x_1(x_2 \oplus 1)(x_3 \oplus 1)(x_4 \oplus 1) \oplus$$
$$\oplus x_1x_2(x_3 \oplus 1)x_4 \oplus x_1x_2x_3(x_4 \oplus 1) \oplus (x_1 \oplus 1)x_2(x_4 \oplus 1) =$$
$$= x_2x_3x_4 \oplus x_3x_4 \oplus x_4 \oplus x_1x_3 \oplus x_2 \oplus x_1.$$

Таблица 4

Задание полиномов Жегалкина в матричной форме

x_1	x_2	x_3	x_4	f^i
1	1	1	1	
—	1	1	1	
—	—	1	1	
1	1	—	1	
—	—	—	1	
1	1	1	—	f^1
—	1	1	—	
—	—	1	—	
—	1	—	—	
—	—	—	—	
—	1	1	1	
—	—	—	1	
1	1	1	—	
—	1	1	—	f^2
1	—	1	—	
1	1	—	—	
1	—	—	—	
—	1	1	1	
—	—	1	1	
—	—	—	1	
1	—	1	—	f^3
—	1	—	—	
1	—	—	—	

Матричные формы полиномов Жегалкина для функций f^1, f^2, f^3 даны в табл. 4.

4. Алгоритм построения полинома Жегалкина по неортогонализованной ДНФ

Идея алгоритма основывается на итеративном применении формулы

$$k_i \vee k_j = k_i \oplus k_j \oplus k_ik_j, \qquad (2)$$

где k_i, k_j — конъюнкции ДНФ, преобразуемой в полином Жегалкина.

Алгоритм построения полинома Жегалкина по ДНФ булевой функции состоит в том, что конъюнкции исходной ДНФ разбивают на пары $k_i \vee k_j$, которые затем заменяют выражениями $k_i \oplus k_j \oplus k_ik_j$, после чего к полученным выражениям итеративно применяют формулу (2). Итерации выполняют до тех пор, пока не будут исключены все операции дизъюнкции, причем после каждого перемножения в полученных выражениях удаляют одинаковые конъюнкции по правилу $k_i \oplus k_i = 0$. Затем каждый инверсный литерал $\overline{x}_i$ заменяется равносильным выражением $\overline{x}_i = x_i \oplus 1$, опять выполняются перемножения, приводятся подобные и получается полином Жегалкина.

Проиллюстрируем данный алгоритм на примере получения полинома Жегалкина для ДНФ функции f^2. Применим формулу (2):

$$f^2 = \bar{x}_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1x_2\bar{x}_3x_4 \vee \bar{x}_1x_2x_4 \vee x_3x_4 \vee x_2x_3 =$$
$$= (\bar{x}_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4) \vee (x_1x_2\bar{x}_3x_4 \vee \bar{x}_1x_2x_4) \vee (x_3x_4 \vee x_2x_3) =$$
$$= (\bar{x}_1\bar{x}_2\bar{x}_3x_4 \oplus x_1\bar{x}_2\bar{x}_3\bar{x}_4) \vee (x_1x_2\bar{x}_3x_4 \oplus \bar{x}_1x_2x_4) \vee (x_3x_4 \oplus x_2x_3 \oplus x_2x_3x_4) =$$
$$= (\bar{x}_1\bar{x}_2\bar{x}_3x_4 \oplus x_1\bar{x}_2\bar{x}_3\bar{x}_4 \oplus x_1x_2\bar{x}_3x_4 \oplus \bar{x}_1x_2x_4) \vee (x_3x_4 \oplus x_2x_3 \oplus x_2x_3x_4) =$$
$$= \bar{x}_1\bar{x}_2\bar{x}_3x_4 \oplus x_1\bar{x}_2\bar{x}_3\bar{x}_4 \oplus x_1x_2\bar{x}_3x_4 \oplus \bar{x}_1x_2x_4 \oplus x_3x_4 \oplus x_2x_3 \oplus x_2x_3x_4 \oplus \bar{x}_1x_2x_3x_4.$$

Заменим инверсные литералы по формуле $\bar{x}_i = x_i \oplus 1$:

$$f^2 =$$
$$= (x_1 \oplus 1)(x_2 \oplus 1)(x_3 \oplus 1)x_4 \oplus x_1(x_2 \oplus 1)(x_3 \oplus 1)(x_4 \oplus 1) \oplus$$
$$\oplus x_1x_2(x_3 \oplus 1)x_4 \oplus (x_1 \oplus 1)x_2x_4 \oplus x_3x_4 \oplus x_2x_3 \oplus x_2x_3x_4 \oplus$$
$$\oplus (x_1 \oplus 1)x_2x_3x_4.$$

В правой части последней формулы выполним умножения, приведем подобные и получим полином Жегалкина:

$$f^2 = x_2x_3x_4 \oplus x_4 \oplus x_1x_2x_3 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2 \oplus x_1.$$

5. Алгоритм 1 минимизации *BDDI*

Предлагаемый эвристический алгоритм решения задачи является локально оптимальным, выполняется итеративно до тех пор, пока текущая система функций (подфункций), представленных полиномами Жегалкина, не выродится до констант 0, 1. На первой итерации в качестве текущей берется исходная система функций, причем ДНФ, представляющие функции системы, преобразованы в полиномы Жегалкина. На каждой итерации j (j = 1, ..., n) выполняется шаг 1 с целью нахождения очередной переменной, по которой надо провести разложения Шеннона, и шаг 2 получения подфункций по выбранной переменной.

Шаг 1. Выбор переменной x_j, по которой проводится разложение Шеннона всех функций текущей системы.

Шаг 1.1. Построение разложения Шеннона по каждой из переменных x_i каждой из функций текущей системы, т. е. получение подфункций разложений Шеннона в виде полиномов Жегалкина. Исключение из рассмотрения подфункций, равных 0, 1.

Шаг 1.2. Проверка полученных подфункций на равенство, в результате чего из всего множества подфункций, полученных при разложении по x_i, формируется множество M_i^j попарно различных (с точностью до инверсии) подфункций.

Шаг 1.3. Оценка переменных, от которых зависят функции текущей системы. Каждая переменная x_i текущей системы оценивается числом S_i^j — мощностью $\left|M_i^j\right| = S_i^j$ множества M_i^j *различных (с точностью до инверсии) подфункций*, полученных при разложении всех функций текущей системы по переменной x_i.

Шаг 1.4. В качестве переменной x_j для разложения Шеннона на итерации j выбирается переменная x_i, оцениваемая *минимальным* числом S_i^j. Если таких переменных несколько, то из них выбирается первая по порядку. Переход на шаг 2.

Шаг 2. Построение разложений Шеннона текущей системы функций и формирование системы функций для итерации $j+1$.

Шаг 2.1. Построение разложений Шеннона функций текущей системы (на итерации j) *по выбранной переменной* x_i. Исключение из рассмотрения подфункций, равных 0, 1. Если все подфункции являются константами, то переход на шаг 3.

Шаг 2.2. Проверка полученных подфункций на равенство, и из всего множества полиномов Жегалкина, представляющих подфункции, формирование множества M_i^j попарно различных (с точностью до инверсии) полиномов. Полиномы из множества M_i^j задают текущую систему функций для итерации $j+1$.

Шаг 3. Конец.

Описанный выше алгоритм использует **эвристику 1** — "*минимальное число различных (с точностью до инверсии) подфункций*" при выборе очередной переменной разложения и позволяет найти перестановку переменных для проведения разложений Шеннона по всем переменным. Если число выполненных итераций алгоритма оказалось меньше числа n, то это означает, что некоторые переменные оказались несущественными для каждой из функций исходной системы.

6. Алгоритм 2 минимизации *BDDI*

Данный алгоритм отличается от алгоритма 1 на шагах 1.4 и 1.5: в алгоритме 2 используются следующие правила (эвристики) выбора очередной переменной для разложения Шеннона:

Шаг 1.4. Оценка переменных, от которых зависят функции текущей системы. Каждая переменная x_i текущей системы оценивается числом Z_i^j — *суммой длин полиномов Жегалкина*, представляющих множество M_i^j различных (с точностью до инверсии) подфункций, полученных при разложении всех функций текущей системы по переменной x_i.

Шаг 1.5. В качестве переменной x_j для разложения Шеннона на итерации j выбирается переменная x_i, оцениваемая *минимальным* числом Z_i^j. Если таких переменных несколько, то из них выбирается первая по порядку. Переход на шаг 2.

Алгоритм 2 использует эвристику 2 — "*минимальная сумма длин полиномов Жегалкина*" при выборе очередной переменной x_j разложения.

7. Пример

Проиллюстрируем алгоритм построения графа *BDDI* на примере векторной функции $\mathbf{f}(\mathbf{x})$, состоящей из трех компонентных функций f^1, f^2, f^3 (см. табл. 3) и перестановки $< x_2, x_3, x_1, x_4 >$, именно такую перестановку находит алгоритм 1, используя эвристику 1.

Формулы разложения Шеннона по переменной x_3 записываются в следующем виде:

$$f^1(\mathbf{x}) = \bar{x}_2 h^2(x_1, x_3, x_4) \vee x_2 h^1(x_1, x_3, x_4);$$

$$f^2(\mathbf{x}) = \bar{x}_2 h^3(x_1, x_3, x_4) \vee x_2 \bar{h}^2(x_1, x_3, x_4);$$

$$f^3(\mathbf{x}) = \bar{x}_2 h^4(x_1, x_3, x_4) \vee x_3 \bar{h}^3(x_1, x_3, x_4).$$

Подфункции, входящие в данные формулы, даны в табл. 5, им соответствуют функциональные вершины второго уровня *BDDI*, на первом уровне располагаются корневые вершины, реализующие исходные функции (рис. 3).

Получение подфункций в виде полиномов Жегалкина проиллюстрируем на следующем примере:

$$f^1(x_1, 1, x_3, x_4) = x_1 1 x_3 x_4 \oplus 1 x_3 x_4 \oplus x_3 x_4 \oplus x_1 1 x_4 \oplus$$
$$\oplus x_4 \oplus x_1 1 x_3 \oplus 1 x_3 \oplus x_3 \oplus 1 \oplus 1 = x_1 x_3 x_4 \oplus x_3 x_4 \oplus x_3 x_4 \oplus$$
$$\oplus x_1 x_4 \oplus x_4 \oplus x_1 x_3 \oplus x_3 \oplus x_3 \oplus 1 \oplus 1 =$$
$$= x_1 x_3 x_4 \oplus (x_3 x_4 \oplus x_3 x_4) \oplus x_1 x_4 \oplus x_4 \oplus x_1 x_3 \oplus$$
$$\oplus (x_3 \oplus x_3) \oplus (1 \oplus 1) = x_1 x_3 x_4 \oplus 0 \oplus x_1 x_4 \oplus x_4 \oplus$$
$$\oplus x_1 x_3 \oplus 0 \oplus 0 = x_1 x_3 x_4 \oplus x_1 x_4 \oplus x_4 \oplus x_1 x_3.$$

При программной реализации подстановка констант и приведение подобных осуществляется на матричных представлениях полиномов Жегалкина, причем символы "–" заменяются нулями, а элементарные конъюнкции упорядочиваются согласно возрастанию их численного представления.

Функция h^2 зависит только от переменных x_3, x_4. Подфункции разложения по переменной x_3 даны в табл. 6.

Функция h^7 зависит только от переменной x_4. Разложение подфункций h^5, h^6 по переменной x_1 приводит к следующим формулам многоуровневого разложения Шеннона:

$$f^1 = \bar{x}_2 h^2 \vee x_2 h^1;\ f^2 = \bar{x}_2 h^3 \vee x_2 \bar{h}^2;\ f^3 = \bar{x}_2 h^4 \vee x_2 \bar{h}^3;$$

$$h^1 = \bar{x}_3 h^5 \vee x_3 h^6;\ h^2 = \bar{x}_3 \bar{h}^7;\ h^3 = \bar{x}_3 h^6 \vee x_3 h^7;\ h^4 = \bar{x}_3 h^6;$$

$$h^5 = \bar{x}_1 h^7;\ h^6 = \bar{x}_1 h^7 \vee x_1 \bar{h}^7;$$

$$h^7 = x_4.$$

Граф *BDDI* (рис. 3) задает функциональную зависимость подфункций, входящих в многоуровневое разложение Шеннона системы функций (см. табл. 3).

На рис. 4 показана логическая схема в базисе мультиплексоров, реализующая многоуровневое разложение Шеннона этой же системы функций. Мультиплексор,

Таблица 5

Подфункции разложения системы ДНФ по переменной x_2

Подфункции разложения по переменной x_2	Полином Жегалкина			
	Матричное представление			Формула
	x_1	x_3	x_4	
$h^1 = h^1(x_1, x_3, x_4) = f^1(x_1, 1, x_3, x_4)$	1	1	1	$x_1x_3x_4 \oplus x_1x_4 \oplus x_4 \oplus x_1x_3$
	1	–	1	
	–	–	1	
	1	1	–	
$h^2 = h^2(x_1, x_3, x_4) = f^1(x_1, 0, x_3, x_4)$	–	1	1	$x_3x_4 \oplus x_4 \oplus x_3 \oplus 1$
	–	–	1	
	–	1	–	
	–	–	–	
$\bar{h}^2 = h^2 \oplus 1 = f^2(x_1, 1, x_3, x_4)$	–	1	1	$x_3x_4 \oplus x_4 \oplus x_3$
	–	–	1	
	–	1	–	
$h^3 = h^3(x_1, x_3, x_4) = f^2(x_1, 0, x_3, x_4)$	–	–	1	$x_4 \oplus x_1x_3 \oplus x_1$
	1	1	–	
	1	–	–	
$\bar{h}^3 = h^3 \oplus 1 = f^3(x_1, 1, x_3, x_4)$	–	–	1	$x_4 \oplus x_1x_3 \oplus x_1 \oplus 1$
	1	1	–	
	1	–	–	
	–	–	–	
$h^4 = h^4(x_1, x_3, x_4) = f^3(x_1, 0, x_3, x_4)$	–	1	1	$x_3x_4 \oplus x_4 \oplus x_1x_3 \oplus x_1$
	–	–	1	
	1	1	–	
	1	–	–	

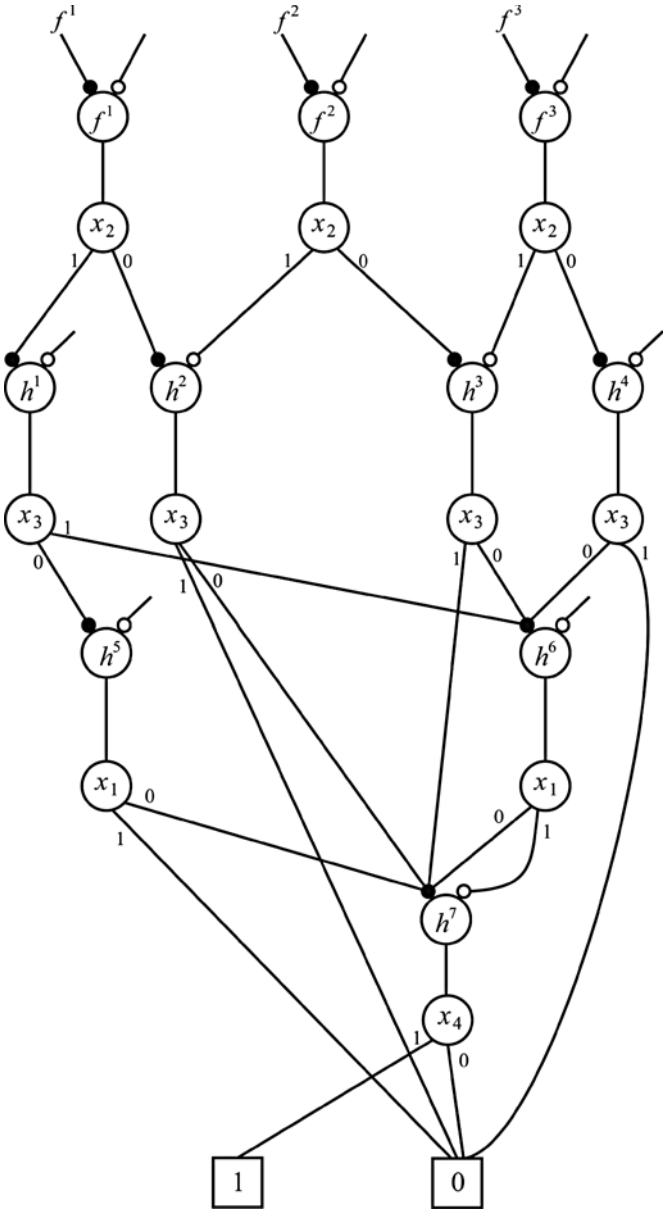

Рис. 3. Граф *BDDI*, представляющий систему функций

реализующий функциональную вершину h^7, выродился до инвертора.

Переход от *BDDI* к *BDD* состоит из двух этапов.

Первый этап заключается в "расщеплении" тех функциональных вершин, которые реализуют пару взаимно инверсных подфункций на две функциональные вершины, реализующие каждую подфункцию по отдельности. При этом для инверсной подфункции добавляется подграф *BDD*, пометки 0,1 на листовых вершинах которого заменяются инверсными (1,0), получаемая *OBDD* является нередуцированной (несокращенной). Например, для *BDDI* (см. рис. 3) результат выполнения первого этапа показан на рис. 5, полужирными линиями выделены добавляемые подграфы *BDD*.

Таблица 6

Подфункции разложения по переменной x_3

Подфункции разложения по переменной x_3	Полином Жегалкина		
	Матричное представление		Формула
	x_1	x_4	
$h^5 = h^1(x_1, 0, x_4)$	1 —	1 1	$x_1x_4 \oplus x_4$
$h^6 = h^1(x_1, 1, x_4) = h^3(x_1, 0, x_4) = h^4(x_1, 0, x_4)$	— 1	1 —	$x_4 \oplus x_1$
$h^7 = h^3(x_1, 1, x_4)$	—	1	x_4
$\bar{h}^7 = h^6(0, x_4)$	— —	1 —	$\bar{x}_4$

Второй этап — сокращение *OBDD* и получение *ROBDD* — является известной процедурой. В рассматриваемом примере результат сокращения *OBDD* показан на рис. 6.

8. Результаты экспериментов

Предложенный в данной статье алгоритм поиска лучшей перестановки для *BDDI* был программно реализован в виде программы *BDD_Builder*. Экспериментально исследованы были три ее варианта, в табл. 7 представлены результаты ее работы для трех испытанных эвристик. Первый вариант программы (столбец "ДНФ" в табл. 7) строил *BDD*-представление (без использования инверсий подфункций), оперируя ДНФ и используя эвристику 1 выбора очередной переменной разложения. Два других варианта *BDD_Builder* осуществляли переход от ДНФ к полиномам Жегалкина и использовали эвристики 1 и 2 соответственно. Этим вариантам соответствуют два последних столбца в табл. 7.

Программу *BDD_Builder* сравнивали с программой *Tie_BDD*, реализующей алгоритм [11] случайного перебора перестановок при минимизации *BDD*, и программой *ESPRESSO IIC* [8]. Все указанные программы выполняли предварительную логическую оптимизацию, при этом программа *ESPRESSO IIC* выполняла совместную минимизацию системы функций в классе ДНФ, а программа *Tie_BDD* минимизировала *BDD*-представление этой же системы функций. Затем по оптимизированным представлениям систем функций выполнялся синтез комбинационных логических схем и оценивались параметры полученных схем.

Сравнение проводилось на потоке из 42 примеров: 38 примеров систем ДНФ векторных функций $\mathbf{f}(\mathbf{x}) = \left(f^1(\mathbf{x}), \ldots, f^m(\mathbf{x})\right)$, $\mathbf{x} = (x_1, \ldots, x_n)$, из библиотеки *Berkeley PLA Test Set* [12] и четыре примера из практики проектирования. Пример X3_matr — это система ДНФ, полученная по многоуровневому представлению системы функций; Sin_16 — это задание

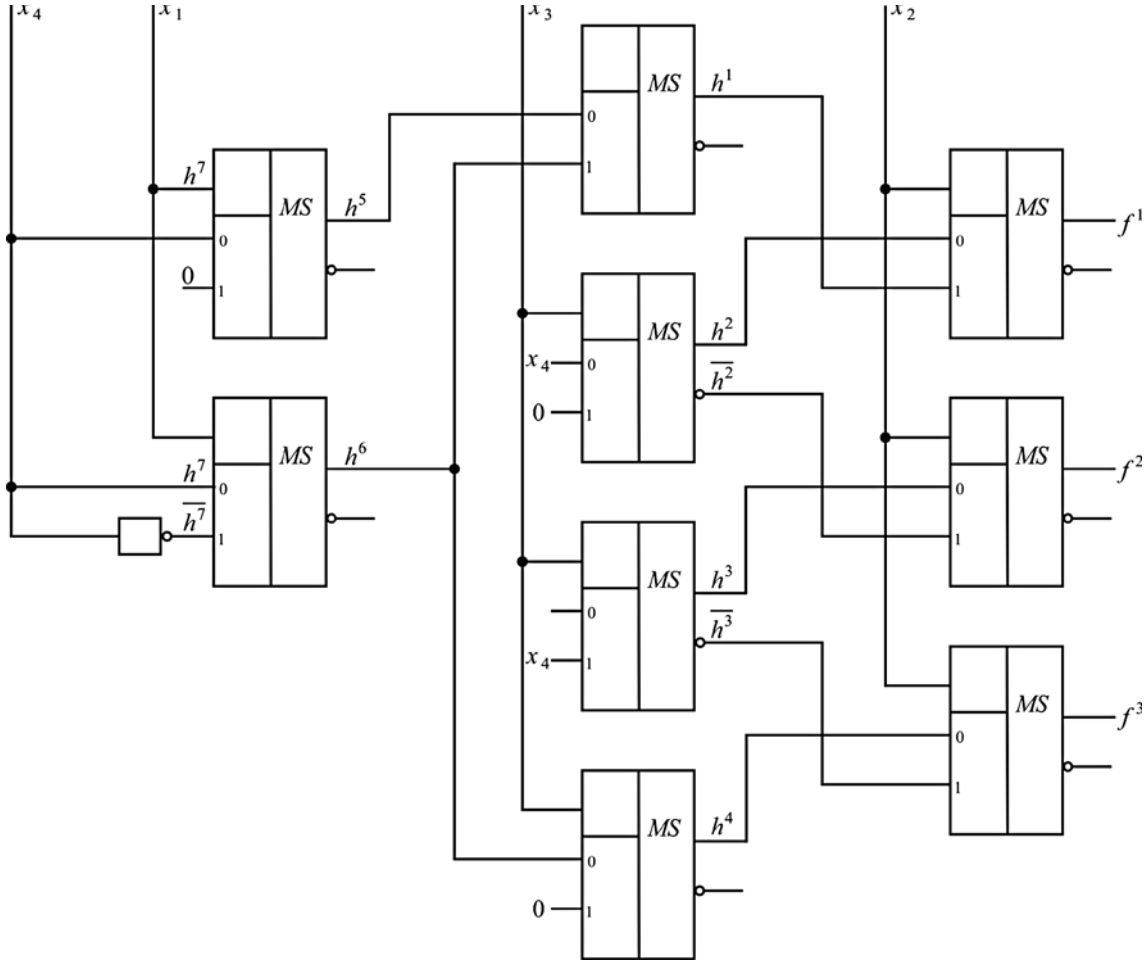

Рис. 4. Каскадная схема из мультиплексоров *MS*, соответствующая *BDDI* (рис. 3)

в виде таблицы истинности 16-битного приближения тригонометрической функции $y = \sin x$; примеры Syst4, Syst8 взяты из практики промышленного проектирования цифровых схем.

В табл. 8, 9 представлены результаты экспериментального сравнения трех программ — *ESPRESSO IIC*, *Tie_BDD*, *BDD_Builder*, лучшие решения выделены полужирным шрифтом. Программы *Tie_BDD*, *BDD_Builder* по матричной форме исходной системы ДНФ, представленной на языке SF в формате SDF [13], получают минимизированное многоуровневое представление системы функций в формате LOG логических выражений. Программа *ESPRESSO IIC* получала систему ДНФ с минимизированным общим числом элементарных конъюнкций, на которых заданы ДНФ всех функций системы. После получения таких представлений осуществлялся их перевод в VHDL-описания [14], использующие операторы назначения сигнала и логические операторы AND (И), OR (ИЛИ), NOT (НЕТ). Описание исходной системы ДНФ также переводилось в VHDL-описание для того, чтобы провести с помощью верификатора *Formal Pro* [15] формальную верификацию исходного VHDL-описания и полученного оптимизированного описания. Затем по каждому из VHDL-описаний выполнялось построение логической схемы в библиотеке КМОП-элементов заказной сверхбольшой интегральной схемы [16], выполнялся подсчет площадей и задержек схем. В качестве синтезатора логических схем из библиотечных элементов использовался *LeonardoSpectrum* [17], исходными данными для которого явились соответствующие VHDL-описания минимизированных логических выражений. Сложность схемы из библиотечных логических элементов на этапе логического проектирования выражается суммой S площадей элементов, составляющих схему. Топологические реализации всех логических КМОП-элементов имеют одинаковую высоту и различаются длиной. Каждый элемент библиотеки характеризуется своей площадью, необходимой для размещения его на кристалле. Пример логической схемы и соответствующие значения площадей логических элементов приведены в работе [16]. Площадь каждого элемента задается в условных единицах, как это требует синтезатор *LeonardoSpectrum*. Информацию о суммарной площади для размещения всех элементов схемы выдает *LeonardoSpectrum* после выполнения процедуры синтеза схемы.

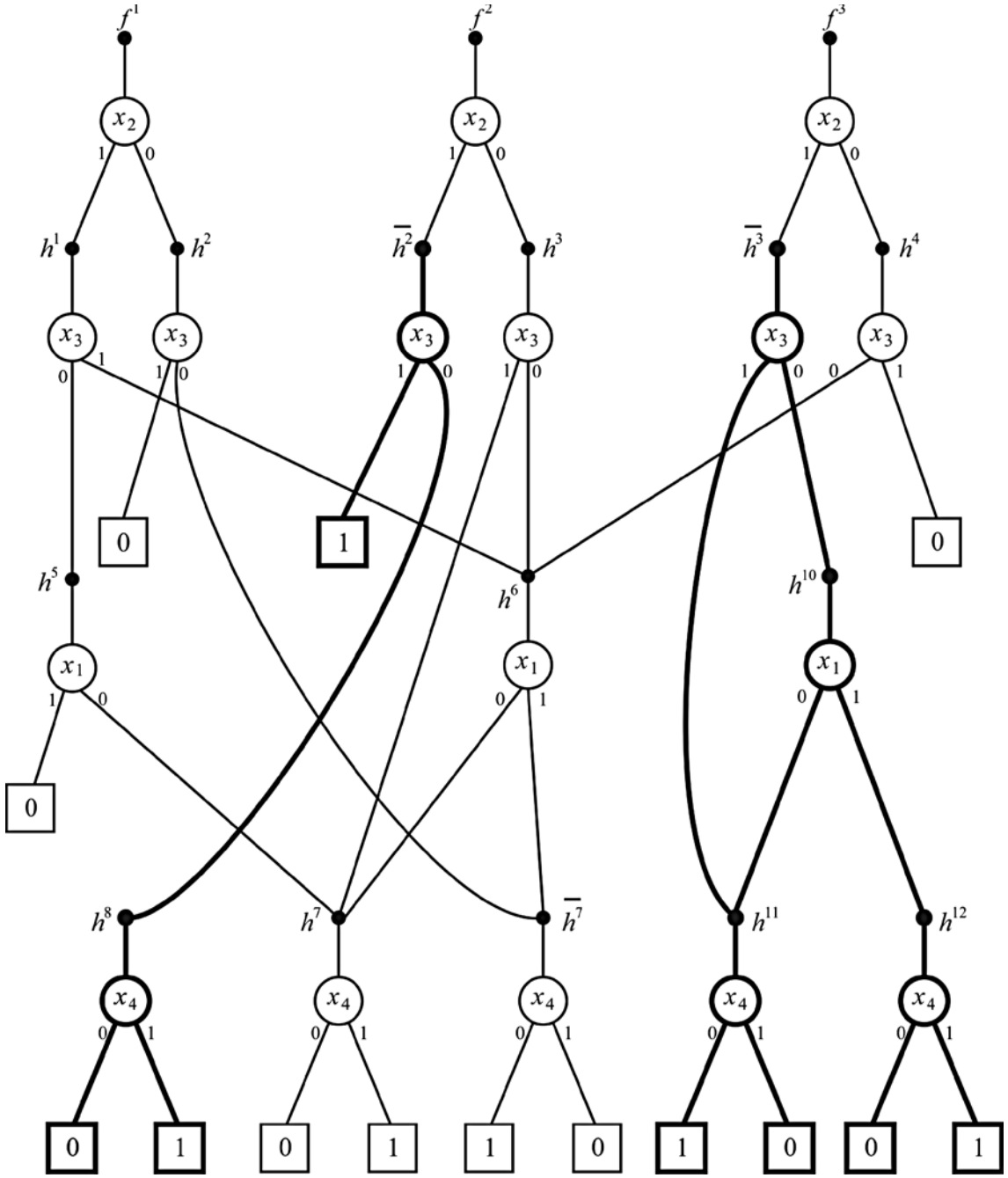

Рис. 5. Переход от *BDDI* к *BDD* (результат этапа 1)

В экспериментах под сложностью многоуровневого представления понимается число R уравнений (при этом уравнения для разложения по последней переменной x_i не записываются, т. е. из записи многоуровневого представления исключаются уравнения вида $h = x_i$, $h = \bar{x}_i$).

В табл. 7—9 используются следующие обозначения:

n — число переменных $x_1, x_2, \ldots, x_n$;

m — число функций;

k — число различных элементарных конъюнкций, входящих в ДНФ компонентных функций f^j, $j = 1, \ldots, m$;

P — число случайных перестановок, испробованных программой *Tie_BDD*;

R — число уравнений многоуровневого разложения Шеннона.

Синтез схемы Sin_16 по минимизированной (программой *ESPRESSO IIC)* системе ДНФ в системе *LeonardoSpectrum* не был выполнен в силу большой размерности логического VHDL-описания — минимизированная система ДНФ содержала 36 509 элементарных конъюнкций, а время работы программы *ESPRESSO IIC* составило более четырех часов рабо-

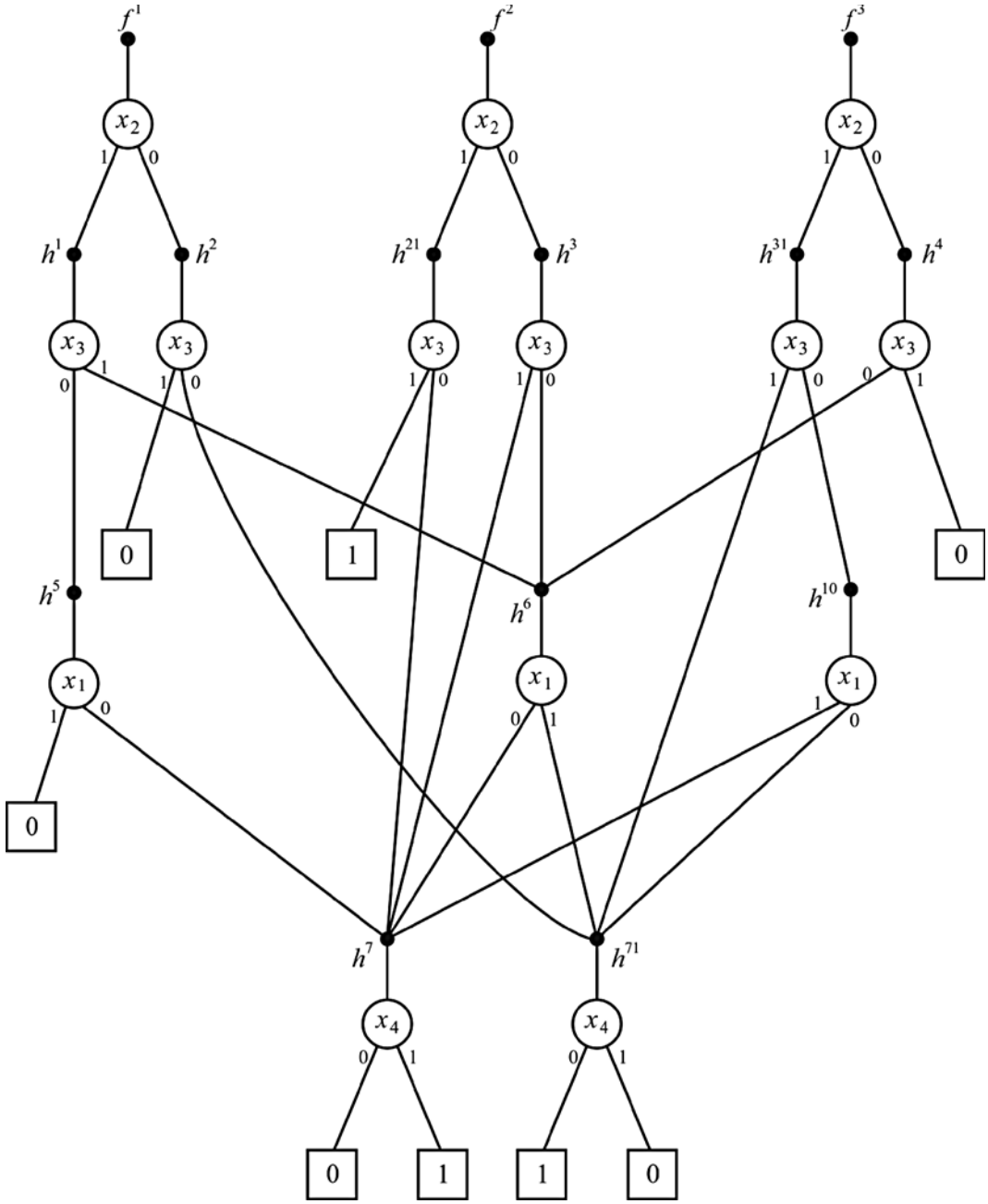

Рис. 6. Переход от *BDDI* к *BDD* (результат этапа 2)

ты персонального компьютера с тактовой частотой 3,5 ГГц. Заметим, что не был выполнен синтез схемы для примера Syst8. Общее время работы программы *BDD_Builder* (эвристика 1) на том же компьютере приведено в табл. 10, отдельно указано время, требуемое для построения полиномов Жегалкина. Примеры X9dn, Vtx1 характеризуются тем, что ДНФ в данных системах функций содержат большое число неортогональных пар конъюнкций — это приводит к длинным полиномам Жегалкина функций, которые были представлены такими ДНФ (см. последний столбец табл. 10).

Результаты экспериментов позволяют сделать следующие выводы.

- Оценка сложности системы функций, выражаемая в числе *R* уравнений (формул разложения Шеннона), является полезной при синтезе схем, так как синтез схемы от представления, в котором меньше уравнений, позволял, как правило, получать схемы меньшей площади. Здесь следует иметь в виду тот

Таблица 7

Результаты экспериментов — сравнение эвристик в программе *BDD_Builder* по числу *R* уравнений разложения Шеннона

Пример	*n*	*m*	*k*	Число уравнений *R*		
				ДНФ	Полиномы Жегалкина	
				Эвристика 1	Эвристика 1	Эвристика 2
ADD6	12	7	1092	54	**27**	32
ADDM4	9	8	512	189	**166**	192
ALU1	12	8	19	**16**	**16**	**16**
B12	15	9	431	69	**66**	68
B2	16	17	110	771	756	**569**
B9	16	5	123	85	73	**68**
BR1	12	8	34	116	119	**93**
BR2	12	8	35	87	**85**	101
DC2	8	7	58	61	**58**	58
DIST	8	5	256	144	**115**	123
IN0	15	11	138	332	**317**	322
IN2	19	10	137	289	**261**	274
INTB	15	7	664	820	681	**597**
LIFE	9	1	512	39	**36**	**36**
LOG8MOD	8	5	47	69	62	**61**
M1	6	12	32	56	**48**	**48**
M181	15	9	430	70	**67**	69
M2	8	16	96	135	**116**	**116**
M3	8	16	128	151	**130**	132
MLP4	8	8	256	164	147	**142**
MP2D	14	14	123	80	**78**	87
NEWTPLA	15	5	23	58	**56**	87
NEWTPLA1	10	2	4	**16**	**16**	20
NEWTPLA2	10	4	9	31	31	**29**
P82	5	14	24	56	**53**	56
RADD	8	5	120	33	**17**	22
RD53	5	3	32	21	**15**	**15**
RD73	7	3	147	41	**29**	**29**
ROOT	8	5	256	73	**56**	**56**
SEX	9	14	23	57	51	**47**
SYM10	10	1	837	36	**29**	**29**
T3	12	8	152	90	87	**65**
TIAL	14	8	640	718	**582**	592
Z5XP1	7	10	128	65	**40**	51
Z9SYM	9	1	420	31	**23**	**23**
Sin_16	16	16	65 536	17 759	**15 015**	15 480
Syst4	17	12	370	364	338	**293**
Syst8	25	28	45 548	15 119	**15 008**	16 386
X9dn	27	7	120	**103**	109	180
Vtx1	27	6	110	101	**100**	150
Soar	83	94	529	528	**470**	909
X3_matr	135	99	915	**856**	**856**	1408
Число лучших решений				4	31	19

Таблица 8

Результаты экспериментов — сравнение программ логической оптимизации по площади *S* схем

Пример	*n*	*m*	*k*	*P*	Площадь схемы *S*				
					ESPRESSO	*Tie_BDD*	*BDD_Builder*		
							ДНФ	Полиномы Жегалкина	
							Эвристика 1	Эвристика 1	Эвристика 2
ADD6	12	7	1092	5000	85 843	20 222	14 419	**12 806**	16 534
ADDM4	9	8	512	5000	124 663	89 269	86 959	**80 782**	92 009
ALU1	12	8	19	5000	**7109**	**7109**	**7109**	**7109**	**7109**
B12	15	9	431	5000	21 879	16 617	**16 137**	18 966	18 442
B2	16	17	110	5000	207 241	197967	183 638	199 106	**164 526**
B9	16	5	123	5000	30 227	**24 770**	26 778	26 081	32 247
BR1	12	8	34	5000	26 243	27 911	28 112	**23 843**	26 131
BR2	12	8	35	5000	21 472	21 634	**19 653**	21 371	20 283
DC2	8	7	58	5000	27 314	21 946	**21 505**	21 684	21 684
DIST	8	5	256	5000	87 980	68 601	69 560	**60 085**	62 808
IN0	15	11	138	5000	105 535	98 911	96 389	**91 116**	94 436
IN2	19	10	137	5000	128 368	92 410	81 513	76 117	**69 811**
INTB	15	7	664	5000	408 300	227 848	327 401	273 532	**246 764**
LIFE	9	1	512	5000	23 972	14 977	**14 391**	18 146	18 146
LOG8MOD	8	5	47	5000	27 906	25 953	26 611	**23 687**	25 350
M1	6	12	32	5000	22 398	18 715	16 695	**15 312**	**15 312**
M181	15	9	430	5000	20 981	17 225	**16 439**	19 513	16 779
M2	8	16	96	5000	62 334	61 095	47 563	44 227	**41 883**
M3	8	16	128	5000	83 901	60 325	58 813	52 580	**51 632**
MLP4	8	8	256	5000	88 895	71854	74 805	**68 439**	71 095
MP2D	14	14	123	5000	18 542	17 968	**17 438**	17 471	19 128
NEWTPLA	15	5	23	5000	15 044	14 229	13 571	**11 316**	12 053
NEWTPLA1	10	2	4	5000	4324	3577	**3421**	**3421**	3962
NEWTPLA2	10	4	9	5000	8588	7310	**6640**	**6640**	**6640**
P82	5	14	24	5000	22 008	**19 971**	20 512	19 988	20 763
RADD	8	5	120	5000	14 982	8465	10 117	**8074**	11 802
RD53	5	3	32	5000	12 142	9843	8270	**7321**	**7321**
RD73	7	3	147	5000	23 414	**15 925**	**15 925**	18 090	18 090
ROOT	8	5	256	5000	34 429	26 717	26 717	26 109	**26 075**
SEX	9	14	23	5000	14 692	13 928	12 538	**11 891**	12 890
SYM10	10	1	837	5000	69 906	19 882	19 882	**19 251**	**19 251**
T3	12	8	152	5000	18 866	19 223	**15 217**	16 534	17 153
TIAL	14	8	640	5000	336 022	295 952	294 830	**261 278**	278 665
Z5XP1	7	10	128	5000	43 172	26 499	25 919	**18 442**	22 582
Z9SYM	9	1	420	5000	42 994	18 018	15 909	**15 289**	**15 289**
Sin_16	16	16	65536	10	—	9 822 718	9 105 109	8 318 112	**8 254 215**
Syst4	17	12	370	500	130075	134121	115791	**108899**	112 722
Syst8	25	28	45548	100	—	13 031 482	8 183 862	**6 567 459**	7 848 331
X9dn	27	7	120	500	27 426	26 399	**24 876**	26 996	34 719
Vtx1	27	6	110	500	**26 215**	28 715	**26 354**	26 996	31 600
Soar	83	94	529	500	171 630	225 934	135 298	**121 627**	163 957
X3_matr	135	99	915	100	515 402	338 042	**242 406**	**242 406**	211 633
Число лучших решений					2	4	13	**23**	13

Таблица 9

Результаты экспериментов — сравнение программ логической оптимизации по задержке схем

Пример	*n*	*m*	*k*	Задержка схемы, нс				
				ESPRESSO	*Tie_BDD*	*BDD_Builder*		
						ДНФ	Полиномы Жегалкина	
						Эвристика 1	Эвристика 1	Эвристика 2
ADD6	12	7	1092	7,46	**5,50**	5,97	8,03	6,82
ADDM4	9	8	512	6,49	7,00	**6,25**	7,84	6,77
ALU1	12	8	19	**1,11**	**1,11**	1,12	1,12	1,12
B12	15	9	431	4,24	3,37	**2,78**	3,62	3,13
B2	16	17	110	11,05	10,24	**7,84**	8,18	7,92
B9	16	5	123	4,59	**3,64**	4,97	4,91	5,32
BR1	12	8	34	5,47	6,08	5,71	6,36	**4,51**
BR2	12	8	35	6,11	6,43	6,55	6,34	**4,85**
DC2	8	7	58	4,09	4,11	3,99	**3,77**	**3,77**
DIST	8	5	256	5,92	5,51	**5,09**	6,08	6,41
IN0	15	11	138	7,60	7,52	6,70	**6,29**	6,44
IN2	19	10	137	8,10	7,49	6,47	7,36	**6,32**
INTB	15	7	664	**9,02**	9,77	9,21	**9,02**	10,35
LIFE	9	1	512	4,90	4,99	**4,54**	4,60	4,60
LOG8MOD	8	5	47	4,55	4,38	3,89	**3,38**	3,78
M1	6	12	32	4,59	3,53	3,66	**3,46**	**3,46**
M181	15	9	430	2,96	3,43	**2,78**	4,07	3,18
M2	8	16	96	6,28	5,53	**4,25**	4,77	4,28
M3	8	16	128	7,69	5,18	5,20	**4,49**	5,13
MLP4	8	8	256	6,17	5,67	**4,80**	5,60	6,09
MP2D	14	14	123	4,46	4,68	**3,56**	**3,56**	4,86
NEWTPLA	15	5	23	3,52	4,60	3,57	3,30	**3,21**
NEWTPLA1	10	2	4	2,60	3,16	**1,89**	**1,89**	2,08
NEWTPLA2	10	4	9	4,94	3,61	**2,79**	**2,79**	**2,79**
P82	5	14	24	3,46	**2,85**	3,57	2,93	3,18
RADD	8	5	120	4,04	4,44	**3,34**	4,90	3,69
RD53	5	3	32	4,05	2,85	**2,37**	3,38	3,38
RD73	7	3	147	4,70	**3,94**	**3,94**	4,55	4,55
ROOT	8	5	256	**4,55**	4,81	4,81	4,78	4,78
SEX	9	14	23	3,12	2,85	2,95	3,63	**2,81**
SYM10	10	1	837	7,85	**5,71**	**5,71**	5,77	5,77
T3	12	8	152	4,20	4,31	**3,93**	5,20	4,31
TIAL	14	8	640	8,67	9,56	**7,71**	8,02	8,62
Z5XP1	7	10	128	5,18	3,85	**3,64**	4,64	5,24
Z9SYM	9	1	420	6,07	5,39	**4,77**	5,03	5,03
Sin_16	16	16	65536	—	**15,86**	17,27	16,40	16,82
Syst4	17	12	370	7,66	8,56	**7,09**	7,78	7,21
Syst8	25	28	45548	—	19,72	19,39	17,95	17,21
X9dn	27	7	120	5,65	5,58	**5,55**	6,00	6,58
Vtx1	27	6	110	5,70	5,28	6,58	5,96	**4,23**
Soar	83	94	529	**5,65**	6,31	5,67	6,60	6,14
X3_matr	135	99	915	10,95	10,69	7,91	7,91	**5,73**
Число лучших решений				4	7	21	9	11

Таблица 10

Результаты экспериментов — время работы программы *BDD_Builder* на промышленных примерах

Пример	Эвристика 1		Суммарная длина полиномов Жегалкина
	Время построения полиномов Жегалкина, с	Общее время построения разложений Шеннона, с	
Sin_16	133,59	206,01	326 420
Syst4	0,04	0,25	1787
Syst8	27,79	127,95	411 946
X9dn	12,82	202,14	1 948 323
Vtx1	9,91	136,76	1 309 308
Soar	4,58	80,31	151 457
X3_matr	37,08	104,52	32 759

факт, что синтезатор *LeonardoSpectrum* выполняет при синтезе собственные встроенные процедуры технологически независимой оптимизации.

- Предложенный локально оптимальный алгоритм выбора перестановки переменных при оптимизации многоуровневых разложений Шеннона с учетом инверсий подфункций позволяет получать меньшее число уравнений по сравнению с *BDD*-минимизацией, при которой инверсии подфункций не ищутся. Это приводит к логическим схемам меньшей площади и большего быстродействия. Для систем ДНФ большой размерности предложенный алгоритм является более предпочтительным по сравнению с алгоритмом [12] перебора случайных перестановок.
- Использование полиномов Жегалкина и эвристики 1 позволяют быстро находить инверсии подфункций и улучшать результаты логической оптимизации. Однако не для всех систем ДНФ оправдан переход к полиномам Жегалкина — переход бывает неэффективен (полиномы имеют большую длину), когда ДНФ содержит большое число попарно неортогональных элементарных конъюнкций.
- Эвристика 1 выбора очередной переменной разложения является более предпочтительной, если имеется цель получения схем меньшей площади. Если целью является получение комбинационных схем (в заданной библиотеке КМОП-элементов) с меньшей задержкой, то более предпочтительным является использование *BDD*-представлений, получаемых первым вариантом программы *BDD_Builder*.
- Предложенные алгоритмы и разработанная на их основе программа *BDD_Builder* высокоэффективны для ортогонализованных систем ДНФ и близких к ним, т. е. содержащим ограниченное число неортогональных пар элементарных конъюнкций, входящих в ДНФ функций.

В целом, используя три варианта программы *BDD_Builder*, можно получать значительно лучшие результаты технологически независимой оптимизации по сравнению с программами *ESPRESSO IIC* и *Tie_BDD*. Программа *BDD_Builder* позволяет быстро выполнять предварительную технологически независимую оптимизацию, получать минимизированные многоуровневые представления, по которым быстро выполняется синтез схем для примеров большой размерности даже в тех случаях, когда имеющиеся инструменты синтеза схем не могут решить задачу, либо для ее решения требуется много времени. Данная программа включена в систему *FLC* [13] и используется на этапе технологически независимой оптимизации при синтезе логических схем из библиотечных КМОП-элементов. Маршруты оптимизации, использующие продукционно-фреймовую модель представления знаний в области проектирования логических схем, позволяют определить целесообразность использования как разработанной программы *BDD_Builder*, так и уже имеющейся в составе *FLC* программы *Tie_BDD*.

Заключение

Возрастание размерностей задач логического синтеза приводит к необходимости совершенствования методов, алгоритмов и программ технологически независимой оптимизации, выполняемой перед реализацией оптимизированных представлений систем булевых функций в требуемом базисе логических элементов. Предложенный в данной работе и программно реализованный, экспериментально исследованный алгоритм позволяет быстро и эффективно выполнять оптимизацию многоуровневых представлений систем булевых функций с сотнями переменных и функций. Реализующая алгоритм программа может служить эффективным инструментом в системах автоматизированного проектирования для минимизации площади и увеличения быстродействия функциональных блоков заказных циф-

ровых сверхбольших интегральных схем, особенно в тех случаях, когда схемная реализация неоптимизированных описаний затруднена либо вообще невозможна в существующих в настоящее время промышленных синтезаторах логических схем для заказных СБИС.

Список литературы

1. **Бибило П. Н.** Применение диаграмм двоичного выбора при синтезе логических схем. Минск: Беларус. навука, 2014. 231 с.
2. **Akers S. B.** Binary Decision Diagrams // IEEE Trans. on Computers. 1978. Vol. C-27, N 6. P. 509—516.
3. **Bryant R. E.** Graph-based algorithms for Boolean function manipulation // IEEE Transactions on Computers. 1986. Vol. 35, N 8. P. 677—691.
4. **Bryant R. E.** Ordered Binary Decision Diagrams // Logic synthesis and verification / ed. by S. Hassoun, T. Sasao, R. K. Brayton. Kluwer Academic Publishers, 2002. P. 285—307.
5. **Meinel C.** Algorithms and Data Structures in VLSI Design: OBDD Foundations and Applications. Berlin; Heidelberg: Springer-Verlag, 1998. 267 p.
6. **Кнут Д. Э.** Искусство программирования. М.: Вильямс, 2013. Т. 4, А: Комбинаторные алгоритмы, Ч. 1. 960 с.
7. **Авдеев Н. А., Бибило П. Н.** Эффективность логической оптимизации при синтезе комбинационных схем из библиотечных элементов // Микроэлектроника. 2015. Т. 44, № 5. С. 383—399.
8. **Brayton K. R., Hactel G. D., McMullen C. T., Sangiovanni-Vincentelli A. L.** Logic minimization algorithm for VLSI synthesis. Boston, e.a.: Kluwer Academic Publishers, 1984. 193 p.
9. **Закревский А. Д., Поттосин Ю. В., Черемисинова Л. Д.** Логические основы проектирования дискретных устройств. М.: Физматлит, 2007. 589 с.
10. **Ebendt R., Fey G., Drechsler R.** Advanced BDD Optimization. Dordrecht, Springer, 2005. 222 p.
11. **Бибило П. Н., Леончик П. В.** Алгоритм построения диаграммы двоичного выбора для системы полностью определенных булевых функций // Управляющие системы и машины. 2009. № 6. С. 42—49.
12. http://www1.cs.columbia.edu/~cs6861/sis/espresso-examples/ex.
13. **Бибило П. Н., Романов В. И.** Логическое проектирование дискретных устройств с использованием продукционно-фреймовой модели представления знаний. Минск: Беларус. навука, 2011. 279 с.
14. **Ashenden P. J., Lewis J.** VHDL-2008. Just the New Stuff, Burlington, MA, USA, Morgan Kaufman Publishers, 2007. 244 p.
15. **Лохов А.** Функциональная верификация СБИС в свете решений Mentor Graphics // ЭЛЕКТРОНИКА: Наука, Технология, Бизнес. 2004. № 1. С. 58—62.
16. **Авдеев Н. А., Бибило П. Н.** Эффективность логической оптимизации при синтезе комбинационных схем из библиотечных элементов // Микроэлектроника. 2015. Т. 44, № 5. С. 383—399.
17. **Бибило П. Н.** Системы проектирования интегральных схем на основе языка VHDL. StateCAD, ModelSim, LeonardoSpectrum. М.:СОЛОН-Пресс, 2005. 384 с.

The Use of Zhegalkin Polynomials for Minimization of Multilevel Representations of Boolean Functions Based on Shannon Expansion

P. N. Bibilo, bibilo@newman.bas-net.by, **Yu. Yu. Lankevich,** yurafreedom18@gmail.com,
The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, 220012, Belarus

Corresponding author:

Bibilo Petr N., Head of Laboratory, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, 220012, Belarus,
E-mail: bibilo@newman.bas-net.by

Received on April 26, 2017
Accepted on May 15, 2017

The synthesis of combinational logic is an actual task of automatic design of digital circuits. It is usually divided into two stages: technologically independent optimization and technological mapping to a given basis of logical elements. The first stage is the most significant one because the minimization of different forms of Boolean function representation is fulfilled in this stage. If the functions of a system are given in DNF (Disjunctive Normal Form) then change to multilevel representations based on Shannon expansion is expedient when synthesizing circuits from logic library elements. The graphical form of this representation is called BDD (Binary Decision Diagram). As a rule, the synthesis of circuits using logical equations that correspond to minimized BDD representation gives better results in comparison with the synthesis using minimized DNF.

A minimization of multilevel representation of Boolean function systems based on Shannon expansion with finding equal coefficients (accurate within inversion) and using Zhegalkin polynomials for these purposes is proposed. Zhegalkin polynomials are easily comparable and can be used to get function inversion. It reduces considerably the computation time. Application of a program that implements the proposed algorithms allows obtaining smaller areas

of VLSI schemes in comparison with the circuits that are synthesized using minimized circuits of DNF and Shannon expansion where the coefficient inversion is not considered.

Keywords: *synthesis of logical circuits, minimization of Boolean functions, Shannon expansion, Zhegalkin polynomial*

For citation:

Bibilo P. N., **Lankevich Yu. Yu.** The Use of Zhegalkin Polynomials for Minimization of Multilevel Representations of Boolean Functions Based on Shannon Expansion, *Programmnaya Ingeneria,* 2017, vol. 8, no. 8, pp. 369—384.

DOI: 10.17587/prin.8.369-384

References

1. **Bibilo P. N.** *Primenenie diagramm dvoichnogo vybora pri sinteze logicheskih shem* (Application of Binary Decision Diagrams at synthesis of logical circuits), Minsk, Belarus. navuka, 2014, 231 p. (in Russian).

2. **Akers S. B.** Binary Decision Diagrams, *IEEE Trans. on Computers,* 1978, vol. C-27, no. 6, pp. 509—516.

3. **Bryant R. E.** Graph-based algorithms for Boolean function manipulation, *Transactions on Computers,* 1986, vol. 35, no. 8, pp. 677—691.

4. **Bryant R. E.** Ordered Binary Decision Diagrams, *Logic synthesis and verification* /ed. by S. Hassoun, T. Sasao, R. K. Brayton, Kluwer Academic Publishers, 2002, pp. 285—307.

5. **Meinel C.** *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*, Berlin, Heidelberg, Springer-Verlag, 1998, 267 p.

6. **Knut D. Je.** *Iskusstvo programmirovanija*, vol. 4, *A: Kombinatornye algoritmy, Ch. 1.* (The Art of Computer Programming, Combinatorial Algorithms, Part 1) Moscow, Willams, 2013, 960 p. (in Russian).

7. **Avdeev N. A., Bibilo P. N.** Jeffektivnost' logicheskoj optimizacii pri sinteze kombinacionnyh shem iz bibliotechnyh jelementov (Logical Optimization Efficiency in the Synthesis of Combinational Circuits), *Mikrojelektronika,* 2015, vol. 44, no. 5, pp. 383—399 (in Russian).

8. **Brayton K. R., Hactel G. D., McMullen C. T., Sangiovanni-Vincentelli A. L.** *Logic minimization algorithm for VLSI synthesis,* Boston, e.a., Kluwer Academic Publishers, 1984, 193 p.

9. **Zakrevskij A. D., Pottosin Ju. V., Cheremisinova L. D.** *Logicheskie osnovy proektirovanija diskretnyh ustrojstv* (Logical bases of design of discrete devices), Moscow, Fizmatlit, 2007, 589 p. (in Russian).

10. **Ebendt R., Fey G., Drechsler R.** *Advanced BDD Optimization,* Dordrecht, Springer, 2005, 222 p.

11. **Bibilo P. N., Leonchik P. V.** Algoritm postroenija diagrammy dvoichnogo vybora dlja sistemy polnost'ju opredelennyh bulevyh funkcij (Algorithm of creation of the Binary Decision Diagram for system of completely specified Boolean functions), *Upravljajushhie sistemy i mashiny*, 2009, no 6, pp. 42—49 (in Russian).

12. http://www1.cs.columbia.edu/~cs6861/sis/espresso-examples/ex.

13. **Bibilo P. N., Romanov V. I.** *Logicheskoe proektirovanie diskretnyh ustrojstv s ispol'zovaniem produkcionno-frejmovoj modeli predstavlenija znanij* (Logical design of discrete devices with use of productional and frame model of representation of knowledge), Minsk, Belarus. navuka, 2011, 279 p. (in Russian).

14. **Ashenden P. J., Lewis J.** *VHDL-2008. Just the New Stuff*, Burlington, MA, USA, Morgan Kaufman Publishers, 2007, 244 p.

15. **Lohov A.** Funkcional'naja verifikacija SBIS v svete reshenij Mentor Graphics (Functional verification of VLSI in the light of decisions of Mentor Graphics), *JeLEKTRONIKA: Nauka, Tehnologija, Biznes*, 2004, no. 1, pp. 58—62 (in Russian).

16. **Avdeev N. A., Bibilo P. N.,** Effektivnost' logicheskoj optimizacii pri sinteze kombinacionnyh shem iz bibliotechnyh elementov (The efficiency of a logic optimization during the synthesis of combinational circuits from the library of elements), *Mikrojelektronika,* 2015, vol. 44, no. 5, pp. 383—399 (in Russian).

17. **Bibilo P. N.** *Sistemy proektirovanija integral'nyh shem na osnove jazyka VHDL. StateCAD, ModelSim, LeonardoSpectrum* (CAD of integrated circuits based on the VHDL. StateCAD, ModelSim, LeonardoSpectrum), Moscow, SOLON-Press, 2005, 384 p. (in Russian).

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Е. В. Комиссарова*

Сдано в набор 05.06.2017 г. Подписано в печать 19.07.2017 г. Формат 60×88 1/8. Заказ PI817
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**