**Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Воронежский государственный технический университет»**

На правах рукописи

ДОРЕНСКАЯ Елизавета Александровна

МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДИНАМИЧЕСКОГО ПРОЕКТИРОВАНИЯ ТРАНСЛЯТОРА СО СПЕЦИАЛЬНОГО МЕТАЯЗЫКА ОПИСАНИЯ ЗАДАЧИ В ЯЗЫК ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ

Специальность 2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
доктор технических наук
Кравец Олег Яковлевич

Воронеж — 2023

ОГЛАВЛЕНИЕ

Введение

Актуальность темы. Благодаря стремительному развитию информационных технологий возникла необходимость в снижении трудоёмкости написания программ и предотвращении появления ошибок. Большой вклад в развитие автоматизации и сокращение числа ошибок в программах внесли Holzmann G.J., Matej Balog, Alexander L. Gaunt, Аветисян А. И., Иванников В. П., Белеванцев А. А, Raghothaman, M. В теоретическом плане ряд методов автоматизации и предотвращения появления ошибок сводится к применению библиотек и специальных правил при написании программного кода. Разумной идеей кажется введение процедуры описания задачи с последующим преобразованием его в исполняемый код на алгоритмическом языке программирования высокого уровня.

Несмотря на то, что последние годы в области развития искусственного интеллекта случился сильный прогресс, он создан в основном с помощью нейронных сетей, которые разработаны по образу и подобию человеческого мышления. А это значит, скорее всего, будут совершать те же ошибки, что и человек. Ведущие исследователи в данном направлении: Andrew Ng, Yann Andre LeCun, Fei-Fei Li. Задача разработки кода с помощью нейронных сетей решается исследователями из Microsoft и Кембриджского университета. Чтобы устранить этот недостаток в области нейросетей, необходимо как можно большую часть работы по написанию кода перекладывать на ЭВМ. Это необходимо для предотвращения появления ошибок, вызванного наличием человеческого фактора, а также сокращения трудоёмкости разработки программного кода.

Реализация крупных проектов в области IT-технологий требует привлечения значительного количества квалифицированных специалистов. Сжатые сроки выполнения проектов приводят к необходимости сокращения времени на отладку программного обеспечения. Как правило, этого не удаётся добиться даже при увеличении числа программистов. Поэтому целесообразно

провести оптимизацию процессов программирования за счёт разработки специализированной оболочки, генерирующей программные коды. В этом случае снимается противоречие, возникающее между разработчиком технического задания и программистом, создающим приложение.

Таким образом, актуальность темы диссертационного исследования продиктована необходимостью разработки математического и программного обеспечения автоматизации процесса программирования за счет введения процедуры описания задачи с последующим преобразованием его в исполняемый код на алгоритмическом языке программирования высокого уровня.

Тематика диссертационной работы соответствует научному направлению ФГБОУ ВО «Воронежский государственный технический университет» «Вычислительные комплексы и проблемно-ориентированные системы управления».

Целью работы является разработка математического и программного обеспечения динамически проектируемого транслятора со специального метаязыка описания задачи на язык высокого уровня для снижения трудоемкости генерации кода.

Задачи исследования. Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать синтаксис метаязыка описания задачи для генерации его с помощью транслятора в язык программирования высокого уровня.
2. Разработать алгоритм динамического выбора состава модулей для автоматического выбора способа решения задачи в рамках коллектива задач одного типа и сокращения количества модулей в базе данных.
3. Разработать алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных для получения корректного результата в большинстве случаев без больших затрат ресурсов и сложных вычислений.

4. Разработать структуру динамического транслятора описания задачи в язык Perl, обеспечивающую автоматическое проектирование программы решения задачи на языке высокого уровня.

Объект исследования: способы автоматизации разработки программного обеспечения.

Предмет исследования: методы оценки снижения числа программных ошибок в коде, методы сокращения трудоёмкости написания программного кода.

Методы исследования: системный анализ, математическая статистика, использование специальных метрик и подходы, применяемые при разработке системного программного обеспечения.

Тематика работы соответствует следующим пунктам паспорта специальности 2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей: п.1 «Модели, методы и алгоритмы проектирования, анализа, трансформации, верификации и тестирования программ и программных систем»; п.2 «Языки программирования и системы программирования, семантика программ»; п.3 «Модели, методы, архитектуры, алгоритмы, языки и программные инструменты организации взаимодействия программ и программных систем».

Научная новизна работы. В диссертации получены следующие результаты, характеризующиеся научной новизной:

1. Синтаксис метаязыка описания задачи, отличающийся ориентацией на создание специальных описаний для генерации программного кода динамически конструируемым транслятором, обеспечивающий снижение трудоёмкости генерации кода на языке высокого уровня.

2. Алгоритм динамического выбора состава модулей для решения коллектива задач одного типа, отличающихся автоматическим учетом специфики задач и обеспечивающий рациональный состав модулей.

3. Алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных, отличающийся учетом расстояний между текстовыми элементами результатов верификации и их весов в специальной базе данных для подбора корректной семантики, обеспечивающий корректный результат в более чем 95% случаев.

4. Структура динамического транслятора описания задачи в язык Perl, отличающаяся автоматическим учетом контекста задачи и обеспечивающая проектирование программы решения задачи на языке высокого уровня.

Теоретическая и практическая значимость исследования заключается в разработке математического и программного обеспечения динамически конструируемого транслятора со специального метаязыка описания задачи на язык программирования высокого уровня с целью снижения трудоемкости генерации кода и сокращения числа ошибок, а также алгоритма идентификации формализованных результатов внешней верификации программных модулей.

Теоретические результаты работы могут быть использованы в проектных и научно-исследовательских организациях, для снижения трудоёмкости создания программного кода, благодаря чему его могут создавать высококвалифицированные специалисты в какой-либо области наук не владеющие языками программирования. Алгоритм идентификации формализованных результатов внешней верификации программных модулей может использоваться для распознавания смысла специализированных текстовых описаний с помощью ЭВМ.

Результаты внедрения. Основные результаты внедрены в группе технической поддержки компьютерных линий связи Национального исследовательского центра "Курчатовский институт" Института теоретической и экспериментальной физики имени А. И. Алиханова (ИТЭФ), для поддержки безопасности веб-серверов.

Апробация работы. Основные положения диссертационной работы докладывались и обсуждались на следующих конференциях: 60-й всероссийской научной конференция МФТИ (Москва, 2017 г.); 8-й Международной конференции «Распределенные вычисления и GRID-технологии в науке и образовании». (Дубна, 2018); Молодежной конференции по теоретической и экспериментальной физике (Москва, 2018-2021 гг.); XV и XVII Международной научно-практической конференции «Современные информационные технологии и ИТ-образование» (Москва, 2020 и 2022гг.); Всероссийской научно-практической конференции "Решение" (г. Березники, 2022 г.), а также на научных семинарах ОИЯИ, МИФИ, НИЦ «Курчатовский институт», ИСП РАН (2021-2022 гг.)

Достоверность результатов обусловлена корректным использованием теоретических методов исследования и подтверждена результатами сравнительного анализа данных вычислительных и натурных экспериментов.

Публикации. По результатам диссертационного исследования опубликовано 18 научных работ (12 – без соавторов), в том числе 18 – в изданиях, рекомендованных ВАК РФ (из них 1 – в издании Scopus, 1 – патент на изобретение и 11 свидетельств о регистрации программы для ЭВМ). В работах, опубликованных в соавторстве и приведенных в конце автореферата, лично автором получены следующие результаты: [1,6] – алгоритм динамического выбора состава модулей для решения коллектива задач одного типа; [2,3,7] – алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных; [4,5] - синтаксис языка описания задачи (CDTL), структура транслятора описания задачи на языке CDTL в PERL и базы данных алгоритмов;

Объем и структура работы. Диссертация состоит из введения, 4 глав, заключения, списка литературы и приложения. Текст диссертации изложен на 148 страницах. Иллюстративный материал включает 33 рисунка и 8 таблиц. Список литературы содержит 134 наименования.

Содержание работы.

Во введении обоснована актуальность исследования, сформулированы его цель и задачи, научная новизна и практическая значимость полученных результатов, приведены сведения об апробации и внедрении работы.

В **первой главе** исследуются особенности обнаружения ошибок в программном обеспечении и методов контекстного анализа, способы автоматизации программирования. Описаны последствия возникновения программных ошибок. Отмечено, что рассчитать сокращение количества ошибок в программах и трудоёмкость их написания можно с помощью специальных метрик. Дан небольшой обзор таких метрик.

По итогу анализа метрик для оценки сокращения количества ошибок были применены: простая интуитивная модель и метрика Холстеда. Простая интуитивная модель была выбрана потому что имелись ресурсы, необходимые для реализации расчётов с её использованием, и можно было получить нужную оценку по количеству ошибок. Помимо этого, применить её было проще чем другие, упомянутые выше, модели. Метрика Холстеда была выбрана потому что она подходит для решаемой задачи и с помощью неё довольно просто сделать нужные расчёты. Результат данного анализа потребовал формализации данных задач, а также алгоритмизации их решения с учетом особенностей. Сформулирована цель и задачи исследования.

Вторая глава посвящена разработке специального метаязыка описания задачи и алгоритма динамического выбора состава модулей для решения коллектива задач одного типа.

Для формирования описания задачи был разработан синтаксис специального метаязыка для сокращения числа ошибок и снижения трудоёмкости написания программного кода, который получил название Creating Description Task Language (CDTL).

Для того, чтобы оценить насколько уменьшается описание, написанное человеком на языке CDTL, по сравнению с кодом на языке Perl был вычислен

коэффициент сжатия. Он был рассчитан для более чем 10 описаний на языке CDTL и аналогичных программ на Perl. Данный коэффициент показывает во сколько раз сокращается текст при использовании CDTL. Минимальный коэффициент сжатия при использовании CDTL в данной выборке равен 2,1. Максимальный коэффициент = 12. Для данных значений была рассчитана дисперсия, которая оказалась равна 9,8.

Разработан алгоритм динамического выбора состава модулей для решения коллектива задач одного типа. Данный алгоритм использует вычисление определенных функций и проверку результата этого расчета. По результатам проверки принимается решение о выполнении того или иного фрагмента кода из числа имеющихся. Алгоритм модифицируется так, чтобы отвечать требованиям списков входных и выходных параметров.

Создан «Универсальный конфигуратор ПО для пакета программ защиты Web-сервера», который предназначен для операционной системы Scientific Linux версии 6. Главной его функцией является настройка операционной системы для использования пакета программ защиты Web-сервера. С помощью него можно проверить физическую конфигурацию компьютера, наличие необходимого ПО и библиотек. Особенность конфигуратора заключается в том, что он устанавливает необходимое ПО автоматически.

Третья глава посвящена алгоритму идентификации формализованных результатов внешней верификации программных модулей в базах данных и фазам работы транслятора из CDTL в код языка Perl.

Разработан банк алгоритмов, в котором находятся модули. Меняя входные данные модулей банка алгоритмов, можно варьировать их работу. Это увеличивает область их применения и уменьшает необходимое количество скриптов. В отличие от депозитария Cpan, код модулей данного банка алгоритмов полностью доступен для редактирования, т. к. они написаны на интерпретируемом языке программирования Perl. И в отличие от библиотек Cpan их не надо специально устанавливать. Для хранения информации о

программных модулях, находящихся в банке алгоритмов, была использована таблица базы данных MySQL.

Разработан алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных. Он работает таким образом, что модули, находящиеся в банке алгоритмов, добавляются в открытый доступ. В результате чего люди используют эти программы, пишут отзывы. Для проверки данных отзывов была создана специальная программа, зарегистрированная в ФИПС[18]. Она создавалась с использованием метода определения контекста слова и текстового файла, на который был получен патент на изобретение[7].

Разработан специальный транслятор из CDTL в Perl. Работа этого транслятора подразделяется на 2 фазы: фазу анализа (parser) и фазу синтеза. Каждая из этих фаз имеет свои особенности. Особенность фазы анализа в том, что на ней определяется значение области проблематики. Оно может быть задано в описании задачи явно и определяет, какой банк алгоритмов и какую библиотеку примитивов следует использовать.

Работа на фазе синтеза в данном трансляторе существенно отличается от аналогичной фазы других трансляторов. На ней последовательно рассматриваются описания задач, и с использованием имеющихся таблиц для каждого описания формируется текст программы на языке Perl.Учитывая модульность метаязыка описания задач, транслятор должен быть также модульным.

Фаза синтеза может работать 3-мя способами:

1. Если в описании присутствует хотя бы один оператор действия, то проводится поиск модулей в банке алгоритмов (хранилище описаний алгоритмов).

2. Если модуль в банке алгоритмов не найден, то в описании задачи должны присутствовать имена примитивов. Тогда код программы генерируется из примитивов.

3. Если при тех же условиях, что в пункте 2, в описании задачи нет имен примитивов, то модуль придётся частично или полностью писать самому программисту.

В четвёртой главе представлены результаты апробации языка описания задачи CDTL в рамках создания транслятора в код языка Perl. Представлена экспериментальная оценка усовершенствования алгоритма внешней верификации программных модулей в базах данных. Даны оценки сокращения сложности написания программ и количества ошибок в них при практическом применении языка CDTL.

Описан пакет программ защиты Web-сервера, сгенерированный транслятором, который мониторирует входящий и исходящий информационный трафик, детектирует атаки переполнения буфера, попытки DoS-атак и path traversal, а также случаи использования нелегальных HTTP-методов (put, delete и пр.) и некоторых других атак. Дана подробная характеристика каждого из модулей, входящих в данных пакет.

Проведена экспериментальная оценка усовершенствованного алгоритма внешней верификации программных модулей в базах данных. Данная техника может найти применение при анализе смысла текстов, и для снижения числа программных ошибок методом описания задачи, а не алгоритма [1,20].

Проведена оценка сокращения количества ошибок в программах при применении метаязыка CDTL с помощью метрики Холстеда и простой интуитивной модели. По метрике Холстеда при применении CDTL количество ошибок сокращается на 71,8%, а по простой интуитивной модели, на 86,7%.

Проведена оценка сокращения сложности написания программ защиты Web-сервера на языке CDTL и на Perl с помощью Метрики Холстеда и метрики Джилба. По метрике Холстеда при применении CDTL сложность разработки кода сокращается на 77,5%, а по метрике Джилба на 97,4%.

В заключении сформулированы основные результаты диссертационной работы.

В приложениях представлены: патент на изобретение, свидетельства о регистрации программ для ЭВМ и акт о внедрении результатов диссертации.

1. Анализ методов снижения числа ошибок в программном обеспечении

1.1. Виды и последствия ошибок в программном обеспечении

Виды ошибок в программном обеспечении

Универсального определение программной ошибки до сих пор ещё не было придумано, наверное, в силу того, что они могут возникать в самых неожиданных местах и допускаться в силу самых разных причин. Существует два более-менее точных определения программной ошибки:

- «Если программа не делает того, чего пользователь от нее вполне обоснованно ожидает, значит, налицо программная ошибка» [21].
- «Не существует ни абсолютного определения ошибок, ни точного критерия наличия их в программе. Можно лишь сказать, насколько программа не справляется со своей задачей, — это исключительно субъективная характеристика» [21].

Второе определение отличается от первого тем, что оно более общее и поэтому не очень подходит для точных вычислений. До сих пор существует такой спорный момент, считаются ли программными, ошибки, связанные с человеческим фактором. Что же касается видов программных ошибок, самые распространённые из них будут описаны ниже.

Арифметические ошибки

Данные ошибки связаны с какими-либо вычислениями внутри алгоритма. Например, когда вместо а+б написали а-б. Арифметическая ошибка является подвидом логической, о которых речь будет идти ниже [22].

Логические ошибки

Заключаются в том, что сам программный код срабатывает верно, однако выдаёт не то, что ожидает пользователь. Примером такой ошибки может быть неверная последовательность действий в программе или команда, похожая на нужную, но работающая немного по-другому и т. д. [22].

Ошибки взаимодействия

Появляются при несоответствии программы и её интерфейса (аппаратного или программного). Например, какие-то кнопки не работают или работают не так, как надо [22].

Ошибки компиляции

На этапе компиляции могут появляться самые разнообразные ошибки, например, синтаксические, речь о которых пойдёт ниже. Ошибки на этом этапе могут быть обнаружены как в программном коде, так и внутри самого компилятора [22].

Ошибки обработки

Обработка ошибок программы тоже нужная часть работы. Но, к сожалению, в командах, созданных для данной цели, тоже могут появляться ошибки. А бывает так, что программа просто плохо о них информирует. Например, не выдаётся сообщение с пояснениями в той части программы, где находится ошибка [23].

Ошибки перегрузки.

Появляются в том случае, если программа не справляется с возложенной на неё задачей. В принципе понятно, что у каждой программы есть свои пределы, но всё равно необходимо, чтобы они подробно были описаны в её спецификации, как и поведение программы в случае перегрузки [21].

Ошибки передачи или интерпретации данных

Появляются при неверной передаче программой каких-либо данных в процессе её исполнения. Например, при передачи каких-то файлов в них меняется кодировка или нарушается их целостность [21].

Ошибки ресурса

Появляются когда значение строковой переменной превышает максимально допустимое значение. Примером такой ошибки может служить переполнение буфера [22].

Ошибки среды выполнения (RunTime)

Данные ошибки возникают при запуске программ. Примером таких ошибок являются случаи обновления программы, вирусов, дефицита ресурсов носителя и т. д. [22].

Ошибки тестирования

Это те ошибки, которые совершает тестировщик во время проверки программы. Примером ошибки тестирования может быть неверное исправление правильно написанного кода программы, пропуск программных ошибок и т. д. [21]

Производительность

Часто в программах важно быстродействие, поэтому при его отсутствии можно низко оценивать качество данного продукта. Особенно это относится к тем случаям, когда существуют аналогичные программы, выполняющиеся с гораздо большей скоростью [21]. Например, когда медленно работает интернет браузер, долго грузится какая-нибудь программа.

Синтаксические ошибки

У любого языка программирования есть свой синтаксис. Когда программист его не придерживается, возникают соответствующие ошибки, которые называются синтаксическими. Например, неправильно написанная команда, незаданная переменная и т. д. [22].

Катастрофические последствия ошибок в ПО

Ошибки программ радиационной терапии

Один из наиболее известных случаев гибели людей, произошёл из-за смертельной дозы облучения, полученной на сеансах радиационной терапии, в процессе которых использовался медицинский ускоритель Therac-25.

В результате с июня 1985 года по январь 1987 на нём получили передозировку как минимум 6 человек и как минимум 3 из них погибли [24]. Огромное количество программных ошибок было выявлено в течении нескольких лет исследований данного инцидента сторонними экспертами.

Но главная беда в том, что данная ситуация далеко не единична [25]. В 2000 году подобную череду передозировок вызвал комплекс программ Multidata [26], также вычисляющий дозу необходимого излучения, для пациентов, проходящих курс лучевой терапии.

Ошибки программ, связанные с запуском космических аппаратов

Спутник NASA «Mars Climate Orbiter» [27] стоящий $ 125 млн разбился в 1998 году по причине того, что английские единицы измерения (фунты) не были переведены в метрическую систему. То же самое случилось с аппаратом Mars Polar Lander [28].

В течении первого запуска на космодроме Куру произошла авария ракеты «Ариан-5» [29]. Данная ракета вышла из строя на 40-й секунде запуска из-за некорректной работы бортовых программ.

Ошибки программ, связанные с запуском баллистических ракет

Из-за того, что ошибочно сработала советская система предупреждения о ракетном нападении (выдала ложный сигнал о запуске ракет с территории США 26 сентября 1983 года) [30], в мире чуть не случилась глобальная ядерная война.

Аналогичная ситуация случилась и в США 9 ноября 1979 года [31]. Спровоцировало данную ситуацию то, что оператор компьютера загрузил плёнку с учебной программой, которая моделировала массовую ракетную атаку. Но поскольку спутники и радары не передавали сигналы об атаке со стороны СССР, и быстрая перепроверка данных не выявила атаки, приказ о взаимном уничтожении не был отдан.

Ошибки программ, связанные с машинами

По заявлению Манфреда Бра, профессора информатики из Мюнхена, цена ПО и электроники в машине составляет около 40 % его рыночной стоимости [32]. И часто ошибка в ПО машины, более опасна, чем аналогичная, находящаяся в механической её части.

Была найдена ошибка в программном обеспечении 160 тысяч машин Toyota Prius, которые были выпущены в 2004-2005 годах. Для исправления данной ошибки во всех машинах пришлось потратить 240 тыс. человеко-часов [33].

7 мая 2016 года произошла трагедия, связанная с электрокаром TeslaModel S. Джошуа Браун, решил довериться автопилоту, будучи уверен в качестве данной марки машины, и в результате попал в автокатастрофу, из-за чего скончался на месте от сильных травм. [33].

Ошибки программ, связанные с электронной торговлей.

В 2012 году компания Knight Capital Group потеряла 440 миллионов долларов, после того, как продала все акции, случайно купленные в среду утром из-за компьютерного сбоя [34].

1.2. Анализ существующих методов снижения числа ошибок.

Наилучший результат по снижению числа программных ошибок на сегодняшний день зафиксирован для системы NASA JPL (Jet Propulsion Laboratory) – 0,003 ошибки на 1000 строк кода [35]. Среднее количество дефектов для Linux, PHP, и PostgreSQL на 1000 строк кода равно 0.62, 0.20 и 0.21 соответственно [36], а ядро Windows имеет 0,5 ошибки на 1000 строк [35] с учетом того, что ядро Windows содержит более 40 млн. строк кода. Поэтому в диссертации поставлена задача создания системного программного обеспечения, позволяющего программисту снижать число допускаемых ошибок.

Решению задач сокращения ошибок в коде с помощью статических и динамических анализаторов и правил Хольцмана[37] посвящены работы отечественных и зарубежных ученых G.J. Holzmann[37], Белеванцев А. А.[38], Бородин А. Е.[38], Игнатьев В. Н.[38], Журихин Д. М.[38], Аветисян А. И.[38], Герасимов А. Ю.[39], Горемыкин А. В.[40], Вартанов С. П.[40], Меньшиков М.А.[41], Беляев М.В.[42], Романенков Е.С.[42], Игнатьев Н.В.[42], Dimovski, A.S.[43], Apel, S.[43], Legay, A.[43], Chen, T.[44], Heo, K.[44], Raghothaman, M.[44]. Задача разработки кода с помощью нейронных сетей решается

исследователями из Microsoft и Кембриджского университета. Нейросеть DeepCoder[45] способна создавать программы на алгоритмическом языке длиной максимум 5 строк кода.

Далее хотелось бы подробнее остановиться и подробнее рассмотреть некоторые из известных методов сокращения числа программных ошибок.

Отладчики кода, статические и динамические анализаторы

Отладка — необходимая стадия в области разработки ПО, на которой выявляют и исправляют ошибки. Очень важен правильный выбор инструмента отладки и понимание того, как он функционирует.

Отладчик – инструмент для проверки программ в текущем времени, с помощью которого можно проводить пошаговую трассировку, узнавать, чему равны переменные на том или ином этапе работы программы и т. д. С помощью отладчика можно отслеживать работу программы по шагам, имея возможность в любое время получить все необходимые данные о текущем функционировании программы или корректировать алгоритм её выполнения.

Отладчики бывают 3-х видов [46]:

1. Отладчики режима пользователя [47].
2. Отладчики режима ядра [46].
3. Эмулирующие отладчики [46].

Статический анализатор – это программа, которая ищет ошибки внутри кода другой программы без её фактического запуска [48]. Обычно проверяется большое количество различных вариантов её исполнения. Поэтому такие анализаторы способны найти ошибки при редких вариантах работы какой-либо программы. Такие ошибки часто не бывают выявлены во время тестирования программы и при этом могут иметь серьёзные последствия (см. раздел «последствия ошибок»). Другой положительной стороной статических анализаторов является поиск места ошибки. При нахождении ошибки статический анализатор сразу указывает её положение и причину.

В России ведутся разработки статических анализаторов в институте системного программирования РАН [49].

Динамический анализатор ищет ошибки в процессе исполнения программы. Сейчас особое внимание уделяется методам объединения статического и динамического анализа программ [39].

Правила Хольцмана

Правила Хольцмана [37] - специально созданные правила для предотвращения ошибок в коде программы. Данными правилами пользовались при создании программ, предназначенных для управления американскими шаттлами. Вероятность ошибки в данных программах оказалась очень низкая - 0,003 на 1000 строк кода. Такое количество ошибок является более низким, чем в ядре операционной системы Windows (0,5 ошибки на 1000 строк кода). Поэтому данными правилами пользуются разработчики программ при создании критически важного ПО.

Некоторые из правил Хольцмана обеспечивают создание чёткой и прозрачной структуры потока управления данными, которая облегчает разработку, тестирование, а также анализ ПО в дальнейшем. Отказ от выделения динамической памяти, прописанный в третьем правиле, убирает риск огромного числа ошибок с ней связанных. Например, разработчик может забыть освободить память, или попробовать выделить больше памяти, чем ему доступно физически, или пересечь границы выделенной памяти, или по ошибке поместить инструкцию выделения памяти внутрь цикла и т. д. Некоторые из нижеследующих правил Хольцмана используются в качестве стандартов для создания качественного программного кода. Другая часть правил предназначена для очерчивания более строгих рамок кодирования.

Столкнувшись с данными правилами, разработчики часто не хотят ими пользоваться из-за наличия очень строгих ограничений. Однако впоследствии обнаруживается, что их соблюдение приносит огромные преимущества в плане безопасности кода. Помимо самого уменьшения ошибок, соблюдение правил

Хольцмана существенно снижает нагрузку на разработчиков и тестировщиков ПО.

Поэтому хоть сначала к этим правилам тяжело привыкнуть, необходимо их использовать при написании программного обеспечения, ошибки в котором могут иметь серьёзные последствия. Да и при написании кода обычной программы эти правила могут помочь.

*Принцип «М*ножества *глаз» (англ. «Many eyes»)*

Одним из наиболее эффективных способов сокращения числа программных ошибок на сегодняшний день является метод «Множества глаз» или «Many eyes». Он заключается в том, что вероятность детектирования ошибки, прямо пропорциональна количеству использования людьми программы. Данный метод работает таким образом, что люди, в процессе использования программного кода находят в нём ошибки и исправляют их. И по этой причине, их количество может сильно сокращаться.

“В качестве примера можно рассмотреть функцию *sin(x)*. Она проверена на огромном разнообразии значений в невероятном количестве пользовательских задач, что позволяет утверждать, что ошибки в этом модуле маловероятны” [20].

Для реализации данного принципа разработчики создают проекты с открытым исходным кодом [50,51]. В таких проектах помимо того, что ошибки находятся и исправляются пользователями, программисты и сами стараются создать код хорошего качества.

Данный принцип в том числе реализует и компания Linux, предоставляя код ядра операционной системы в открытый доступ [19,52,53].

Несмотря на это существует мнение разработчиков о том, что данный метод может не работать потому, что люди используют код без просмотра. Наверное, данное явление может существовать, однако те эксперты, которым для решения своих задач надо будет понять алгоритм кода всё равно могут сильно помочь разработчикам в устранении ошибок и тем самым улучшить качество программ.

Генераторы тестов

Генерация тестов – это программное создание тестовых примеров, с помощью которых можно понять, исполняется ли без ошибок проверяемый код или в нём всё же есть какие-то недочёты.

Минимальный набор генераторов тестов должен находиться у пользователей программы для того, чтобы они могли проверять рабочие версии программ в процессе их исполнения. Он находится в комплекте поставки пользовательских версий множества программ. Для того, чтобы провести более глубокое исследование того, как работают версии и где в основном находятся ошибки, нужно создать набор средств, которые будут имитировать рабочую среду на специализированной ЭВМ. Такие ЭВМ применяются специалистами в области испытаний и сертификации [54]. Частично они могут применяться для проверки и имитации особенностей внешней среды, при которых пользователи нашли ошибки.

Средства для создания тестовых примеров, а также обработки результатов отладки, подразделяются на 3 группы:

- Средства автоматизации тестирования в статике [54].
- Средства, имитирующие внешнюю среду в реальном времени [54]. Обычно используются для отладки как отдельных программных модулей, так и программного средства полностью.
- Реальные объекты внешней среды [54]. Например, аппаратные или компьютерные комплексы, на которых будет применяться данное программное средство.

Основное внимание в вышеперечисленных работах уделено алгоритмическим языкам программирования, что ограничивает возможности создания нужного программного продукта. В моей диссертационной работе описаны некоторые способы решения данной проблемы.

1.3. Способы автоматизации разработки программного обеспечения

В процессе автоматизации ПО можно выделить следующие актуальные тенденции[55]:

- создание шаблонов, способов и правил разработки программного обеспечения, стандартов, методик и библиотек;
- скрытие и объединение деталей в некие наборы, которые применяются по принципу конструктора;
- замещение компьютерных приложений для разработки и развертывания облачными сервисами;
- использование баз знаний, содержащих данные о программном модуле на всех стадиях разработки;
- популяризация декларативного программирования, особенно функциональных языков;
- разработка языков и способов написания кода, содержащих более чем одну парадигму программирования;
- описание всех этапов разработки ПО с помощью единой сервисной модели;
- разделение среди ИТ-профессионалов по используемым программным средствам и языкам;
- использование нейросетей для генерации программного кода [45];
- определение контекста слов с помощью программ. Данный метод можно применять для автоматизации ПО, поскольку в зависимости от смысла описания или кода программы может зависеть то, какие действия она должна выполнить.

На основании данных тенденций можно сформулировать основное направление развития автоматизации программных средств. Оно характеризуется непрерывным увеличением уровня абстракции. Задача, поставленная в диссертационной работе, отлично вписывается в это направление. Наибольший успех в автоматизации достигнут с помощью специально обученных нейронных сетей, которые сами могут генерировать

программный код. Методы обучения таких сетей называются “Автоматизированное машинное обучение”. Ниже будут описаны 2 метода, наиболее близкие к работам, выполненным в рамках данного диссертационного исследования. Это “*Автоматизированное машинное обучение”* и *“Определение контекста слов с помощью программ.*

Автоматизированное машинное обучение ***(****AutoML или Automated Machine Learning)”* [56] – специально подготовленные методы машинного обучения, которые могут применять люди без обладания специализированными знаниями и навыками.

Автоматизированное машинное обучение занимается разработкой и реализацией методов и алгоритмов, которые могут сделать Machine Learning более доступным для специалистов, область деятельности которых с ним не связана. AutoML занимается разработкой методов повышения эффективности машинного обучения и увеличением скорости проведения исследований в сфере Machine Learning.

Нейросеть DeepCoder, была создана исследователями из Microsoft и Кембриджского университета. Она из строчек кода других программ генерирует свой оригинальный код. Строки кода, подходящие для программы, определяются на основании входных и выходных значений каждого фрагмента кода. В настоящее время эта нейросеть научилась писать программы длинной максимум 5 строк кода [45]. Но большинство программ включают в себя гораздо больше строк, поэтому создавать их с помощью данного метода не получится.

Проблема работы с нейронными сетями заключается в том, что они созданы по образу и подобию человеческого мозга, поэтому скорее всего склонны делать те же ошибки, что и люди.

А люди склонны делать очень много ошибок в программах. Поэтому такой метод сокращения числа ошибок не очень эффективен для решения данной проблемы, несмотря на то, что разработки Google и Microsoft несомненно

являются прорывом в области deep learning и очень близки к проблеме, которая решается в данной работе.

Определение контекста слов с помощью программ

Поскольку один из таких способов был разработан в этой диссертационной работе, остановлюсь на нём подробнее.

В наше время проблема распознавания контекста слов компьютером весьма актуальна [57,58]. Она важна для поисковых систем, машинного перевода, интерпретации текста при грамматическом разборе и в машинном анализе содержания документов.

Проблема определения контекста слова, на данный момент, относится к AI-полным задачам [59], требующим сильного искусственного интеллекта. Повышение удобства взаимодействия компьютера и человека в данной области определяет эффективность тех или иных решений.

Слова современного русского языка подразделяются на однозначные и многозначные. Однозначными слова выступают в том случае, когда они имеют только одно значение. Это явление называется моносемией [60]. Примеры таких слов: Москва, дуб, ведро, макароны, чайник и т. д.

Однако большинство слов русского языка имеют более одного значения. Они называются полисемантическими или многозначными и противопоставляются однозначным словам. Способность какой-либо лексической единицы иметь несколько вариантов смысловых значений называется контекстной многозначностью или полисемией [61].

Благодаря существованию полисемии [62], одно и то же слово может употребляться в разных значениях. Например, слово «ключ» может иметь значения ключ от замка или ключ родник, или криптоключ. Человек может определить контекстное значение слова, анализируя соседние слова в предложении и сам текст в целом.

Для корректной работы программы, созданной с помощью языка описания задачи, правильное определение контекста важно также по той причине, что при

неверном истолковании машиной человеческой речи, возможна некорректная генерация её кода. Чтобы таких проблем было как можно меньше, нужно постараться сократить возможность непонимания компьютерной программой текста, написанного человеком.

Для того, чтобы решить данную проблему, нами был разработан метод определения контекста слов и текстовых файлов. Данный метод хорош тем, что по сравнению с уже существующими методами, более прост в вычислении и не требует большого количества ресурсов. Далее в данном разделе, речь пойдет о существующих методах определения контекстной близости файлов и слов и некоторых понятиях, встречающихся в данной области. Описание самого метода будет дано в главе 3.

Определение контекстной близости текста внутри файлов

Исследователи подразделяют контексты на 3 вида согласно их длине:

- микро-контекст [63] (определяется для слов, находящихся около целевого слова),
- тематический контекст [63] (определяется для предложений, находящихся рядом с целевым словом)
- контекст области знаний, которая является частью текстового файла или же составляет всё его содержимое, для которой нужно узнать семантическое значение [63]

Методы, с помощью которых определяют тематическую близость текстовых файлов в настоящее время в основном применяются и разрабатываются для задач, связанных с поиском информации.

Проблемы и методы определения контекста слова

Данной проблемой занимается *NLP* (*Natural Language Processing* или *Обработка естественного языка*) — общее направление искусственного интеллекта и математической лингвистики, занимающееся изучением компьютерного анализа и синтеза естественных языков [64].

Для искусственного интеллекта под анализом подразумевается понимание языка, а под синтезом – генерация грамотного текста. Повышение удобства взаимодействия компьютера и человека в данной области является решением этих проблем.

Их пытаются решать, например, с помощью семантических сетей и онтологий [65]. В информатике данным словом называют попытку представления некоторой области знаний в виде формальной системы, пользуясь для этого концептуальной схемой. Концептуальная схема – это схема, которая подразумевает какую-то идею или понятие (схема понятий).

Онтологии тоже подразделяются на разнообразные виды. Например:

1. *Тезаурус* - словарь, в котором указаны семантические отношения (синонимы, антонимы, паронимы, гипонимы, гиперонимы и т. п.) [66]
2. *Словарь* – список терминов с определённым значением.

И т. д

«Семантическая сеть - это информационная модель предметной области, представляющая собой ориентированный граф, вершины которого - объекты предметной области (понятия, события, свойства, процессы), а рёбра – связи между ними» [7,67].

«Понятиями называются отражённые в мышлении самые важные свойства, связи и отношения предметов или явлений. Понятием также является мысль или система мыслей, которая выделяет и обобщает предметы определённой группы согласно одинаковым, а также и специфическим признакам, в отношении них» [7,31]. Одна из самых популярных, в настоящее время семантических сетей – сеть WordNet [68,69].

К проблеме определения контекста слова подходили и с помощью корпусных методов. Данными методами занимается дистрибутивная семантика. Дистрибутивная семантика – область лингвистики, которая на основании распределения лингвистических единиц в массивах лингвистических данных

большого размера, занимается расчётом степени семантической близости между ними. [70].

Корпусом в лингвистике называется, используемая как база для изучения языка, а также выбранная по специальным правилам, совокупность текстов. В частности, они используются для изучения статистических гипотез, а также лингвистических правил в каком-либо естественном языке.

“Корпусные методы разрешения многозначности — это класс алгоритмов построения классификатора на основе размеченного вручную набора текстов, использующих методики машинного обучения” [71]. Значимые компоненты данных методов - выборка текстов, словарь значений, синтаксический анализатор. С помощью этих методов обычно изучается всего лишь одно многозначное слово в тексте, причём какой-то заданной части речи, посредством категоризации.

Обзор патентов, близких к методу определения контекста, описанному в данной работе

Изобретение, описанное в патентном документе RU 2632126, опубл. 02.10.2017 [72], относится к средствам предоставления контекстуальной информации, относящейся к документу, с использованием семантической сети. Однако в данном документе не определено, как оценивается контекстуальная релевантность объекта и как выполняется контекстуальный поиск. Из приведенного описания указанного документа ясно, что в оценку контекста вовлечен человек, который передает серверу нужные данные [7].

В отличие от патентного документа RU 2632126, опубл. 02.10.2017 [72], в предложенных в этой работе способах, вся работа происходит на одном компьютере, где анализируется текст и определяется контекстное значение слов и текстовых файлов. В том числе, за счет формулирования числового способа оценки контекста, предложенная методика применима как к отдельным словам или текстовым фрагментам, так и к тексту в целом, что позволяет эффективно

определить контекст, предоставляя возможность числовой оценки вероятности правильности выбора контекстного значения.

Наиболее близким решением является изобретение, описанное в патентном документе US 9,632,999 B2, опубл. 25.04.2017 [73], который относится к определению контекста текстового фрагмента с помощью семантического анализатора и семантических сетей. Текст в нём анализируется слово-за-словом с привлечением технологии семантических сетей, где учитываются смысловые связи отдельных слов [7].

Основными недостатками указанного патентного документа US 9,632,999 B2, опубл. 25.04.2017 [73], являются следующие [7]:

- способ применим скорее к фрагментам текста, чем к отдельным словам;
- способ довольно трудоёмок и по этой причине с помощью него сложно определять контекст больших фрагментов текста;
- определяется только приблизительный смысл слова.

Техническая проблема заключается в трудоёмкости проведения семантического анализа и связана с тем, что обработка текста требует слишком больших временных и машинных ресурсов [7].

1.4. Возможности декларативного программирования и метапрограммирования

Парадигма программирования - это “набор представлений о некотором классе программных систем, допускающих реализацию с помощью этой парадигмы набора представлений о способе программирования”. Парадигмы программирования могут быть разными. Императивное программирование, декларативное и т. д. В данном разделе будет описано декларативное программирование, так как языки, написанные этим способом программирования, являются предшественниками CDTL. [74]

При использовании декларативного программирования обозначается заранее понятный результат, например, статическая страничка, написанная

на HTML. Как ещё один пример декларативного языка программирования можно привести язык «Норма», речь о котором пойдёт ниже. Помимо этого, языка будет рассмотрено функциональное программирование, которое также является подвидом декларативного. *Декларативный язык НОРМА* был разработан в ИПМ им. М.В. Келдыша РАН для параллельных компьютеров различного типа. [75]

Прикладной специалист может записать свои математические формулы с применением языка НОРМА, а после компилятор данного языка создаёт программу на Си. С помощью этого языка можно описать решения разнообразного спектра задач математической физики. Описание на языке НОРМА не ориентировано на конкретную архитектуру вычислительной системы, по этой причине оно имеет большие возможности для выявления естественного параллелизма

Функциональное программирование

В функциональном программировании описание программы происходит с помощью математических функций. Функциональная программа имеет довольно несложную форму: последовательность определений функций, после которой идёт последовательность вызовов этих функций [76]. Вычисления запускаются после вызова какой-то функции, которая запускает другие функции и т.д. Каждый вызов отправляет некое значение в запускающую функцию, расчёт которой после этого не останавливается. Данный процесс повторяется до того, как запустившая расчёты функция не выдаст итоговый результат.

Функции только получают входные данные и выдают какой-либо результат. Взаимодействие между функциями происходит только с помощью вызовов в процессе выполнения, однако последовательность вычисления подвыражений не влияет на результат. Итог выполнения функциональной программы – значение выражения, которое можно определить терминами базовых и/или определенных пользователем программы функций. [76].

В настоящее время увеличился интерес к функциональному программированию. Выросло количество людей, интересующихся методологией и технологией написания программного кода с применением функциональной парадигмы. Она хорошо подходит и используется для создания систем искусственного интеллекта, разработки графических интерфейсов, создания графических приложений реального времени, разработки систем параллельных вычислений, построения компиляторов [76].

Далее будут рассмотрены 2 языка функционального программирования, ЛИСП (первый функциональный язык) и Wolfram Language (является предшественником разработок, описанных в данной диссертации).

Язык ЛИСП

Лисп утратил главенствующую роль в области программирования задач искусственного интеллекта, он очень удобен для изучения процесса функционального программирования [77].

Язык Wolfram Language и система Mathematica

Язык Wolfram Language [78] – это символьный язык программирования, разработанный для системы Mathematica. Wolfram Language является алгоритмическим языком, с интегрированным машинным обучением. Изначально данный язык планировался для того, чтобы выполнять математические вычисления и строился на фундаменте наиболее важных идей о символьных системах [79]. Он был создан Стивеном Вольфрамом более 30 лет назад, и на сегодняшний день в нём насчитываются миллиарды строк кода. Впервые принципы этого языка были применены в системе Mathematica для осуществления серьёзных математических расчетов.

Метапрограммирование

Метапрограммирование (МП) - это процесс генерации метапрограмм для создания или изменения кода целевой программы [80]. Метапрограммы - программы, в процессе работы которых генерируется код целевой программы.

Язык CDTL можно назвать языком метапрограммирования, так как из описаний, созданных на нём, генерируется код на языке программирования высокого уровня.

1.5. Критерии оценки надёжности и сложности ПО. Обоснование и выбор моделей для оценки количества ошибок в ПО

Модели оценки надёжности ПО

Надёжность ПО – “способность системы или компонента выполнять требуемые функции в заданных условиях на протяжении указанного периода времени” [81].

Методы оценки надёжности ПО в первую очередь подразделяются на *аналитические* и *эмпирические* [54]. А сами аналитические модели подразделяются ещё на 2 категории: *динамические* и *статические.*

Динамические методы работают с результатами выполнения программы [82]. С помощью данных методов можно оценить “абсолютные показатели надежности ПО” [82].

В мире используются следующие динамические модели: Модель Шумана [83], Модель Муса [81], Модель Ла Падула [81], Модель Джелинского — Моранды [81], Модель Шика — Волвертона [81], Модель переходных вероятностей [81].

Статические методы не похожи на динамические в первую очередь тем, что в них не рассматривается время возникновения ошибок. Они опираются на изучение разных объектов процесса проектирования, например, спецификаций или кода программы. С помощью данных методов можно выполнить оценку кода программы автоматически, поэтому они имеют минимальную трудоёмкость.

Существуют следующие статические модели определения количества ошибок в программах: Модель Миллса [50], Модель Липова [81], Простая

интуитивная модель, Модель Коркорэна [81], Модель Бернулли [81], Модель Нельсона [81].

В рамках данного исследования для оценки количества ошибок была применена простая интуитивная модель.

Простая интуитивная модель

Применение данной модели подразумевает проведение тестов программистами из 2-х групп, которые при этом работают с не связанными между собой тестовыми наборами. Во время проведения данных тестов каждая из этих групп записывает все обнаруженные ошибки. Для того, чтобы спрогнозировать количество не найденных ошибок, сравниваются между собой тестовые данные, полученные этими группами. [81]

Если считать шансы нахождения общего числа ошибок равными для этих 2-х групп, то можно предположить, что раз первая из групп выявила конкретный их процент, она могла бы проделать то же самое с другим кодом. [81]

$$N = \frac{N_1 * N_2}{N_{12}} \qquad [81] \qquad (1.1)$$

N – количество ошибок в программе

N_1 – количество ошибок, найденное первой группой

N_2 – количество ошибок, найденное второй группой

N_{12} – количество ошибок, найденных и первой и второй группой.

Эмпирические методы основываются на изучении структуры программ. Они во время проектирования работают с такими данными как информация о том, как структурированы процессы проектирования, сертификаты, уровни зрелости [82] и о работе предыдущих проектов. С помощью этих моделей в большинстве случаев нельзя получить окончательные результаты показателей надежности, но их применение удобно для прогнозирования необходимых ресурсов для проведения тестирования, уточнения плановых сроков завершения проекта и т. д. [84].

Модель сложности

Существуют разные виды данной модели. В каждом из этих видов определяется сложность программы, она предполагается пропорциональной её надежности.

Сложность программ определяется вычислением количества программных модулей и сложностью устройства интерфейсов, находящихся между ними.

Программный модуль – это единица, которая выполняет некую часть кода и которая взаимосвязана с другими такими же единицами (модулями этой программы) [81].

В рамках данной модели для определения количества ошибок используется Метрика Холстеда. Количество ошибок (B) в ней рассчитывается таким способом:

$$B = V/3000 \text{ [85]} \qquad (1.2)$$

V-объём программы, главная характеристика данной модели, потому что благодаря ней вычисляется количество потенциальных ошибок в программе [85] Согласно данной модели считается, что в программе, написанной на языке программирования высокого уровня на 3000 бит кода приходится одна ошибка.

Модель, определяющая время доводки программ

Расчёт оценки взаимосвязей модулей ПО, базируется на таком положении, что любая пара модулей обладает конечной вероятностью того, что при изменениях внутри одного модуля, автоматически поменяется второй. Данная модель создана для того, чтобы уже при тестовой компоновке системы можно было приблизительно посчитать какое количество исправлений потребуется, а также время на него затраченное [81].

Для того, чтобы оценить уменьшение ошибок в данной работе выбраны 2 способа, которые подробно описаны выше. Это простая интуитивная модель и метрика Холстеда. При выборе сразу же были исключены динамические методы, потому что они анализируют количество ошибок во времени, а надо было сравнить исходный код. Из статических методов была выбрана простая

интуитивная модель, потому что имелись ресурсы, необходимые для реализации расчётов с её использованием, и можно было получить нужную информацию по количеству ошибок. Помимо этого, применить её было проще чем другие, упомянутые выше, модели. Метрика Холстеда была выбрана потому что она подходит для решаемой задачи и с помощью неё довольно просто сделать нужные расчёты. Подробнее про применение этих метрик написано в четвёртой главе (раздел 4.5)

Количественные метрики оценки сложности программ

Количественные характеристики программ обычно широко применяются из-за их простоты. Эти метрики рассчитываются на основе исходного кода программ, что не подходит для анализа бинарного кода приложений.

- *Количество строк кода* (SLOC – Source Lines Of Code). Если данная метрика применяется к бинарному коду, она должна определяться числом инструкций, которые находятся в ассемблерном листинге. Однако ценность этой метрики невысока. Для того, чтобы построить профиль сложности может быть применен такой вид этой метрики, как количество инструкций базового блока – чем меньше базовые блоки, тем относительно чаще встречаются передачи управления в коде. [86]
- *Среднее число строк для функций* (классов, файлов, модулей). Для бинарного кода – среднее количество инструкций внутри функции. Возможная вариация – средний размер базового блока внутри функции. [86]
- *Метрики Холстеда* (Halstead) позволяют частично учитывать возможность записи одной и той же функциональности программы различным количеством строк и операторов. Они основаны на разных количественных показателях, таких, как число уникальных операторов программы, число уникальных операндов программы, общее число операторов, общее число операндов, теоретическое число уникальных операторов и теоретическое число уникальных операндов [86]. С помощью этих показателей различными формулами вычисляются уровень качества программирования,

сложность понимания программы, трудоемкость кодирования программы, уровень языка выражения, информационное содержание программы и оценка интеллектуальных усилий при разработке программы [86], которая характеризует количество необходимых элементарных решений при создании кода программы.

Метрика Холстеда вычисляется как [87]:

(Halstead Difficulty, HDiff): HDiff = (NUOprtr/2) × (NOprnd / NUOprnd) [87];

NUOprtr (Number of Unique Operators) — количество уникальных операторов программы, в том числе символы-разделители, имена процедур и знаки операций;

NUOprnd (Number of Unique Operands) — количество уникальных операндов программы;

Noprnd (Number of Operands) — общее количество операндов в программе.

- *Метрики Джилба* (Jilb) показывают сложность программного обеспечения с помощью наполненности кода программы условными операторами, а также операторами цикла. CL – абсолютная сложность (количество управляющих операторов), cl=CL/n, где n – общее число операторов программы – относительная сложность программы по Джилбу [86]. Несмотря на простоту, метрика Джилба отлично отражает как трудоемкость разработки, так и сложность понимания кода программы. При присоединении показателя максимального уровня вложенности условных операторов и циклов, эффективность этой метрики очень сильно возрастает. Лёгкость расчёта метрик Джилба, гибкость в выборе оцениваемых конструкций, возможность увеличения оценки из-за введения показателей сложности и возможность расчёта относительной меры, делают эту метрику подходящей для задач анализа бинарного кода программных приложений.
- ABC-метрика. Данная величина основана на подсчете присваиваний значений переменным (Assignment), явных передач управления за пределы области видимости (она связана с особенностями команд процессора), т.е. вызовов

функций (Branch), и логических проверок (Condition)" [86]. Эта мера просто вычисляется, может быть рассчитана для разных частей кода.

Для расчёта сложности программного кода были выбраны метрики Холстеда и Джилба. Данный выбор был сделан потому, что эти метрики просты в использовании, не требуют специальных ресурсов и с помощью них можно было сделать необходимые расчёты. Метрика Джилба использовалась для уточнения результатов метрики Холстеда.

1.6. Снижение числа ошибок в программном обеспечении на основе сокращения степени участия программиста

Сегодня от качества компьютерных программ зависят практически все стороны человеческой жизни, включая здоровье [88]. По этой причине важно максимально сократить число программных ошибок. В этой работе рассматривается частная задача оптимизации программирования с целью снижения числа ошибок. Более общая задача уже была рассмотрена в [38, 89, 90, 91, 92].

Впервые данная тема была поднята в 2013 году на семинаре ВМК МГУ, где обзор по проблемам программных ошибок делал В.П. Иванников.

Главной целью моих исследований является снижение числа программных ошибок. Проблема в том, что полностью выявить и устранить ошибки невозможно. Не существует даже хороших методов оценки числа имеющихся программных ошибок. Одним из эффективных проверенных способов сокращения числа ошибок является использование программы большим числом пользователей с последовательным выявлением и корректировкой ошибок (принцип «Множества глаз»).

Одним из мотивов программы «open source» является снижение количества программных ошибок за счет многократной проверки для самых разных приложений. Другим подходом, который можно совмещать с методикой

«Множества глаз», является сокращение участия человека в написании программы.

В процессе формирования исполняемого программного модуля могут использоваться библиотечные программы, макросы и т.д.

Язык описания задачи CDTL на сегодняшнем уровне реализации является макроязыком, который работает поверх Perl.

Вероятность найти нужный программный модуль в банке интуитивно пропорциональна логарифму числа модулей.

Ошибки могут возникнуть при неправильном указании имени примитива или макроса. Последствия таких ошибок будут тяжелыми. Выявить их можно только с помощью специализированных тестовых программ, которые должен создать разработчик. Задания для таких программ может быть описано на языке описания задачи.

В варианте задачи обеспечения безопасности WEB-сервера источниками угроз могут быть:

а. Фишинг (попытка убедить пользователя-жертву загрузить вредоносный программный модуль на свой компьютер). Именно такие модули ищут хакеры на WEB-серверах.

б. Атака переполнения буфера. Запрос get, где в конце списка параметров помещен исполняемый код.

в. Атака path_traversal (попытка попасть в каталог за пределами разрешенной зоны, заданной при конфигурации Apache).

г. Атака SQL-injection (если на сервере предусмотрен доступ к какой-то базе данных)

д. Атаки типа DoS (слишком высокая частота запросов, или в запросе много последовательностей ./../ - path_traversal)

Все эти атаки не регистрируются антивирусными программами. Антивирусы работают только с сигнатурами и по необычности поведения программы.

По данным Proofpoint, после того как компания Microsoft прекратила применение макросов VBA и XL4 для приложений Windows Office, атакующие для транспортировки вредоносных кодов стали использовать контейнеры ISO и RAR [93]. Microsoft блокировала VBA-макросы с помощью атрибута MOTW (Mark of the Web). Но такая блокировка может быть обойдена с помощью контейнеров типа ISO (.iso), RAR (.rar), ZIP (.zip) и IMG (.img). Для доставки жертве вредоносного кода используются HTML-приложения к передаваемому документу. С октября 2021 по июнь 2022 число таких атак удвоилось.

Описание алгоритма на любом алгоритмическом языке содержит много малосущественных деталей. Рассмотрим пример системы WEB-безопасности.

Система не может работать без скрипта filter, так как из-за ограничений быстродействия она неизбежно зависнет. Темп роста размера журнального файла access_log будет превосходить скорость его обработки.

Но, если бы мы имели компьютер с быстродействием в 10 раз больше, скрипт filter был бы не нужен. С другой стороны, если поток запросов возрастет на порядок, это не поможет, и система все равно повиснет. По существу, это классический вариант программы реального времени, где результат зависит от состояния системы в целом. В наших программах WEB-безопасности критическую опасность создают слишком длинные списки параметров GET-запроса (попытка атаки переполнения буфера). Разумеется, и слишком частые запросы опасны, но они не могут разрушить нашу систему. Они могут “повесить” систему за счет слишком быстрого роста журнального файла, но при спаде темпа запросов, система восстановит функциональность. А атака переполнения буфера может систему разрушить необратимо.

При анализе задачи всегда известно значение области проблематики, но не всегда известен алгоритм решения. Существуют задачи, для которых вообще нет решения.

1.7 Постановка задач работы

Проведенный системный анализ проблемы снижения трудоёмкости написания программного кода привёл к следующему дизайну исследования (рис. 1.2).

Рис. 1.2 Дизайн исследования

Целью работы является разработка математического и программного обеспечения динамически проектируемого транслятора со специального метаязыка описания задачи на язык высокого уровня для снижения трудоемкости генерации кода.

Задачи исследования. Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать синтаксис метаязыка описания задачи для генерации его с помощью транслятора в язык программирования высокого уровня.

2. Разработать алгоритм динамического выбора состава модулей для автоматического выбора способа решения задачи в рамках коллектива задач одного типа и сокращения количества модулей в базе данных.

3. Разработать алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных для получения корректного результата в большинстве случаев без больших затрат ресурсов и сложных вычислений.

4. Разработать структуру динамического транслятора описания задачи в язык Perl, обеспечивающую автоматическое проектирование программы решения задачи на языке высокого уровня.

2. Исследование и разработка метаязыка описания задач для автоматизации программирования

2.1 Технологии снижения числа программных ошибок в процессе создания кода, использующие сокращение участия программиста

Диалоговый метод

Одним из путей полуавтоматического формирования программы является диалог между компьютером и программистом. Этот метод применим, когда такой диалог может быть описан древовидным графом и имеется набор готовых программных модулей в банке алгоритмов. Данный метод можно применять как инструмент взаимодействия программ и программных систем.

Если сама программа может быть создана в диалоговом режиме, то конечный узел, соответствующий имени проблемы, будет корневым узлом диалога создания программы (рис. 2.1).

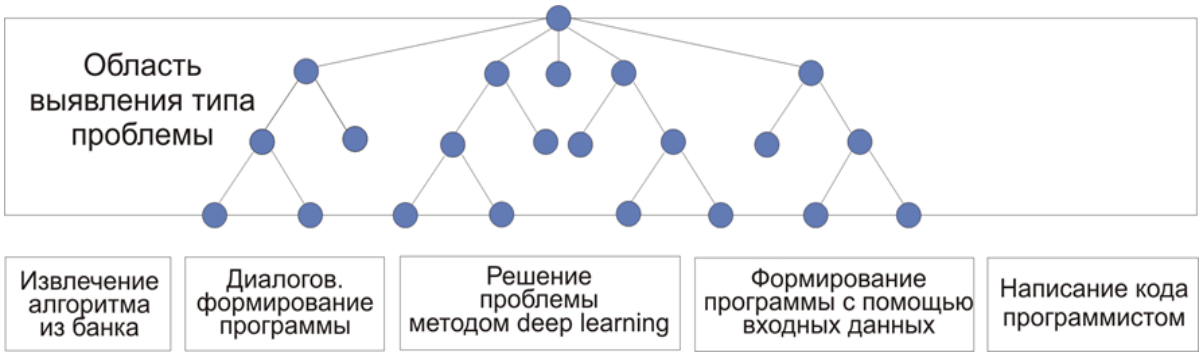

Рис. 2.1 Метод графа для описания проблемы в диалоговом режиме

Прямоугольники в нижней части рис. 2.1 соответствуют разным типам техники создания программы. Вариант метода deep learning используется для проблем распознавания (SPAM, атаки нулевого дня и пр.).

Оптимально формирование списка ключевых слов для поиска программного модуля посредством диалога (вопрос-ответ). Диалог при составлении списка ключевых слов для банка алгоритмов и при формировании запроса поиска в банке алгоритмов должен проводиться идентично, что будет гарантировать идентичность списков ключевых слов. В случае регулярного выражения метод древовидного графа успешно работает (рис. 2.2). Дерево графа

определяет число вариантов регулярных выражений, которые могут быть сформированы.

Каждая диалоговая программа ориентирована на создание определенного класса кодов. Число таких программ будет достаточно велико, и со временем увеличиваться.

Программа или конкретный человек может сформировать свой список ключевых слов, характеризующих алгоритм, который нужен $W_i=(w_{1,i}, w_{2,i}, w_{3,i}, \ldots, w_{m,i})$, где i – номер запроса алгоритма в банк данных, а m – число ключевых слов в списке. Причем, этот список может отличаться как по содержанию, так и по длине от списка, хранящегося в банке описаний алгоритмов. Задача может осложняться существованием синонимов для некоторых ключевых слов, а также наличием опечаток. Для исключения опечаток надо обязательно проводить орфографический контроль. Порядок ключевых слов в запросе и в поле описания банка данных при такой схеме будет практически всегда отличаться.

Если узел графа (рис. 2.2) однозначно определяет регулярное выражение, то ключевые слова будут не нужны, да и сама база данных или хэш-массив также будут не нужны, так как граф будет выходить непосредственно на регулярное выражение. На рис. 2.2 приведен фрагмент дерева для получения регулярного выражения. Программа, реализующая алгоритм, отображенный на рис. 2.2, может быть также помещена в банк алгоритмов.

Программы диалоговой техники программирования могут создаваться с привлечением шаблонных схем, куда программист заносит тексты вопросов и возможных ответов в поля определенной формы, подготавливая программу диалога.

Надо заметить, что диалоговый метод обладает одним существенным недостатком. Все варианты диалога описать и предугадать нереально, потому что их число стремится к бесконечности. Но, тем не менее, этот метод успешно может применяться для решения простых задач с ограниченным количеством ветвлений диалогового графа.

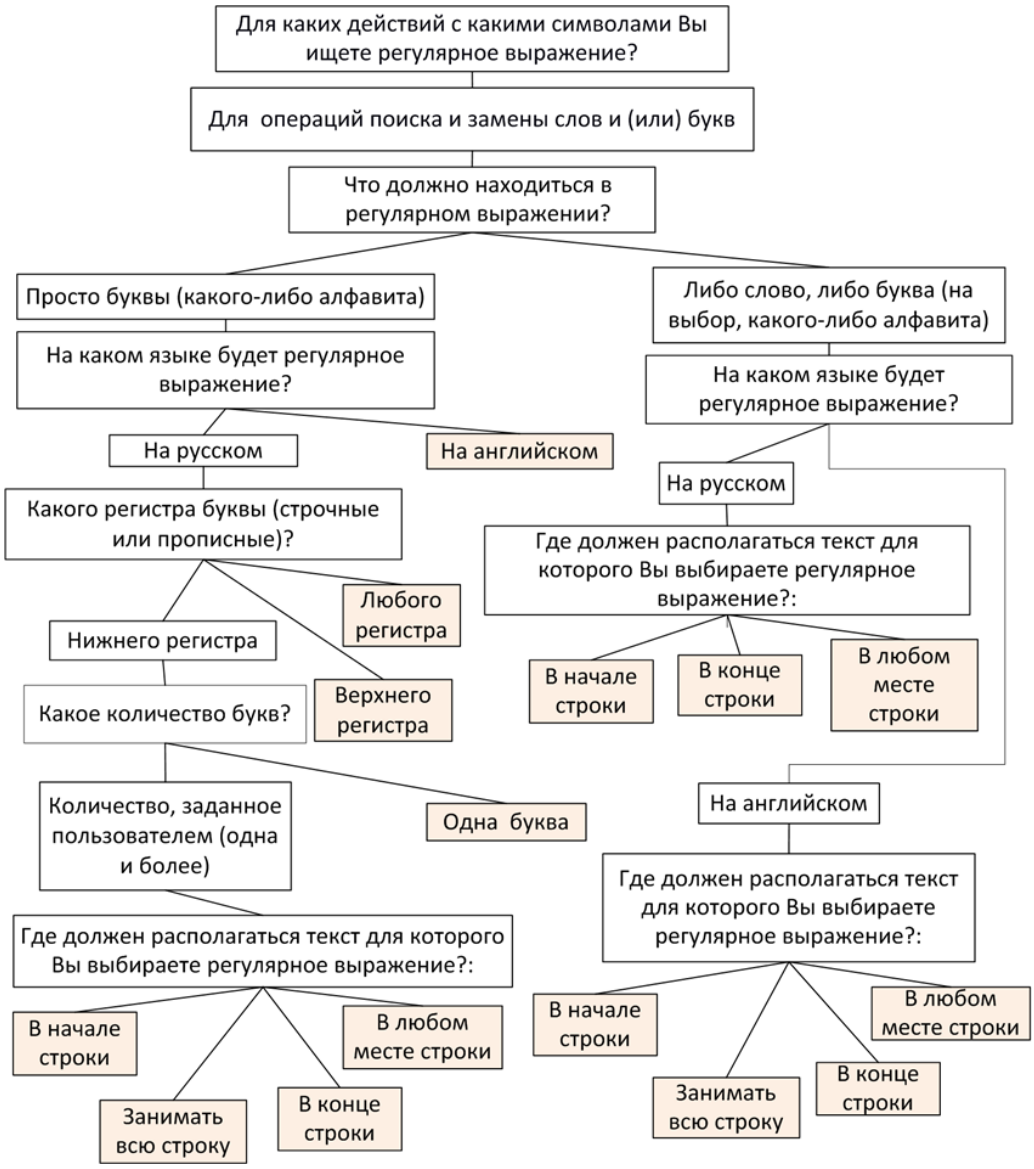

Рис. 2.2 Граф описания регулярных выражений

Метод описания с помощью специального языка

Наиболее близким методом к CDTL является метод описания с помощью специального языка (создан для снижения трудоёмкости генерации кода регулярных выражений). Он реализован с помощью слов русского языка и символов, которые описывают часть кода, того или иного регулярного выражения. За счёт этого снижается трудоёмкость формирования регулярного выражения и сокращается количество возможных ошибок в нём. Однако, главным его недостатком является то, что при сложном коде программы, нужно много слов, описывающих данный метод. Поэтому он всё ещё далёк от идеального. Идеальным, как было сказано выше, я считаю такой метод, когда

человек пишет текст описания на естественном языке, а программа преобразует его в язык программирования высокого уровня. Данный метод применим для снижения трудоёмкости генерации программного кода (например, регулярных выражений) и используется в узкоспециализированных областях. Однако он может быть некой частью глобального языка описания задачи.

Для его проверки была создана программа генерирующая регулярные выражения с помощью специальных слов на языке программирования Perl [9]. Она предназначена для того, чтобы дать возможность создавать регулярные выражения людям не владеющим этой технологией. Небольшой фрагмент кода данной программы схематично представлен на рисунке 2.3. для демонстрации принципов работы метода.

В нём показано как может происходить генерация регулярного выражения, в котором присутствует русская или английская буква. Сначала задаётся язык. Для этого нужно в поле ввода программы написать соответствующее слово (например, английский). Но информации всё равно будет мало, поэтому дальше необходимо добавить сведения о регистре. Допустим буква нужна верхнего регистра. Чтобы её получить надо дописать в поле ввода «верхнего регистра». И уже по этим данным генерируется регулярное выражение. Поэтому под квадратиками на рисунке 2.3 где указывается регистр программы написано слово «выражение». Часть кода блока (язык и регистр) выше написанного в скобочках слова «выражение», обязательна.

Как пример можно написать фразу «английский верхнего регистра» и по этой информации программа генерирует такое регулярное выражение [A-Z]. Дальше можно выбрать сколько раз повторяется эта буква. На данном рисунке показаны варианты 1 и более раз (от единицы) и заданное количество, хотя на самом деле их гораздо больше. Поскольку это демонстрационный фрагмент кода в нём не всё отображено. При этом к уже существующему выражению будет добавлен соответствующий код, как показано на рисунке. Помимо этого, можно задать положение буквы в строке кода.

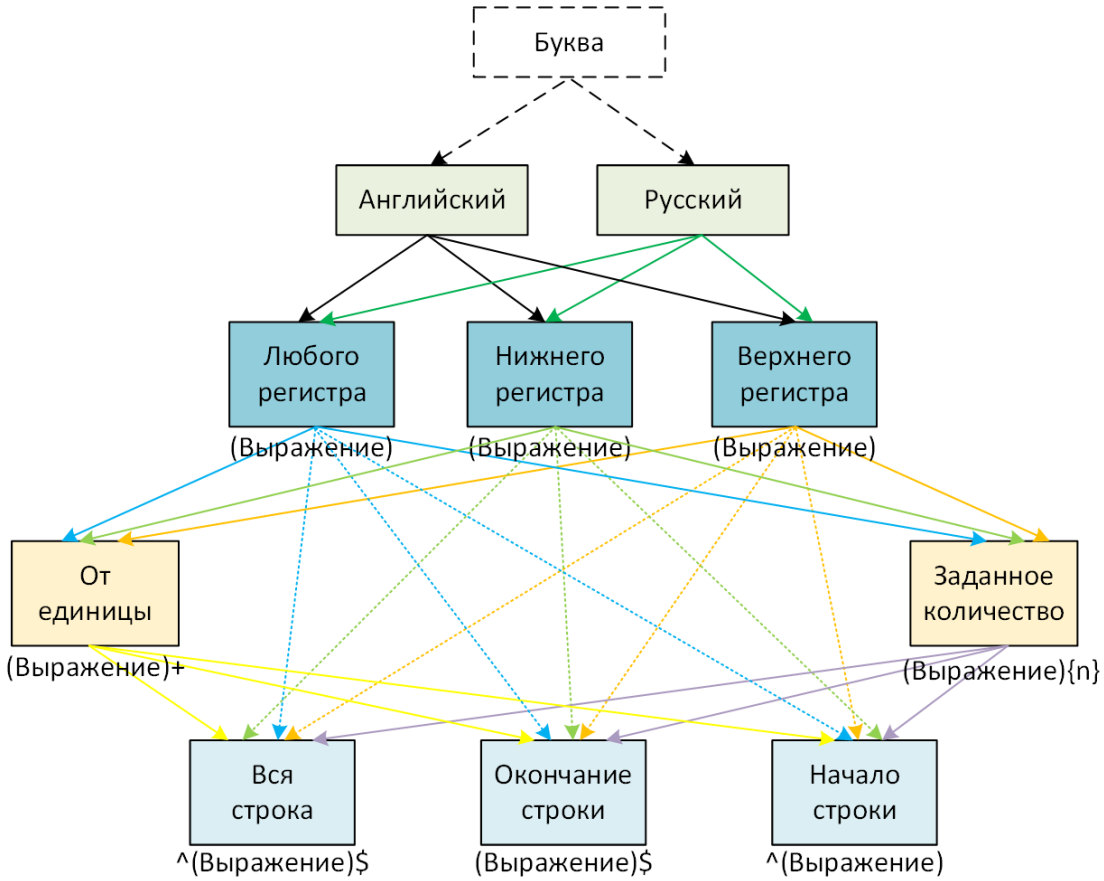

Рис. 2.3. Фрагмент кода для генерации регулярных выражений с буквами на русском или английском языке

Можно добавлять только указания количества букв, или только положение в строке кода, или всё вместе, или вообще ничего не добавить, оставив только язык и регистр. Квадратики на рисунке изображённые на одном уровне и закрашенные одним цветом взаимоисключающие. Как пример, если ввести в строку ввода программы фразу «английский верхнего регистра», вы получите такое регулярное выражение: [A-Z]. Если добавить к этой фразе «от единицы» регулярное выражение примет такой вид: [A-Z]+. Если добавить ещё фразу «вся строка», то получится уже такое регулярное выражение: ^[A-Z]+$. Стрелочками показано как можно дополнить регулярное выражение.

Как видно из описания выше, данная программа предоставляет определённую степень свободы пользователю. Теоретически с помощью неё можно сгенерировать почти любое регулярное выражение. К сожалению

некоторые виды регулярных выражений не удалось добавить в данную программу из-за того, что для их описания необходимо множество переменных. Из-за этого сильно усложняется создание кода программы и описания регулярного выражения. Например, по этой причине не были добавлены операторы проверки совпадений, подстановки текста и его замены.

Программы подобного типа должны работать по определённым правилам и иметь фиксированный набор используемых слов. Правила и слова, применяемые в данной программе необходимы при использовании этого метода. Для описанной выше программы они даны в специальной инструкции.

Опробованный в работе диалоговый метод программирования предполагает фиксированный набор вариантов программы и не является универсальным, но он позволяет подготовить достаточно обширный перечень программ и уменьшить число программных ошибок. А метод описания с помощью специального языка позволяет создавать программный код более свободно. Пользуясь им, человек может сгенерировать любой код, ограничиваясь лишь областью проблем, которую затрагивает программа, и заданными командами.

2.2 Формализация понятия метаязыка описания задач

Раньше задача программирования решалась комбинированием команд процессора. Поэтому для ещё большего сокращения человеческого участия было решено попробовать комбинировать более крупные программные модули. Многообразие таких модулей будет более чем на порядок-два больше множества команд процессора. Функция модулей может варьироваться посредством специальных вычисляемых модификаторов.

Такие модули должны размещаться в отдельных банках данных соответствующей специализации. Должна быть построена иерархия этих модулей.

Примерами таких модулей могут быть программы анализа текстовых файлов (поиск оговоренных образцов в log-файлах), преобразование массивов

данных в графические образы, модули статистической обработки данных и т.д. Такие модули должны стать в данном языке основными структурными элементами, из которых строится текст описания задачи и сама программа.

Все модули исполняемой программы должны быть написаны на одном и том же алгоритмическом языке.

Следует иметь в виду, что описание задачи на языке CDTL (Creating Description Task Language), может также содержать в себе ошибки, они могут быть и в самом трансляторе. Все это будет приводить к ошибкам в конечной программе. Но так как транслятор будет использоваться часто, можно ожидать, что ошибки в нём будут устранены относительно быстро. Ошибки же в описании задачи должны выявляться в процессе отладки и тестирования программы. Но так как текст описания должен быть относительно коротким, можно предположить, что ошибок там будет мало.

Текст описания задачи на языке CDTL сначала анализируется на предмет выявления области проблемы (граф диалога компьютер-программист). Далее предпринимается попытка разделить задачу на субзадачи. После этого делается попытка найти описание алгоритма, соответствующее каждой из субзадач. Для каждого из найденных алгоритмов определяются списки входных и выходных параметров. Для каждой из функций, если ее описание не удалось найти в банке описаний алгоритмов, создается свой программный модуль или пакет модулей [6].

Данная проблема может быть разделена на несколько уровней:

1. Создание банка алгоритмов, где ошибки минимизируются за счет многократного использования программы большим числом пользователей [94];
2. Для проблем, которые могут быть описаны древовидным графом, текст программы формируется из модулей, взятых из банка алгоритмов, а также модулей, написанных самим программистом (возможно помещение этих модулей в банк алгоритмов);

3. Сложные проблемы, характеризуемые графом с циклическими структурами, программа формируется из стандартных исполнительных модулей разного уровня.

Если модель языка описания задачи отличается от описанной выше, то:

а. Для описания задачи используется естественный язык;

б. Создается специальный язык, а транслятор переводит описание на один из алгоритмических языков высокого уровня, например, на C#;

В варианте “а” потребуется привлечение искусственного интеллекта, с неочевидными последствиями для частоты программных ошибок;

В варианте “б” либо нужен искусственный интеллект, либо алгоритмический язык нужно модифицировать и приблизить к модели, описанной выше.

2.3 Разработка синтаксиса метаязыка описания задач

При описании алгоритма каждой строке кода соответствует определенная команда или последовательность команд процессора. При описании задачи генерируемая программа может зависеть не только от текущего фрагмента текста, но также от предыдущего и, возможно, последующего текста. По этой причине синтаксис метаязыка описания задачи должен минимизировать такие возможности и сократить многозначность интерпретации его фрагментов.

Тем не менее описание задачи, даже выполненное строго по правилам, не гарантирует того, что существует решение данной проблемы. Возможно, что одному описанию будет соответствовать несколько программ реализации. Синтаксис метаязыка CDTL должен минимизировать такую вероятность.

Внутри описания задания может содержаться описание субзадачи. Начало описания задачи отмечается знаком <#, а завершение - #>. Начальный и завершающий разделители отделяются от описания задачи пробелом. Например, <# name="filter.pl"; .... #>. Имя описанной программы задается оператором name,

здесь "filter.pl" - имя будущей программы. Сокращенное CDTL-описание скрипта filter.pl в качестве примера предоставлено ниже.

<# name="filter.pl"; librs "IO::File, Fcntl, Socket, Time::Local, Time::localtime"; IN="access_log"; OUT = "all records from IN, not_containing elements from @list_of_exceptions"; #>;

Скрипт filter.pl читает журнальный файл Apache access_log и удаляет из него вспомогательные запросы, например, графических файлов, запросы со стороны поисковых серверов и т.д., что в разы ускоряет последующую работу программного пакета.

Язык CDTL состоит из операторов, атрибутов, объектов, параметров, описаний, переменных и массивов (табл. 2.1).

Таблица 2.1.

Примеры элементов метаязыка CDTL.

Операторы	**Атрибуты**	**Объекты**	**Параметры**	**Описания**	**Переменные**	**Массивы**
Develop; Create; Write; Read; Analyze;	Fast; double- precision; By polinomial,	Program; File; files DB; Table;	Label; parameters Accuracy; dt;	Что программа должна делать: Compute System_control;	Задаются IN= OUT= $=	Задаются @имя

Пример CDTL-описания при наличии субзадач:

<# name="**graf1**";
convert.file.name="/home/strigiforme/programs/pril1.txt".to.picture.name=
"/var/www/html/secur/html/images/pril1.png"; #> ## subtask 1

<# name="**graf2**"; convert.file.name=
"/home/strigiforme/programs/pril2.txt".to.picture.name=
"/var/www/html/secur/html/images/pril2.png"; #> ## subtask 2

<# name="**appro2**" approximate{ requests by time, requests with "////" or more
create.file.name=/var/www/html/secur/txt/pril1.txt;
create.file.name=/var/www/html/secur/txt/pril2.txt; graf1 graf2 #>

Операторы в метаязыке CDTL, как и в обычном языке программирования обозначают задание, которое необходимо выполнить [95]. Операторы могут быть простыми и композитными. Простые подразделяются на операторы описания и операторы действия. Операторы действия являются глаголами, указывающими на конкретные задачи. Операторы описания – это существительные, обозначающие объект, который надо добавить в программу. Например, оператор librs задаёт список библиотек, которые должны быть добавлены.

Композитные операторы создаются по следующей схеме: Оператор.объект.атрибут=значение;.

Пример: create.picture.name=var/www/mrtg/hostname_bytes-day.png; Возможны разные вариации, например, такая: оператора и атрибута нет, но есть подряд 2 объекта (например, chart.name=gnuplot1.png). Обязательно должна соблюдаться последовательность оператор=>объект=>атрибут. При этом объектов может быть более одного, а оператор или атрибут отсутствовать.

Объекты (второй ряд рис. 2.4) позволяют уточнить задание операции. Объект определяет то, над чем будет произведена та или иная операция, заданная оператором action.

Атрибуты предназначены для дальнейшего уточнения смысла**.** С помощью атрибутов описывают как нужно что-то сделать, например, с какой частотой проводить измерения, с какой точностью. Атрибуты в CDTL бывают 2-х видов: атрибуты объектов и атрибуты операторов (см. рис. 2.4). Отличаются они между собой тем, что первые уточняют смысл объекта, а вторые – оператора.

Параметры – это какие-либо значения переменных, функций и т.д., задающиеся для конкретного описания. Например, dt=300; (временной интервал в секундах).

Описания – поясняют то, что программа должна сделать. Например, System control (управление системой).

Переменные в CDTL бывают разных видов: входные, выходные, внутренние, системные. Входная (IN) и выходная переменная (OUT) могут встречаться в тексте только один раз. Остальные виды переменных могут встречаться сколько угодно раз.

Входная переменная - это строка описания входных данных. Она может содержать имена файлов, переменных, строк, массивов, имена баз данных и таблиц, терминальный ввод, а также URL в том числе и удаленных. Здесь могут присутствовать описания форматов и атрибуты (например, имя-пароль-сертификат). Источником входных данных могут быть отклики на запросы к поисковым системам, а также почтовые сообщения, SMS, голосовые и графические данные.

Выходная переменная - это строка описания выходных данных. Она может содержать имена файлов, включая графические и медийные, переменных, строк, массивов, URL, mail-адреса, телефонные номера для SMS, и атрибуты типа имя-пароль. Здесь могут фигурировать описания форматов и типов терминального оборудования, в том числе управляющих сигналов. Имеются в виду описания, предназначенные и анализируемые программой, а не человеком.

Внутренняя переменная используется в тексте описания CDTL. Задаются такие переменные, как и обычные переменные в языке Perl. Например, $x=5;

Системная переменная может быть целочисленной или символьной (строковой). Она может изменять функцию оператора, например, action, модифицировать значения переменных, изменять работу функций или подпрограмм. Изменение происходит простым присвоением, например, context=filter. Для каждого значения может быть несколько переменных, функций или подпрограмм с одинаковыми именами, но с разной функциональностью или значением. Например, когда значение переменной context=программа, допускается уточнение значения контекста [2,3] в виде программа[код|вычисление] или программа[обучения] или программа[обучения|университет] или программа[внедрения|сетевой график].

Массивы в CDTL представляются аналогами массивов в Perl и задаются аналогично. Например @core_f=("1","2","3");

Внутри описаний CDTL могут использоваться циклы, которые аналогичны тем, что применяются в языке Perl. Комментарий в CDTL начинается с помощью последовательности ## и продолжается до конца строки.

На рис. 2.4. показано, как создается словарь дополнений (имен) объектов, над которыми выполняются действия (операции). Например:
program, routine, file, array, list, web-cite, value, constant, date, string, symbol, DB, table, pointer, limit, approximate, convert(from/to),

Каждая операция из верхнего ряда рис. 2.4 может подразумевать бесконечное число вариантов описаний. Мощность множества операций (actions) $M_A=\infty$. Если выбирается какое-то конкретное значение action, выделяется некоторый достаточно большой (возможно бесконечный) объем в этом многомерном пространстве.

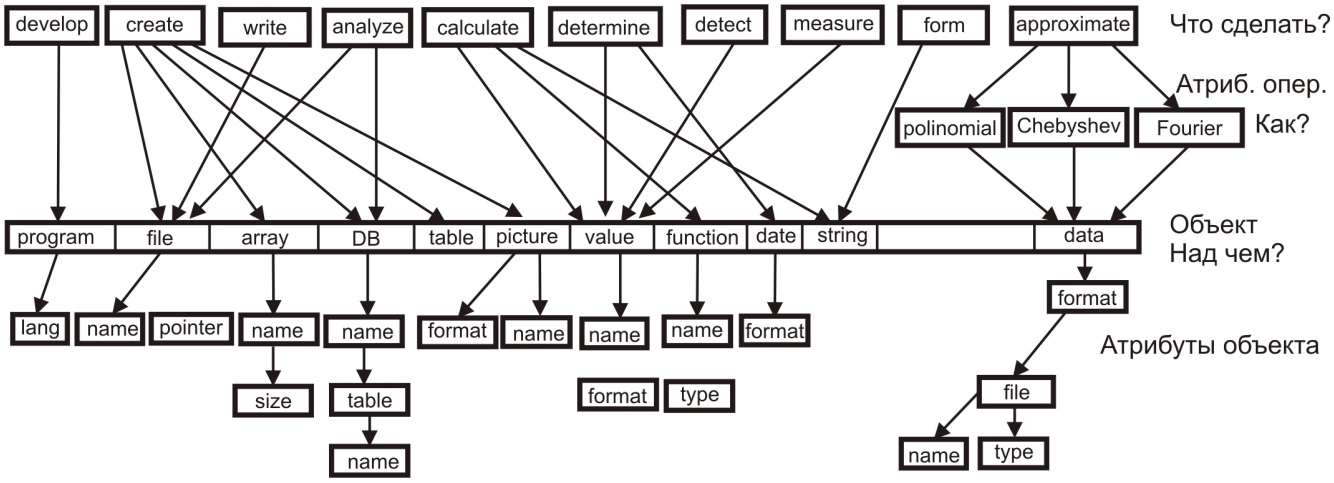

Рис. 2.4. Семантическое дерево операций

Объекты (второй ряд рис. 2.4) позволяют уточнить задание операции и сократить этот объем. Объект определяет то, над чем будет произведена та или иная операция, заданная action. Атрибуты объектов производят дальнейшее уточнение. Конечной целью создания описания на CDTL является сведение мощности оконечного множества к 1 ($M_A=1$, см. таблицу 2.2).

Таблица 2.2.

Примеры семантических связей

Название операции (action)	**Объект**	**Параметры объекта (obj_par)**	**Задание**	**Назначение оператора**
develop	**Program**	**name**	**Что программа должна делать:** Calculate; System control; Measure, ...	Разработать программу
create	**file**	**name**		Создать файл
	DB	**name**		Создать базу данных
	table	**name**		Создать таблицу DB
	array	**name**		Создать массив
	picture	**format**		Создать рисунок
write	**file array**			Произвести запись в файл, базу данных
read	**file record**			Прочитать строку файла или запись BD
analyze	**file array value**			Анализ текста, массива данных или результатов измерения с целью, например, оптимизации

2.4 Оценка сжатия текста, написанного программистом, при использовании CDTL

Для того, чтобы оценить насколько уменьшается описание, написанное человеком на языке CDTL, по сравнению с кодом на языке Perl был вычислен коэффициент сжатия. Он был рассчитан для более чем 10 описаний на языке CDTL и аналогичных программ на Perl. Данный коэффициент показывает во сколько раз сокращается текст при использовании CDTL. Была построена зависимость вероятности коэффициента сжатия текста при использовании CDTL (см. рис. 2.5).

Минимальный коэффициент сжатия при использовании CDTL в данной выборке равен 2,1. Максимальный коэффициент = 12. Для данных значений

была рассчитана дисперсия, которая оказалась равна 9,8. Я предполагаю, что величина сжатия будет сильно варьироваться в зависимости от задачи.

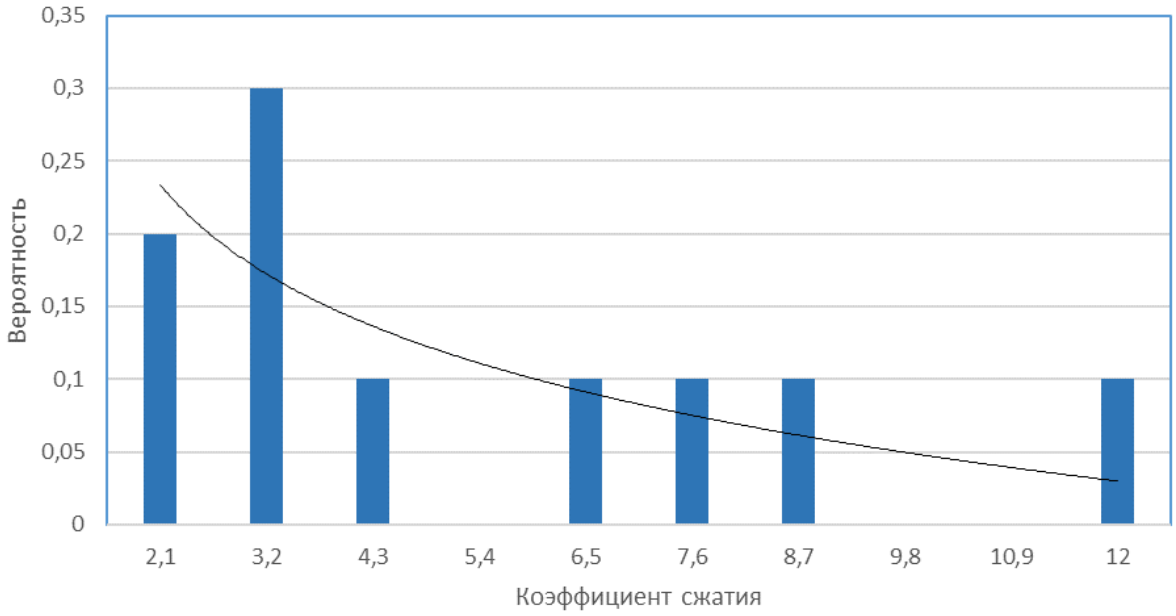

Рис. 2.5. Зависимость вероятности коэффициента сжатия текста при использовании CDTL

Математическое ожидание коэффициента сжатия текста описания CDTL по сравнению с языком PERL составляет 5,3±3,1. Приблизительно во столько раз наиболее вероятно, что текст описания задачи будет меньше кода программы на Perl при использовании метаязыка CDTL. Данный результат является довольно значительным и это показывает, что при использовании CDTL трудоёмкость написания программы существенно снижается. Помимо этого, коэффициент сжатия будет скорее всего увеличиваться при создании новых модификаций метаязыка CDTL. Среднеквадратичное отклонение коэффициента сжатия σ получилось довольно небольшим – 3,1.

Таким образом, разработан синтаксис метаязыка описания задачи, отличающийся ориентацией на создание специальных описаний для генерации программного кода динамически конструируемым транслятором, обеспечивающий снижение трудоёмкости генерации кода на языке высокого уровня.

2.5 Алгоритм динамического выбора состава модулей для решения коллектива задач одного типа

Все существующие программы вариативны. В одних обстоятельствах используется один фрагмент кода, в других – другой. Любой процессор имеет фиксированный набор команд и, комбинируя их, можно решить практически любую задачу. Для управления порядком команд и снижения трудоёмкости написания программного кода были созданы десятки алгоритмических языков разного уровня и назначения. Строка кода на алгоритмическом языке соответствует двум и более командам процессора. Это, кроме удобства программирования, понижает также и вероятность программных ошибок.

Для вариации программы используется вычисление определенных функций и проверка результата этого расчета. По результатам проверки принимается решение о выполнении того или иного фрагмента кода из числа имеющихся.

Алгоритм модифицируется, так чтобы отвечать требованиям списков входных и выходных параметров (см. рис. 2.6). Нужно помнить, что ошибка в программе модификации будет дублировать свои ошибки в каждое приложение! По этой причине требования к таким программам должны быть особо жесткими. Надеяться здесь можно только на то, что такие программы будут достаточно примитивны и ограничены по объему. Возможности модификации конкретного программного модуля не беспредельны. Варианты модификации предусматриваются в процессе создания такого модуля.

Далее часть кодов можно ввести внутрь стандартных модулей, сделав их более универсальными. Эти коды (сходные с макросами, применяемыми в некоторых языках) будут генерироваться с помощью программ, на основе параметров, заданных извне. Туда попадут части программ, адаптирующих данные для работы стандартных модулей. Это могут быть описания регулярных выражений. Программы подготовки данных, например, для работы Gnuplot и т.д. Для решения этих проблем создаются специализированные программные интерпретаторы.

Особое внимание нужно уделить выявлению программных функций, которые оказываются часто востребованными. Именно такие программы следует в первую очередь включить в библиотеку описаний алгоритмов.

Одним из путей снижения числа программных ошибок может стать создание большого числа библиотечных программ, а также обеспечение адаптации библиотечных программ для решения более широкого круга задач.

Требования к описаниям программных модулей для людей и программ принципиально различны. Такие описания для людей могут быть составлены на естественном языке в предположении, что клиент будет их просматривать последовательно. Любые описания и комментарии в программах должны выполняться на английском языке (чтобы язык описаний стал универсальным).

Для компьютера описание программы может представлять собой список ключевых слов (W_k), которые наилучшим образом характеризуют данный алгоритм или регулярное выражение, например, $W_k=(w_{1,k}, w_{2,k}, w_{3,k}, \ldots, w_{n,k})$, где k – номер описания алгоритма, n – число ключевых слов, описывающих данный конкретный алгоритм). Этот список должен составлять человек, желательно автор данной программы. Число таких ключевых слов для каждого из алгоритмов индивидуально. Задание делится на части, каждая из которых однородна и решает одну и только одну задачу.

Программа может формироваться на основе входных параметров (параметрическое управление), например, как в случае модификации кода для многовариантной программы, например, m_gnuplot рис. 2.6. Программы, созданные в результате диалога, могут быть также помещены в банк алгоритмов. Данный метод можно применять как инструмент взаимодействия программ и программных систем.

Описание процедуры может содержать строку:

<input parameter list> <action> <output parameter list>, где оператор <action> должен быть тщательно описан, так чтобы исключить неоднозначность интерпретации. Имени <action> может соответствовать запись в банке

алгоритмов. Должно существовать описание на русском и английском языках и список ключевых слов, которые могут использоваться при поиске.

В пределах одного типа <action> может быть много субтипов. Для типов и субтипов должны быть определены конкретные операции, например:

А. Вычисления, включая параллельные;

Б. Анализ текста и его преобразование;

В. Статистический анализ;

Г. Диагностика сети;

Д. Мониторинг операций реального времени;

Е. Работа с базами данных;

Ж. Работа со списками;

З. Работа с объектами Интернета вещей;

И. Компьютерная аналитика (когнитивный компьютинг);

К. Криптографические вычисления и т.д.;

Если проблема может быть решена программным модулем, хранящемся в банке алгоритмов, ищем и используем такой модуль.

Если нет, используем диалоговый язык программирования, где программа задает программисту последовательность вопросов, требующих ответов. Среди списка входных и выходных параметров могут быть: переменные (числовые и строковые), массивы переменных, списки, файлы, каталоги, базы данных и даже программные модули, извлекаемые из банка алгоритмов со своими списками входных параметров. Транслятор должен тщательно анализировать входные и выходные параметры и диагностировать допущенные программистом ошибки. Рекурсивных вложений параметров нужно избегать, так как это делает программу непрозрачной и увеличивает вероятность ошибок.

На рис. 2.6 показан пример программы, управляемой входными параметрами. Программа служит для формирования графических распределений с привлечением библиотечных функций gnuplot (библиотека Chart-Graph-3.2, которая содержит в себе несколько модулей). Разные модули Chart-Graph

предназначены для получения различных выходных графических форм и требуют разных форматов входных данных.

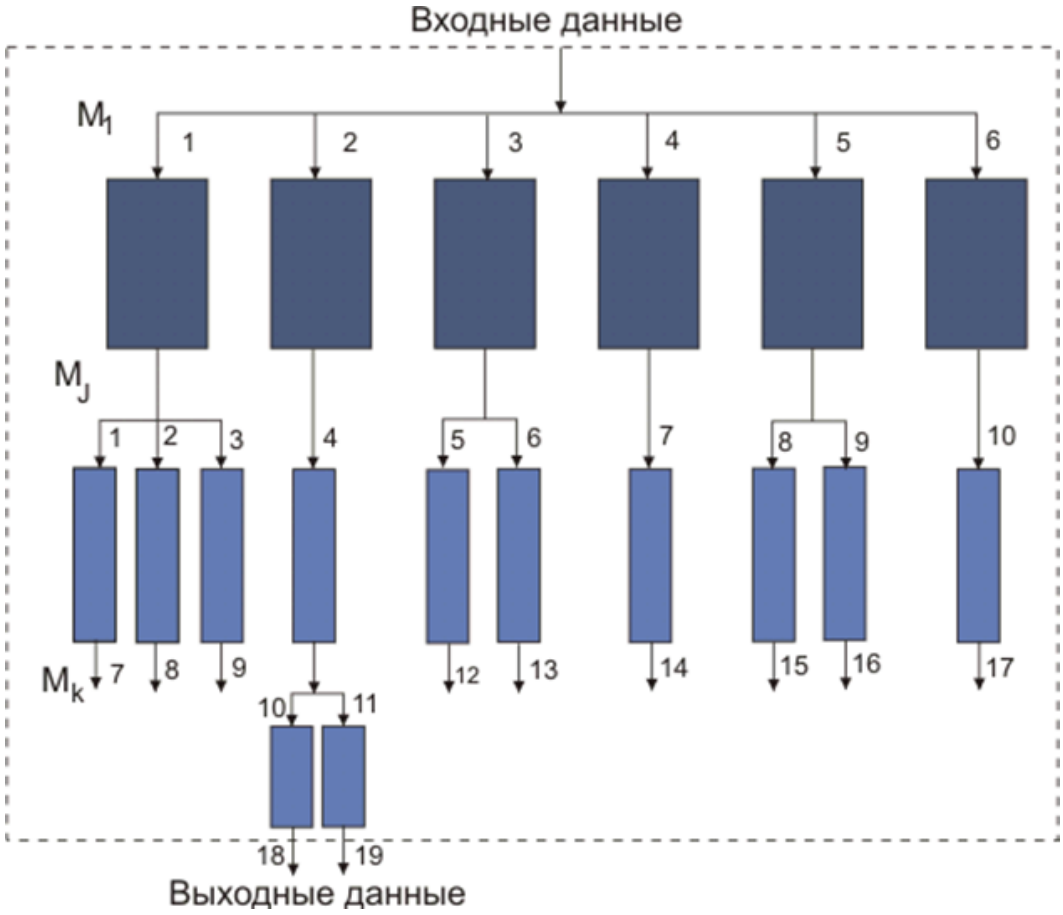

Рис. 2.6. Схема реализации программы типа m_gnuplot

Если какая-то функция в банке описаний отсутствует, то она может быть создана в тексте описания задачи на стандартном алгоритмическом языке, например, на Perl (он был выбран в качестве языка программирования для написания кода). Эта техника используется и в других языках, например, в HTML/XML, где делаются вставки текста на JavaScript или PHP.

В начале и в конце такого модуля ставятся метки-разделители, указывающие на алгоритмический язык вставки. Например:

<perl> (текст программы на языке Perl с учетом рабочей ОС) </perl>

На первом уровне M1 (рис. 2.6) на основе входных данных производится выбор варианта 1-6. На следующих уровнях может производиться уточнение варианта. Алгоритм динамического выбора состава модулей для автоматического выбора способа решения задачи в рамках коллектива задач

одного типа и сокращения количества модулей в базе данных представлен на рис. 2.7.

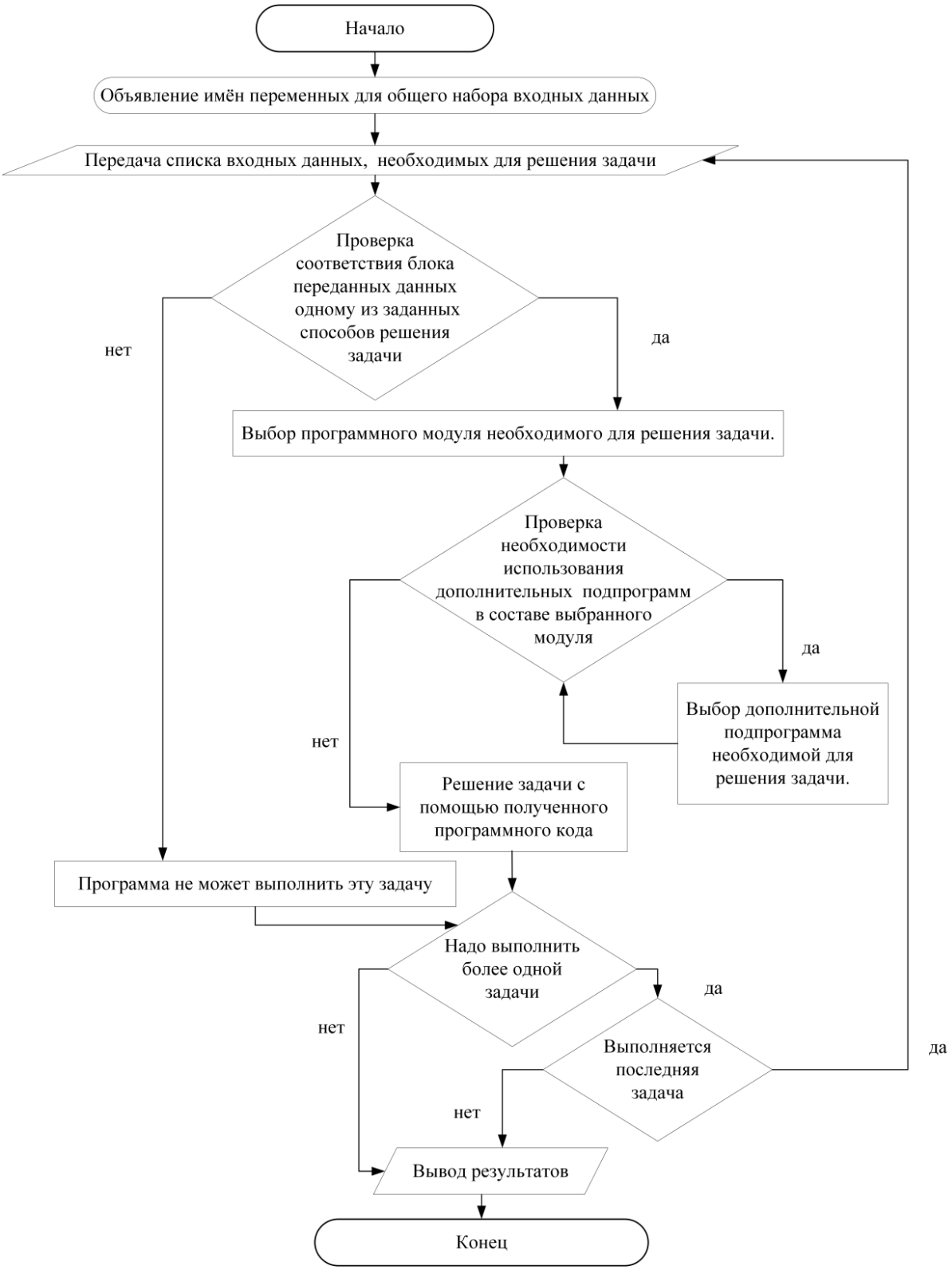

Рис. 2.7 Структурная схема алгоритма динамического выбора состава модулей для автоматического выбора способа решения задачи

Таким образом разработан алгоритм динамического выбора состава модулей для автоматического выбора способа решения задачи в рамках коллектива задач одного типа и сокращения количества модулей в базе данных.

2.6 Универсальный конфигуратор ПО, для пакета программ защиты Web-сервера

Программа «Универсальный конфигуратор ПО, для пакета программ защиты Web-сервера» состоит из 2-х частей: файл установки (скрипт «installator.pl») и программа для проверки текущей конфигурации (скрипт «konf_analiz.pl»). Данный скрипт пишет сведения о конфигурации в файл «linux.conf». Этот файл находится в той же папке, что и скрипт «installator.pl».

В скрипте «linux.conf» находятся следующие данные о компьютере: общее количество оперативной памяти, количество свободной оперативной памяти, общий размер раздела диска (или диска) на котором установлена операционная система, количество свободной и занятой памяти в этом разделе (или на этом диске), количество ядер и возможность измерения их температуры.

В скрипте «linux.conf» находится много данных о конфигурации операционной системы. Есть информация о том, какие из нужных каталогов для работы пакета программ для защиты Web-сервера созданы, а какие необходимо создать. Установлены ли на данном компьютере apache и mysql и если да, то какой версии. Ещё там сказано установлены ли пакеты mrtg и gnuplot. Дана информация о названии версии, номере версии Linux и разрядности. Выводится имя компьютера и его ip-адрес.

Если конфигурация компьютера позволяет измерить температуру ядер процессора, программа установки «installator.pl» задаст вопрос, установлен ли соответствующий модуль. И, если он установлен, запустит программу «mail.pl». Данный скрипт запросит ввод почтового адреса администратора и smtp. В программе есть функция проверки правильности ввода почтового адреса, если он введён неправильно, появится уведомление об ошибке. Если модуль для

измерения температуры ядер процессора будет установлен позже основной конфигурации компьютера, нужно запустить скрипт «mail.pl» самостоятельно через терминал соответствующей командой.

Ещё скрипт «konf_analiz.pl» проверяет установлены ли библиотеки, которые необходимы для корректной работы программ, написанных на языке Perl. Если все библиотеки в наличие, то он выводит информацию о них в виде списка, в обратном случае сообщает, что не все библиотеки установлены. Все библиотеки perl, кроме «Graf_dec», «Grafic» и «Gnu», из списка, указанного в файле «linux.conf», можно найти на сайте CPAN (всеобъемлющая сеть архивов perl)[96] в открытом доступе. Библиотеки «Graf_dec», «Grafic» и «Gnu» созданы сотрудниками ИТЭФ, поэтому их нет на данном сайте. Файлы данных библиотек с аналогичными названиями и расширением .pm, лежат в той же папке, что и файл «installator.pl». В процессе работы файла «installator.pl» они автоматически копируются в директорию «/usr/share/perl5». Если нужно добавить данные библиотеки в другую директорию, их можно скопировать или перенести вручную из папки, в которой лежит файл «installator.pl».

Сам конфигуратор включает в себя некоторый набор программ и файл запуска «installator.pl».

Такая система установки очень удобна потому, что если возникнут проблемы с какой-то частью ПО, можно запустить определённый файл (или файлы) для их устранения. Например, в случае если не хватает только библиотек perl, можно запустить файл «lib_install.pl». Все файлы необходимые для установки ПО лежат в одной папке.

Вот список файлов, которые запускает скрипт «installator.pl» и их описание.

- Файл «apache_ins.pl» предназначен для автоматической установки и запуска Apache, а также открытия 80-го порта.
- Файл «mysql_ins.pl» предназначен для автоматической установки MySQL.
- Файл «snmp.pl» предназначен для автоматической установки MRTG.

- Файл «lib_install.pl» предназначен для автоматической установки необходимых библиотек Perl.
- Файл «cgi.pl» предназначен для автоматической установки и настройки Perl CGI[97], lm-Sensors[98] (программа, необходимая для измерения температуры ядер процессора), gnuplot[99] (программа для построения графиков) и crontab.
- Файл «folder.pl» предназначен для автоматического создания нужных каталогов.
- Файл «mail.pl» предназначен для настройки почты, на которую будет отправляться сообщение в случае обнаружения слишком высокой температуры процессора.
- Файл «libwork.pl» предназначен для тестирования библиотек и запускается из скрипта «konf_analiz.pl», информация о библиотеках записывается в файл «linux.conf». Поэтому данный скрипт не входит в вышеуказанный список.

В случае его отдельного запуска, если появляется надпись «libraries_work», значит с работой библиотек всё в порядке.

В файле «installator.pl» есть функции проверки наличия программного обеспечения, которое с помощью него может быть установлено, на компьютере. Если необходимое ПО уже имеется, то программа не устанавливает его снова, что существенно сокращает время её работы. Данный механизм подробно изображён на рисунке 2.8, где представлена схема работы файла «installator.pl». Поскольку специализированные библиотеки perl и некоторые пакеты программ устанавливаются нечасто, проверка их наличия не производится.

Скрипт «konf_analiz.pl», больше по объёму чем «installator.pl». Но с помощью скрипта «konf_analiz.pl» производится только проверка наличия необходимых существующих программ, библиотек и физических параметров компьютера, а не установка. Поэтому на рисунке 2.8. была изображена именно схема работы скрипта «installator.pl». Данный скрипт имеет небольшой размер из-за множественного включения в него других скриптов, с помощью которых решаются более узкие задачи.

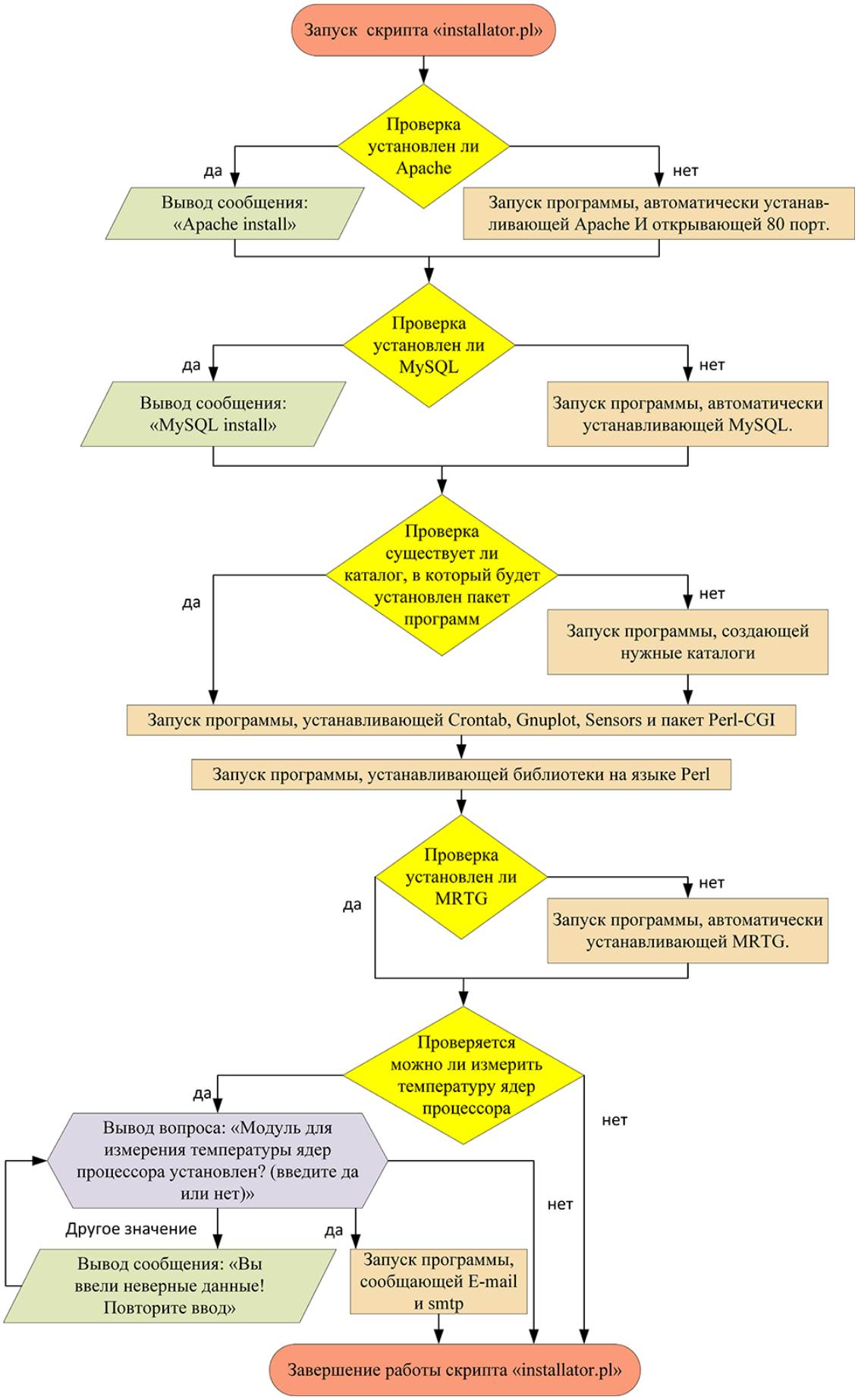

Рис. 2.8 Схема работы установочного скрипта «installator.pl»

Конфигуратор, описанный выше, автоматически устанавливает mrtg (инструмент для измерения трафика с течением времени). Для измерения трафика mrtg используются переменные MIB (Management Information Base [100]). По умолчанию в файле mrtg.cfg задан расчёт следующих переменных MIB:

ifInOctets (1.3.6.1.2.1.2.2.1.10) - полное число полученных байтов;

IfOutOctets (1.3.6.1.2.1.2.2.1.16) - число отправленных байтов;

tcpOutSegs (1.3.6.1.2.1.6.11) - полное число посланных сегментов, исключая повторно пересылаемые;

tcpRetransSegs (1.3.6.1.2.1.6.12) - полное число повторно пересланных сегментов.

2.7 Выводы по главе

1. Разработан синтаксис метаязыка описания задачи, который получил название CDTL (Creating Description Task Language). Он создан для сокращения числа ошибок в программах и уменьшения трудоёмкости разработки кода. Данный язык предназначен для использования с целью генерации программного кода из простых описаний, созданных человеком по специальным правилам. Он является некой надстройкой над алгоритмическими языками программирования высокого уровня, предназначенной для обеспечения снижения трудоёмкости разработки программного кода.
2. При использовании CDTL количество текста, написанного разработчиком, существенно сокращается. Что было доказано статистически. Математическое ожидание коэффициента сжатия текста описания CDTL по сравнению с языком PERL составляет 5,3±3,1. Этот результат является одним из доказательств того, что при использовании CDTL снизится трудоёмкость разработки программного кода, а, следовательно, станет доступной гораздо большему числу людей.
3. Разработан алгоритм динамического выбора состава модулей для решения коллектива задач одного типа. Он был создан для объединения блока схожих

программ в один модуль с целью уменьшения их численности и автоматизации выбора способа решения задачи. Данный алгоритм использует вычисление заданных функций и проверку результата этого расчета. По результатам проверки принимается решение о выполнении того или иного фрагмента кода из числа имеющихся. Алгоритм модифицируется так, чтобы отвечать требованиям списков входных и выходных параметров.

4. Разработаны специальные технологии снижения числа программных ошибок в процессе создания кода и универсальный конфигуратор ПО. Данный конфигуратор разработан для автоматического создания необходимой программной среды для работы транслятора с языка CDTL на язык программирования высокого уровня.

3. Разработка программной среды и оптимизация метаязыка описания задачи

3.1 Выбор и обоснование архитектуры программной среды

Web-сайты представляют собой наиболее привлекательный ресурс для хакеров. Для обеспечения безопасности WEB-сервера saturn.itep.ru [101] был создан набор скриптов, которые запускаются по crontab [102] и анализируют содержимое файлов access_log [103], error_log [103] и secure. При этом выявляются атаки типа подбора пароля, pass-traversal, переполнения буфера, IP-адреса, откуда ищутся файлы с известными уязвимостями, попытки использования запросов с нелегальными методами протокола HTTP (put, delete и др), источники DoS-атак. Если число атак определенного типа за оговоренное время превышает некоторый порог, автоматически самим компьютером блокируется доступ с соответствующих IP-адресов.

Сервер Saturn.itep.ru подвергся нападению в конце ноября 2017 года. Атака длилась с вечера пятницы до утра понедельника. В атаке принимали участие более 50 компьютеров. Проводился подбор пароля, искались уязвимые файлы, предпринимались попытки атак path-traversal и т.д. Поток входных данных был настолько большим, что невозможно было подключиться к серверу удаленно. Было перебрано более 2 миллионов вариантов паролей. Но самое важное, для 15 машин из числа участниц атаки был заблокирован доступ к WEB-серверу самим компьютером.

Было решено попробовать применить разработки, предназначенные для снижения числа программных ошибок модулей ПО для защиты WEB-сервера.

Подбор и модификация модулей данного ПО должны осуществляться посредством диалога. Программа сама предлагает разработчику варианты ответов на заданный ей вопрос, из которых он должен выбрать верный и кликнуть на него мышкой. В результате выполнения программистом данных действий будут выбраны и сконфигурированы нужные модули для защиты Web-сервера. Более подробно о диалоговом методе было написано в главе 2.

Для того, чтобы настроить необходимое ПО для работы транслятора пользователь может воспользоваться специальным конфигуратором, устанавливающим его автоматически. В рамках данной работы был создан «Универсальный конфигуратор ПО, для пакета программ защиты Web-сервера», который предназначен для операционной системы Scientific Linux[104] версии 6.10. На данную программу было получено авторское свидетельство [15]. Главной её функцией является настройка операционной системы для использования пакета программ защиты Web-сервера. С помощью этой программы можно проверить физическую конфигурацию компьютера, наличие необходимого ПО и библиотек. Особенность данной программы заключается в том, что она устанавливает необходимое ПО, в том числе и библиотеки, с которыми работает транслятор, автоматически. Минимальное участие пользователя заключается только в том, что он вводит свой адрес почты и smtp, в случае если ему надо настроить модуль proc.pl, и запускает нужные ему скрипты, воспользовавшись специальной инструкцией. Подробнее об этом конфигураторе написано в следующем разделе данной главы.

3.2 Банки описаний алгоритмов программных модулей

При разработке пакета программ для защиты Web-сервера был создан банк алгоритмов. В него были загружены программные модули на языке Perl (для этого использованы программы, созданные работниками ИТЭФ для целей сетевой диагностики и безопасности).

Структура банка алгоритмов представлена на рис. 3.1

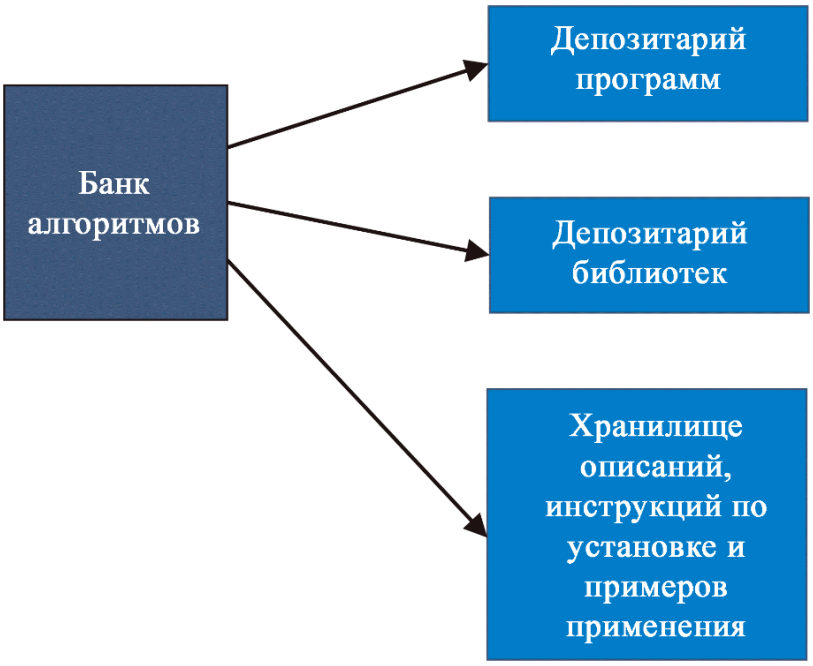

Рис. 3.1. Структура банка алгоритмов

В банке алгоритмов находятся модули, меняя входные параметры, которых, можно варьировать их работу. Поэтому увеличится масштаб их использования и уменьшиться количество (см. рис. 3.2).

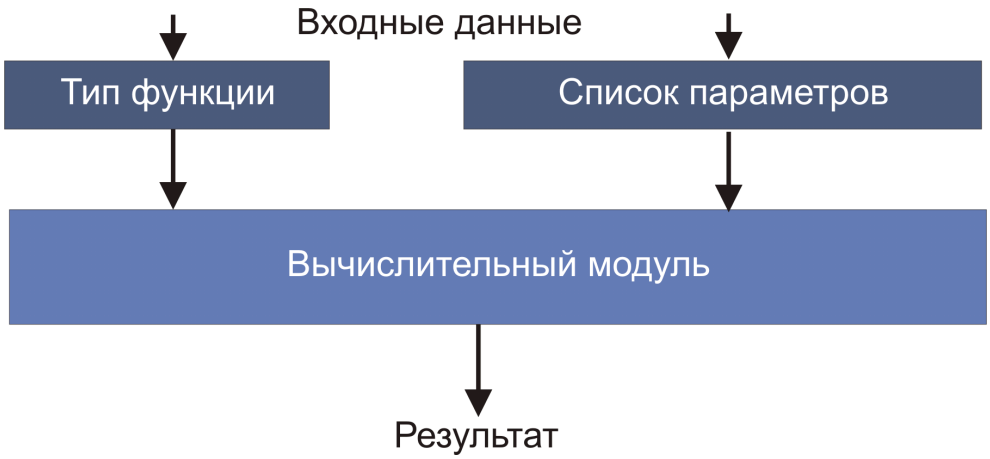

Рис. 3.2. Структура модуля со входными параметрами.

В блоке “Список параметров” может производиться подготовка (преобразование) данных.

Помимо этого, входные параметры можно задавать на языке описания задачи. Список и формат выходных параметров определяется описанием, которое находится внутри банка данных алгоритмов или же его задаёт программист.

Подобный способ использования программных модулей был мной опробован на примере некоторых программ, на которые были получены авторские свидетельства.

Перечень таких входных параметров может быть задан диалоговым методом в процессе создания описания задачи (число параметров, конкретные значения, типы параметров, возможные диапазоны значений и т.д.). Число параметров и диапазоны значений нужны для нахождения ошибок, как при трансляции, так и при исполнении кода.

Банк алгоритмов находится в свободном доступе для того, чтобы ошибки, которые находятся внутри программных модулей могли устраняться с помощью методики «Множества глаз», описанной ранее. В результате количество ошибок будет снижаться за счёт выявления и устранения их во время использования программы.

Но даже такой подход не даст гарантий отсутствия программных ошибок, потому что в принципе устранить до конца их невозможно. Люди склонны ошибаться, в том числе при написании программного кода. Поэтому, даже с учётом применения методики, описанной в данной работе, ошибки могут быть сделаны в коде самих модулей и в тексте описания задачи. Но описание задачи всегда будет во много раз короче, чем программный код, который нужен для её решения. А это сокращает вероятность появления ошибки в коде, а, следовательно, и их количество.

Перед помещением описания алгоритма в банк надо узнать является ли эта программа достаточно изолированной и универсальной. Если это так, то можно создавать описание, соответствующее ей, и заносить его в банк. В ином случае, программа разбивается на части, каждая из которых будет отдельным алгоритмом, помещаемым в банк.

3.3 Применение банка алгоритмов как части транслятора описания задачи на CDTL

Причины выбора языка «Perl»:

- Используемый язык программирования высокого уровня (какими являются, в том числе, Perl и Python) должен быть интерпретируемым, так как только там относительно легко найти и устранить ошибку. Именно по этой причине растет популярность программ с открытыми кодами.
- В языках типа C или C++ библиотечные модули имеют объектный формат и находить и исправлять ошибки постороннему человеку там практически невозможно.
- Perl самый старый язык для работы с текстами, следовательно, на нём написано большое количество библиотек. Perl создан в 1987 году, а Python в 1991.
- В Perl низкое количество ошибок на 1000 строк кода[105].
- Язык Perl имеет встроенную поддержку регулярных выражений. Многие приложения и библиотеки реализации регулярных выражений, которые, как утверждается, поддерживают регулярные выражения языка Perl или совместимы с ними, на самом деле просто используют регулярные выражения в стиле языка Perl. Они поддерживают синтаксис, похожий на синтаксис Perl, но не поддерживают тот же набор особенностей регулярных выражений. (Напр. PHP, .Net, Java, JavaScript, Python, Ruby)
- В Perl присутствует хранилище модулей CPAN. Комплексная архивная сеть Perl (CPAN) в настоящее время насчитывает 197 056 модулей Perl в 42 465 дистрибутивах, написанных 14 109 авторами, зеркально отраженных на 1 сервере. Архив находится в сети с октября 1995 года и постоянно растет.
- Perl изначально ориентирован на работу с текстовыми документами, а описание задачи – текстовый документ. Есть другие современные языки, ориентированные на работу с текстом (например, Python), но в них нет других

преимуществ, которые есть в Perl. Поэтому он больше всего подошёл для нашей цели.

При создании транслятора описания задачи на CDTL в код языка Perl был создан соответствующий банк алгоритмов и занесены в него необходимые модули. Для хранения информации о программных модулях была использована таблица базы данных MySQL. Для занесения описаний в эту таблицу, использовалась специальная программа “Loader v 1.0” [14]. С помощью неё решается проблема загрузки описаний в базу данных. Она спроектирована для людей не знакомых с MySQL. Интерфейс данной программы представляет собой HTML-форму, в которую заносится информация для ввода в таблицу. Выгрузка занесённых данных в таблицу происходит с помощью специального скрипта на языке Perl. Контрольная сумма модулей генерируется программным методом. Пользователь указывает лишь путь к файлу в поле «программный модуль» нажав на кнопку «обзор». Это сделано для того, чтобы пользователь не ошибался вводя ссылку на файл. При занесении записи в базу данных программа проверяет совпадение имени заносимого модуля с уже имеющимися. Это необходимо для того, чтобы записи не дублировались. В программе “Loader v 1.0” существует проверка правильности ввода почты автора, года создания модуля и т. д. Большинство полей сделаны обязательными для ввода и помечены звёздочкой, чтобы пользователь не забывал занести в них данные. Если данные не были введены, но пользователь уже пытался передать их в базу, передачи данных не происходит и появляется красная надпись о том, что поле обязательно для заполнения.

Таблица базы данных MySQL,в которую заносятся описания, состоит из 23 полей с кодировкой utf-8. 3 поля числовые, остальные строковые.

Структура таблицы базы данных алгоритмов, которая используется транслятором описания задачи на CDTL в код языка Perl

Запись базы данных алгоритмов должна содержать:

1. Имя программы.
2. Цель (назначение модуля).
3. Что реализует (алгоритм работы модуля).
4. Перечень входных параметров программы и их характеристики.
5. Перечень выходных объектов (вычисленные значения, файлы, списки, массивы и пр.) и их характеристики.
6. Перечень модификаторов программы и их функции. Модификатор – это некая переменная (или их набор), с помощью которой модифицируется работа алгоритма данного модуля.

Таблица базы данных алгоритмов состоит из следующих полей:

1. Название программного модуля.
2. Размер файла.
3. Число строк кода.
4. Тип файла.
5. Перечень модификаторов.
6. Когда и как запускается.
7. Контрольная сумма программного модуля.
8. Язык программирования.
9. Необходимые библиотеки.
10. Код проблематики, к которой относится данный модуль.
11. Фамилия, имя и место работы создателя (телефон опционно).
12. Email автора.
13. Год создания (4 цифры).
14. Дата последней коррекции.
15. Ссылка на программный модуль.
16. Входные параметры.
17. Выходные параметры.
18. Описание алгоритма.

19.Назначение (описание для пользователя).

20.Производительность программы.

21.Дополнительная информация.

22.ML-сигнатура.

23.Ключевые слова.

Желательно, чтобы модули в банке имели минимальное число ошибок. Для этой цели мною был применен метод «Множества глаз». Для реализации данного метода, нужно, чтобы пользовали могли применять код программных модулей для своей цели и находить в нём ошибки. А также сообщать о них разработчикам.

3.4 Алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных

Метод определения контекста слов был применён в диссертационной работе для анализа отзывов внешних пользователей на модули из банка алгоритмов [18]. Данный анализ необходим для выявления и устранения ошибок в программах, находящихся в банке алгоритмов. Сейчас этот банк находится в открытом доступе.

Так как отзывы написаны на естественном языке, нужна автоматизированная система его анализа. При таком анализе часто бывает нужно определять контекстные значения отдельных слов и фрагментов текста. Смысл текста отзывов на модули банка алгоритмов программой распознаётся с помощью ключевых слов. А для определения контекста ключевых слов использован алгоритм, описанный в данной диссертационной работе.

По контексту слов создана база данных на MySQL “context_db”, куда занесены слова, которые указывают на контекст отзыва. Для анализа контекста с помощью базы данных была написана соответствующая программа [17].

С помощью метода, описанного в данной работе, можно определять контекст различных слов по тематике Web-безопасности (программа, сеть,

протокол, канал, порт и др.). Для контекстного анализа этих слов уже была создана специальная база данных и использована программа [17].

Проблема распознавания контекста слов компьютером, в том числе в описании задачи, весьма актуальна. Она важна для поисковых систем, машинного перевода, интерпретации текста при грамматическом разборе и в машинном анализе содержания документов.

Главными недостатками существующих методов является сложность их применения, а также то, что они требуют часто больших вычислительных ресурсов [38,64,72,106-111]. Поэтому одной из целей данной работы является создание простого метода машинного определения контекстного значения отдельных слов, частей текста и текстовых файлов.

Данный метод заключается в том, что контекстное значение слова зависит от расстояния L между этим словом и другими словами, задающими контекст. Расстояние между словами определяется числом слов N, размещенных между ними (L=N+1). Предполагалось, что контекст конкретного слова можно определить по положению некоторых семантически связанных с ним слов, содержащихся в тексте.

Корневое слово W1 может иметь два или более значений, зависящих от контекста и определяемых словами W2. Слова W2 могут и отсутствовать в тексте документа. Контекстное значение слова W1 в этом случае может определяться семантически связанными с ним словами W3. Варианты семантических сетей показаны на рис. 3.3. Вариант **А** предполагает наличие в тексте документа корневого слова W1, которое может иметь разные контекстные значения, определяемые словами W2. Некоторые слова-значения W2 (например, $W2_2$) могут в документе отсутствовать (рис. 3.3В).

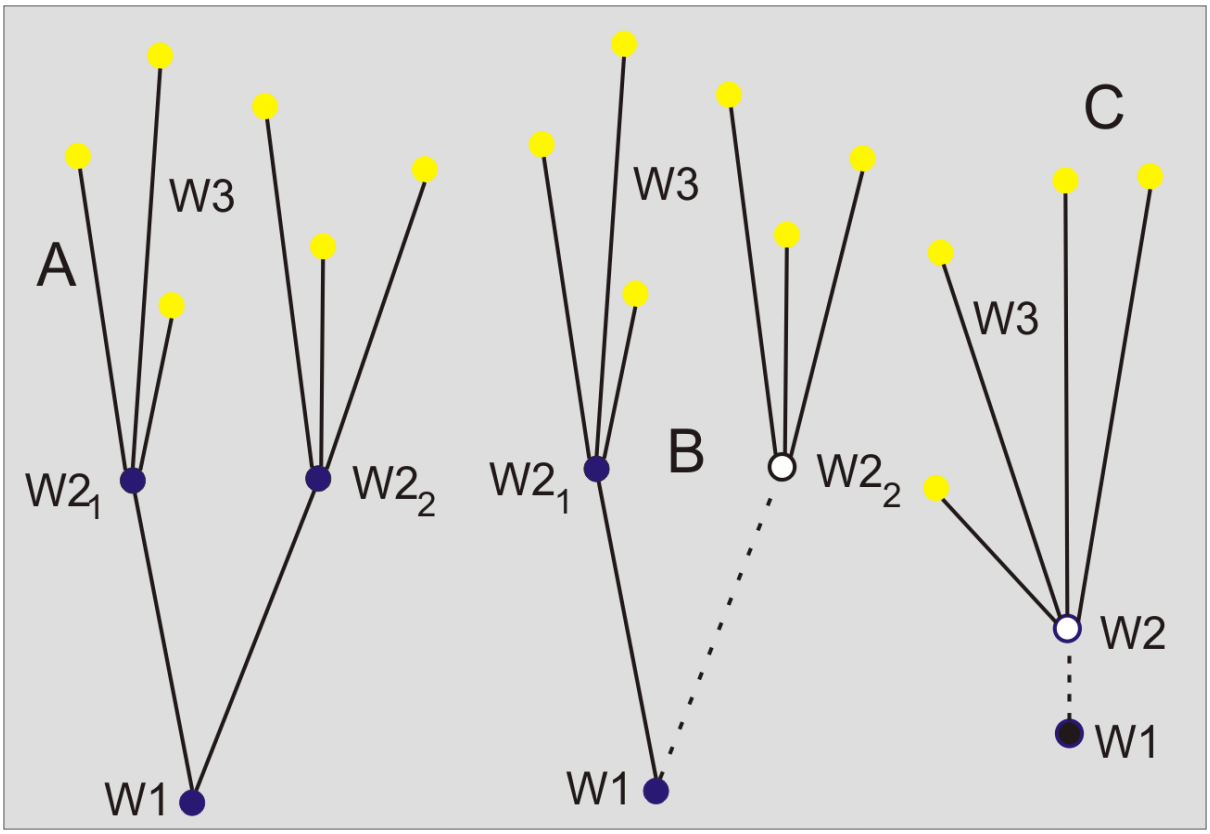

Рис. 3.3. Варианты семантических связей в тексте

Предполагается, что каждому из слов W2 соответствует некоторое число слов W3 (слова-характеристики), именно они и определяют выбор контекстного значения слова W1. Секция рис. 3.3С иллюстрирует вариант оценки контекста документа в отсутствии слова W2.

Рассмотрим это на примере разделения контекстных значений слова "программа": *компьютер* и *обучение*. W1= программа; $W2_1$ = компьютер; $W2_2$= обучение. Если имеется в виду компьютерная программа, то в тексте могут встретиться слова*: подпрограмма, цикл, файл, библиотека, прерывание, память, код, трансляция; метка; исполнение; исключение; наследование; скрипт; накопитель; синтаксис; присвоение; комментарий; итерация* и т.д. Эти слова в таблицу не были включены из-за экономии места. Если имеется в виду программа обучения, в тексте могут встретиться слова: *учитель, лектор, студент, тестирование, ЕГЭ, зачет, экзамен и т.д.* Эти слова также не были включены в таблицу из-за экономии места. (см. таблицу 3.1). Таблица должна быть создана заранее и никак не зависит от исследуемого текста.

Таблица 3.1.

Фрагмент таблицы корневых слов (W1), слов-значений (W2) и слов-характеристик (W3)

Корневое слово W1	**Слова-значения W2**	**Слова-характеристики (W3)**	**Метрика [M]**
Программа	компьютер	программирование	**70**
		отладка	**60**
		тестирование	**40**
		подпрограмма/subroutine	**30**
		объект	**15**
		файл	**26**
		прерывание	**40**
		Оперативная память	**70**
		переменная	**30**
		константа	**20**
		SSD	**30**
		массив/array	**50**
		библиотека (программ)	**15**
		язык (программирования - название)	**60**
	обучение	пособие	**45**
		преподаватель	**50**
		учащийся	**95**
		учебник	**90**
		дистанционное	**70**

Следует учитывать, что слова могут встретиться в разных падежах, числах и пр.

В таблицу заносятся только слова, имеющие два или более контекстных значений (W2). Полная таблица даже для отдельной области знаний может быть в сотни раз больше. Содержимое таблицы должно храниться в банке данных, что облегчит доступ к хранящимся в ней словам.

В первой колонке таблицы размешаются слова, которые могут иметь несколько контекстных значений (корневые слова - W1) и могут также определять контекст документа в целом. Во второй колонке (W2) помещаются слова, которые обозначают возможные контекстные значения слов из первой колонки. В третьей колонке (W3) записаны слова, конкретизирующие значения

слов из второй и первой колонки. Слова из этих трех колонок образуют древовидный граф. Значения метрики M относятся к словам из третьей колонки таблицы.

На рис. 3.4 изображён пример фрагмента семантической сети в виде графа, иллюстрирующего таблицу 3.1. На нём отображены слова из первой (W1), второй (W2) и третьей (W3) колонок данной таблицы.

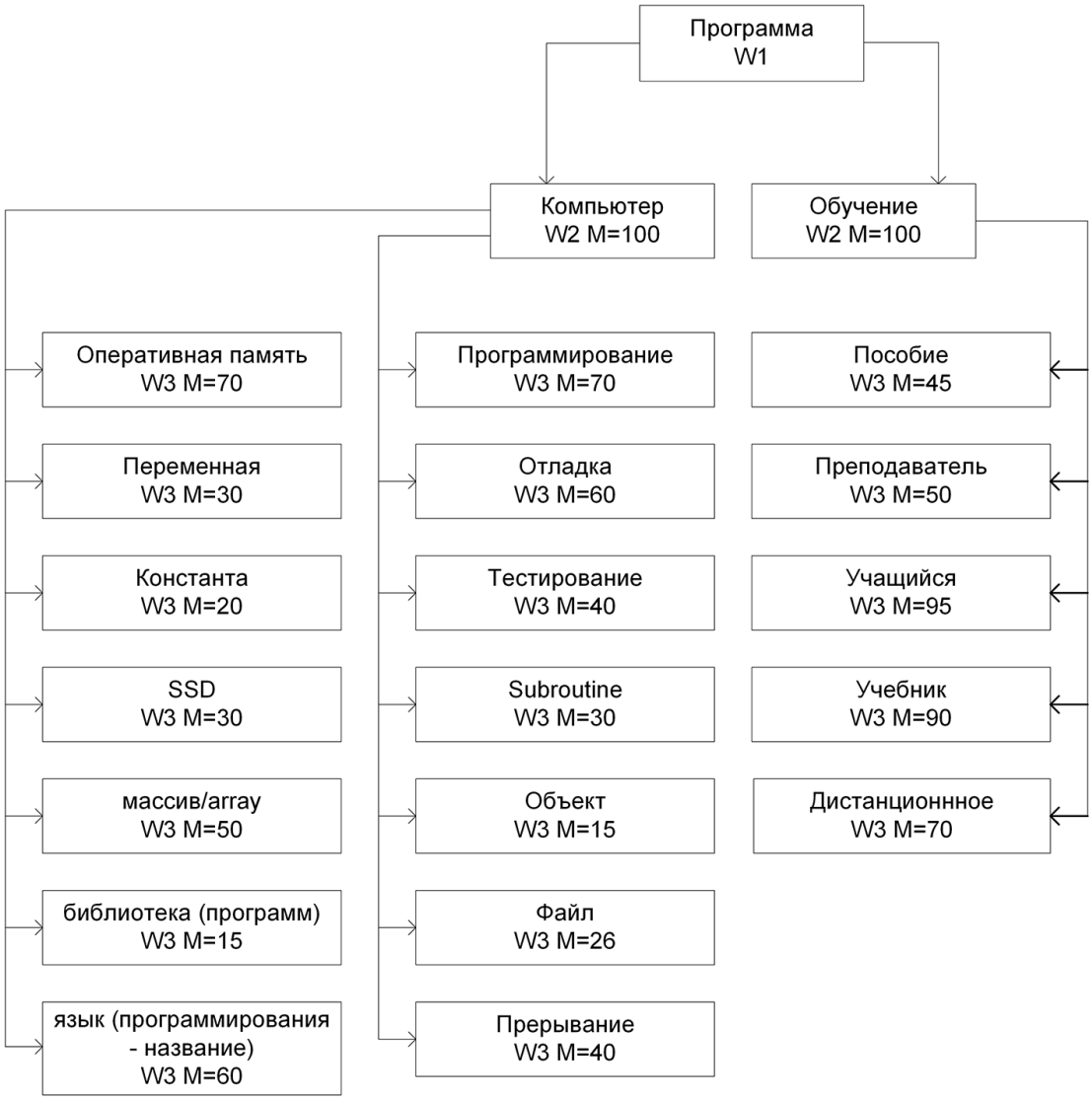

Рис. 3.4. Пример фрагмента семантической сети в виде графа, иллюстрирующего таблицу 3.1

В самом его верху находится корневое слово W1 «программа». От него идут ветвления к словам W2, которые обозначают контекстный смысл слова W1 "программа". Метрика характеризует близость слова к контекстному значению, с которым оно связано и обозначается буквой M. Слова во второй колонке всегда

имеют метрику равную 100. Поэтому для слов «компьютер» и «обучение» M=100. Далее от них идут ветвления графа к словам из третьей колонки W3. Это все прочие слова в таблице. Под каждым из этих слов указана его метрика (<100) и они сильно влияют на контекст, определяемый для слова W1.

На рис. 3.4 отображен фрагмент схемы, поскольку контекстных смыслов у слова «программа» может быть много. Существует также много слов-характеристик, помимо указанных в колонке W3 таблицы 3.1. Графы такого вида и являются основой семантической сети, используемой для оценки контекстного значения слов и документов.

Значения метрик могут настраиваться с помощью контрольных текстов на стадии отладки системы. Слово в первой колонке является корнем дерева семантических связей. Любое из слов первой колонки (W1), второй -W2 и третьей - (W3) может встретиться в документе больше одного раза. Слово из колонки W1 должно присутствовать в документе обязательно, в противном случае не возникает задачи определения его контекстного значения. Слову из второй колонки, если оно встретилось в документе, присваивается метрика M=100. Но это должно учитываться лишь при определении контекстного значения всего документа. Слово из второй колонки, определяющее контекстное значение слова из первой колонки, может и не встречаться в документе вовсе.

При отсутствии в тексте слова из второй колонки, но при наличии слов из третьей колонки, сопряженных с ним семантически, можно однозначно определить контекстное значение слова из первой колонки (W1).

Можно предположить, что чем ближе слово-характеристика к слову из вышестоящей вершины графа, тем с большей вероятностью оно определяет контекст этого слова. Наличие слова из третьей колонки, размещенного в тексте ближе к слову из второй колонки, должно влиять на выбор контекстного значения слова сильнее, чем в случае слов, размещенных дальше. Одним из возможных методов оценки контекстного значения слова может быть формула 3.1.

После того как положение слов W1, W2 и W3 определено, производится вычисление суммы **C.**

$$\mathbf{C_{k,n}} = \sum_{i=1}^{m} (M_i \times f(L_i)); \quad (3.1)$$

где C – мера, определяющая контекстное значение слова W1, L - расстояние между словом, например, "компьютер" и "отладка" (см. таблицу 3.1), M_i - метрика слова-характеристики W3 (M=1÷100), m - число семантически связанных слов W3 (см. таблицу 3.1), $f(L_i)$ - весовая функция от $\mathbf{L_i}$, i - номер встретившегося слова из колонки 3. В простейшем случае $f(L_i) = 1/L_i$, а для небольших документов $f(L_i) = 1$. **L** определяется числом слов **N** размещенных между словом **W2** и одним из слов **W3 (L=N+1).** Весовая функция $f(L_i)$ нужна для ослабления влияния удаленных слов на оценку контекстного значения слова W1**.** Если в тексте присутствует две или более копий слова W2, формула 3.1 может быть модифицирована.

Для больших документов контекст каждого конкретного слова **W1** может оказаться неодинаковым для разных областей документа. Размер области может быть настраиваемым, с дискретом в одну страницу (~400 слов). При этом можно варьировать начало и размер области и отслеживать вариации значений C и контекстного значения конкретного слова W1.

Индекс k для **C** определяет, к какому из возможных значений W2 относится данная мера (k=1,..n). см. вторую колонку таблицы 3.1. **n** – число возможных значений слова W1 (чаще всего n=2÷3). Значение слова W2 с большим значением **C** в контекстном смысле считается предпочтительным.

Значения M_i выбираются при настройке с использованием тестовых документов.

Для анализа контекстного значения было использовано специализированное ПО (авторская программа на языке Perl) [11] Её схема изображена на рисунке 3.5.

Рис. 3.5 Схема работы программы “Расчёт контекстного значения для заданного слова в текстовом файле”

На рис. 3.5 приведена предпочтительная реализация работы программы, которая может быть создана с использованием предложенных способов.

1. Производится запуск программы;
2. Программа считывает содержимое текстового файла и заносит его в массив;
3. Массив разбивается на слова. Каждое слово текста – отдельный член массива;
4. Программа ищет нужное слово W1, если оно не найдено - прекращает свою работу;
5. Если хоть одно слово W1 найдено в тексте, программа ищет слова из колонок W2 и W3, во всех падежах и числах, чтобы проводить с ними нужные операции;
6. Программа вычисляет расстояние от каждого слова W1 до слов W2 и W3;
7. Программа присваивает метрику найденным словам из колонок W2 и W3 (у всех слов W2 метрика равна 100, для слов W3 задаётся уникальное значение в таблице);
8. Вычисляется величина C для каждого из заданных контекстных значений (слов W2);
9. Вычисленные контекстные значения программа сравнивает между собой (согласно наибольшему и определяется контекст документа);
10. Результат выводится пользователю.

В таблице 3.2 представлены данные анализа контекста в конкретных файлах. Расчеты контекста были проведены для более чем 10 файлов. Значения C вычислены по формуле 3.1. В скобках приведено число слов W1, W2 и W3, обнаруженных в конкретном документе.

Если бы для таблицы 3.1 в семантической цепочке слова “программа” среди слов-характеристик присутствовало слово blockchain (статья "Технология blockchain"[112]), то значение C для слова-значения "компьютер" было бы равно 32,54, а не 8,69. Из этого следует, что полнота семантической сети (таблицы 3.1)

существенно влияет на результаты оценки контекстного значения слова или документа.

Таблица 3.2.

Примеры результатов контекстного анализа

URL файла	Число слов	Корневые слова (W1)	Слова-значения (W2)	Слова-характеристики (W3)	Значения **C**
http://book.itep.ru/4/6/blockchain.htm "Технология blockchain"	5180	Программа (7)	Компьютер (3)	Объект (5) файл (24) код (6)	8,69
			Реализация проекта (9)	Этап (1) Инновация (2)	4,04
			План (0)	Годовой (1)	0,045
http://book.itep.ru/6/i2p.htm **"Стек протоколов I2P и немного о TOR"**	10812	Программа (5)	Компьютер (2)	Метка (30) Объект (7) Файл (5) Тестирование (9) код (19) html (13) сайт (6) бит (6)	9,51
			Реализация проекта (16)	Этап (9)	1,58
			План (0)	Обслуживание (1)	0,022
http://book.itep.ru/4/6/set_66.htm **"SET и другие системы осуществления платежей"**	40631	Программа (62)	Компьютер (0)	Объект (33) код (146) бит (14) массив (5) метка (4) переменная (5) исключение (6)	9,12
			Реализация проекта (18)	Этап (14) Стоимость (12)	2,38
			План (0)	Обслуживание (9)	0,059

Механизм распознавания контекста моделировался по методу Монте-Карло. Предполагалось, что в документе имеется N слов. При моделировании считалось, что положение слов в документе имеет постоянную плотность вероятности (слова размещены в документе статистически равномерно, что не всегда справедливо).

Для анализа в документ закидывали случайным образом слова "программа" и слова-характеристики.

На рис. 3.6 представлено распределение вероятности значений C при фиксированном положении слова "программа" и случайном распределении положений слов-характеристик (n=213) в документе, содержащем 40000 слов.

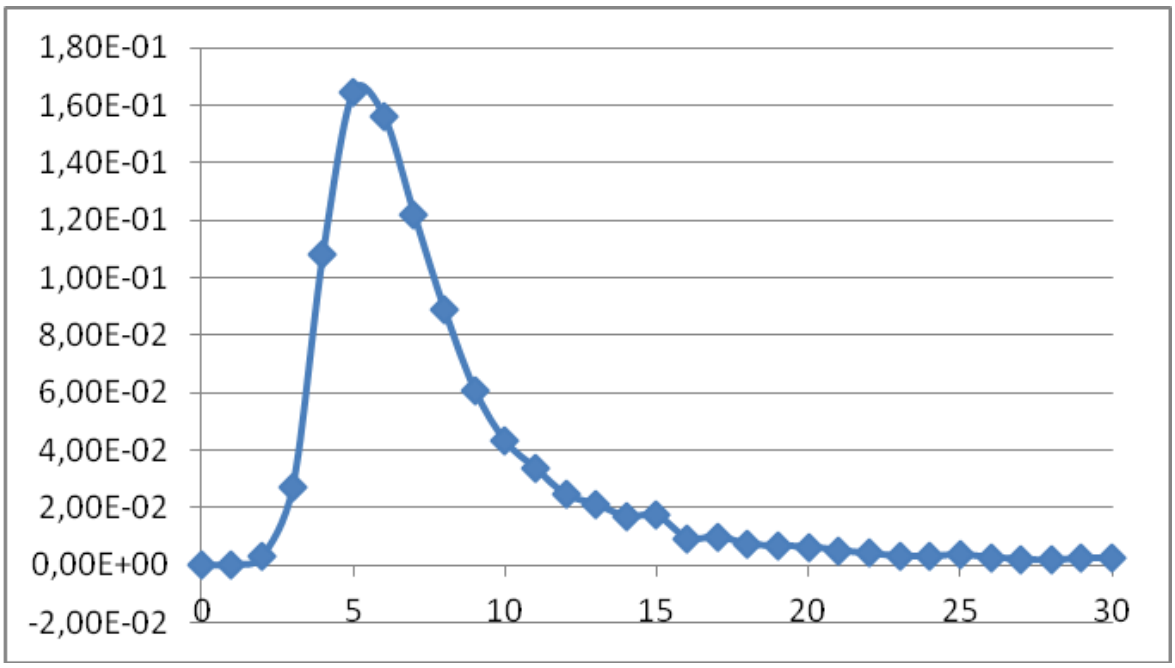

Рис. 3.6. Распределение плотности вероятности для значения C

По вертикальной оси отложено значение вероятности, а по горизонтальной - значение суммы C. Для выявления статистического распределения C расчет повторяется 10000 раз. Распределение C имеет гауссоподобную форму, но относительно длинный "хвост" в сторону больших значений C.

Распределение плотности вероятности позволяет оценить эффективность идентификации контекстных значений слов и документов.

Опробовались варианты, где вместо весовой функции $1/L_i$ используется $1/L^2$ или $\exp(-\alpha L)$, где α – постоянный коэффициент <1. Варианты сравнивались по отношению σ/C_{avr}., где C_{avr} - среднее значение C, вычисленное по формуле 3.1, а σ - средкеквадратичная ошибка определения C. Зависимость отношения σ/C_{avr} от формы весовой функции оказалась слабой. Для определенных классов документов могут использоваться специальные весовые функции, где при малых значениях L весовая функция характеризуется константой, а в области больших L быстро стремится к нулю.

На рис. 3.7 показана зависимость значения C (ромбики) и его среднеквадратичного отклонения (квадратики - σ) от числа слов-характеристик в документе (10÷150). Документ содержал 40000 слов.

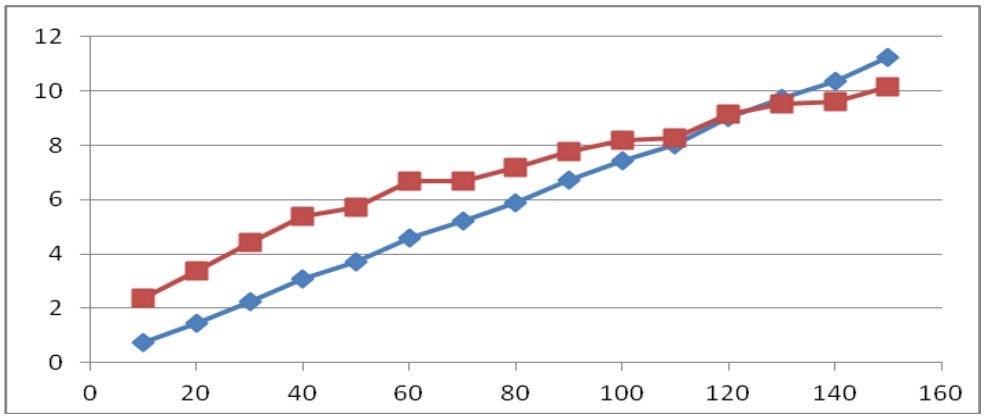

Рис. 3.7 Зависимость C (ромбики) и σ_C (квадратики) от числа слов-характеристик в документе (10-150)

Из рисунка видно, что значение среднеквадратичного отклонения C (σ_C) практически всегда больше C. Важно уметь определить, какова вероятность того, что полученное значение C задает корректно то или иное контекстное значение слова из первой колонки (W1).

Вероятность **p**, например, получения определенного значения C может быть оценена на основе распределения плотности вероятности. Вероятность P получения C=9,12 (см. рис. 3.6) равна 0,06, при этом вероятность C=2,38 <0,001.

В случае использования неравенства Чебышева [113] имеем:

$$p(|x-\overline{C}| \geq \Delta C) \leq (\sigma^2/(\Delta C)^2) \qquad (3.2)$$

Это неравенство определяет верхнюю границу вероятности того, что разность случайной величины x и $\overline{C}$ превышает определенный порог ΔC для произвольного распределения с дисперсией σ^2 и средним значением $\overline{C}$.

Рассмотрим третий пример из таблицы 3.2. При 62 словах "программа" в документе "SET и другие системы осуществления платежей"[114] можно вычислить значение для слова "компьютер" $\overline{C}$ = 9,12 и σ =14,0. Для слова "реализация" (программы) $\overline{C}$= 2,38, а σ=4,73.

$\Delta C = 9,12-2,38 = 6,74$ (разница между математическими ожиданиями взятых мной распределений).

Неравенство Чебышева для этого случая имеет вид:

$$P(|X-2,38|\geq(9,12-2,38)) \leq 4,73^2/(9,12-2,38)^2$$

$$P(|X-2,38|\geq 6,74) \leq 4,73^2/6,74^2$$

Исходя из этого получается что:

$$P(|X-2,38|\geq 6,74) \leq 0,49$$

Это вполне согласуется с оценкой по плотности вероятностей при моделировании (рис. 3.6) и подтверждает корректность распознавания контекста. Неравенство Чебышева удобно использовать, когда число слов W1 в документе достаточно велико.

Испытание алгоритма на полутора десятках текстовых файлах показало, что достоверная оценка получается в более чем 95% случаев. Сравнение проводилось для результатов оценки программой и человеком, а также путем численной оценки вероятности полученного результата. В частности, для документа о протоколе "SET и другие системы осуществления платежей" с числом слов 40631 и числом W3=213 получено отношение вероятностей оценки контекста для выражений "компьютерная программа" и "программа реализации проекта"= 60.

Предложенный метод оценки контекстных значений слов и документов нельзя считать универсальным. В нем, в частности, не учитываются смысловые связи. Но предложенный алгоритм не требует сложной программной реализации, серьезных вычислительных ресурсов и в большинстве случаев дает правильную оценку значения контекста.

3.5 Исследование алгоритма идентификации формализованных результатов внешней верификации программных модулей в базах данных

В процессе дальнейших исследований в методе, предложенном выше (формула 3.1) был обнаружен ещё один существенный недостаток. Он заключается в том, что в нём не учитываются расстояния между словами W1-W2.

Если в тексте имеется одно слово W1 и несколько слов W2, определяющих разные контекстные значения W1, то для определения истинного контекстного значения W1 можно воспользоваться формулой 3.1

Если в тексте имеется несколько слов W1, а размер текста достаточно велик, можно оценивать контекстное значение каждого слова W1 отдельно, определив границы области определения.

Считается, что W2 размещено после W1, а W3 – после W2 и только эти слова дают вклад в сумму для C, т.е. предыдущие слова не учитываются.

Пусть в файле имеется **n** идентичных слов W1 ($W1_1$, $W1_2$,…$W1_i$,…,$W1_n$). Пусть также имеются **k** идентичных слов W2 (см. рис. 3.8), определяющих контекстное значение W1 ($W2_1$, $W2_2$,…,$W2_j$,…,$W2_k$). Вообще говоря, каждое из значений слов W2 (да и W3) может встретиться в тексте более одного раза.

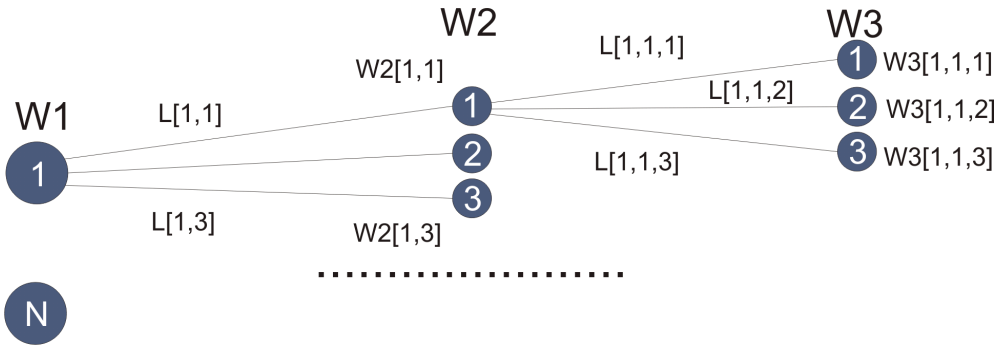

Рис. 3.8. Схема определения параметров

Пусть имеется m слов W3, соответствующих названным W2. L [W1i,W2j] – расстояние между словами W1i и W2j. В W2 [1,1] первый индекс относится к номеру слова W1, а второй – к номеру слова W2. В W3 [1,1,3] первый индекс

относится к номеру слова W1, второй – к номеру слова W2, а третий – к номеру слова W3. PL[1,1] – вероятность того, что расстояние между W1 и W2 [1,1] равно L [1,1]. На рис. 3.8 N соответствует числу слов W1 в тексте.

Любому слову W1 должно соответствовать два или более слов W2, содержащихся в тексте. Если присутствует только одно слово W1 и одно слово W2, ничего вычислять не нужно, контекст слова W1 очевиден, он определяется W2.

Если в документе содержится более одного образца слова-значения W2, соответствующего данному W1, то формула расчета контекстных значений W1 должна быть изменена.

$$C_{k,n} = \sum_{j=1}^{K} \frac{\sum_{i=1}^{m} (M_i \times f(L_i))}{D_j} \qquad (3.3),$$

где D_j характеризует расстояние от данного W1 до одного из слов W2 с номером j; K – число конкретных слов-значений W2 в документе. Если в документе слово W2=”code” встречается 4 раза, то K=4. $D_j = |L_j - L_0| + 1$. L_0 – минимальное расстояние от W1 до данного конкретного W2.

Пусть слово W1 может иметь **m** контекстных значений W2. Мощность множества **ω** всех W2 равно **m** (содержит **m** элементов). Реально в тексте может содержаться **k** слов W2 ($k \leq m$). $0 \leq k \leq m$, т.е. W2 присутствует не для всех возможных контекстных значений W1.

Если имеется только одно слово W2 и число W1>1, то при большом объеме текста можно попытаться определить наилучшее региональное значение контекста для некоторого слова W1.

Для каждого W2 из числа идентичных (число W2 > 1) производится расчет $C_{k,n}$. Очевидно, что эти величины будут отличаться друг от друга, так как для них расстояния W1-W2 будут разными.

Теорема:

Результат в формуле 3.3 всегда больше или равен результату в формуле 3.1

Доказательство:

$$\begin{cases} \sum_{j=1}^{k} \frac{\sum_{i=1}^{m}(M_i \times f(L_i))}{D_j} \geq \sum_{i=1}^{m}(M_i \times f(L_i)) \\ D \geq 1 \\ K \geq 1 \end{cases}$$

Если K=1, то

$$\sum_{j=1}^{k} \frac{\sum_{i=1}^{m}(M_i \times f(L_i))}{1} = \sum_{i=1}^{m}(M_i \times f(L_i))$$

$$\sum_{j=1}^{k} \frac{\sum_{i=1}^{m}(M_i \times f(L_i))}{D_j} = \sum_{i=1}^{m}(M_i \times f(L_i))$$

Если K>1, то

$$\sum_{j=1}^{k} \frac{\sum_{i=1}^{m}(M_i \times f(L_i))}{D_j} > \sum_{i=1}^{m}(M_i \times f(L_i))$$

$$\frac{\sum_{i=1}^{m}(M_i \times f(L_i))}{1} + \sum_{j=1}^{k-1} \frac{\sum_{i=1}^{m}(M_i \times f(L_i))}{D_j} > \sum_{i=1}^{m}(M_i \times f(L_i))$$

$$\sum_{j=1}^{k-1} \frac{\sum_{i=1}^{m}(M_i \times f(L_i))}{D_j} > 0$$

По итогу этих работ был разработан алгоритм идентификации формализованных результатов внешней верификации программных модулей. Данный алгоритм представлен на рис. 3.9

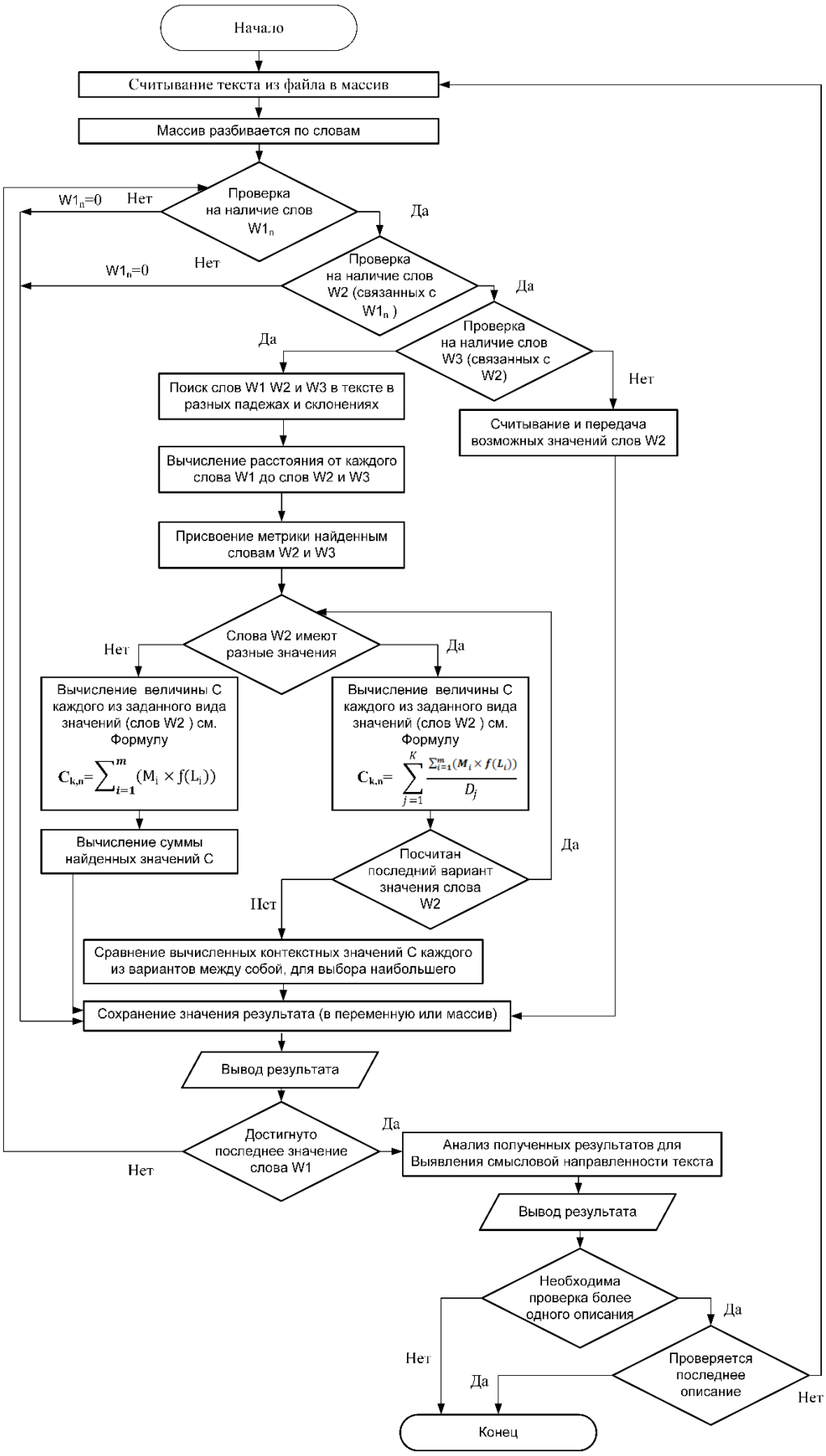

Рис. 3.9 Структурная схема алгоритма идентификации формализованных результатов внешней верификации программных модулей в базах данных

На основе алгоритма идентификации формализованных результатов внешней верификации программных модулей в базах данных была написана программа, предназначенная для оценки качества модулей банка алгоритмов [18]. Программа создана на основе методики определения контекста, изложенной в патенте [7]. По итогу работы программы выводится список результатов, в котором указаны: названия модулей, имена авторов, типы отзывов и информация о найденных авторами ошибках.

Таким образом, разработан алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных, отличающийся учетом расстояний между текстовыми элементами результатов верификации и их весов в специальной базе данных для подбора корректной семантики, обеспечивающий корректный результат в более чем 95% случаев.

3.6 Описание фаз работы транслятора CDTL

Работа такого транслятора подразделяется на 2 фазы: анализа (parser) и фазу синтеза. На фазе анализа (parser) определяется значение области проблематики. Её значение может быть задано в описании задачи явно и определяет, какой банк алгоритмов и какую библиотеку примитивов следует использовать. На фазе анализа описания задачи сначала формируется каталог с именем <имя> (имя каталога совпадает с именем файла описания задачи, заданным name). Исходный текст файла <имя>.pdl читается построчно и формируются таблицы: operators (операторов), descriptions (описаний), variables (переменных), arrays (массивов), objects (объектов), attributes (атрибутов) и parameters (параметров) и т. д. Все перечисленные файлы укладываются в каталог <имя>. Если в описании задачи содержится несколько описаний субзадач, то для них создаются субкаталоги с именами, совпадающими с именем субзадачи. В этих субкаталогах формируются соответствующие таблицы-файлы. Транслятор может распознавать вставки на языке Perl, добавленные в описание

задачи. Они должны быть выделены специальными тегами (<perl> текст программы на Perl </perl>). При выявлении ошибок формируется файл errors.pdl. Этот файл размещается в корневом каталоге описания задачи и является общим для всех субзадач. Все слова описания задачи должны быть распознаны. Если какое-то слово не распознано, производится запись в файл ошибок errors.pdl. Если в результате работы транслятора на фазе анализа обнаружена хотя бы одна ошибка в описании, фаза синтеза транслятора не запускается. Большинство файлов в таблице 3.3 формируются при анализе описания задачи. Исключение составляет только файл с текстом описания задачи.

Таблица 3.3.

Файлы, создаваемые и читаемые парсером

Имя файла	**Назначение**
<имя>.pdl	Текст описания задачи *(формируется пользователем)*
operators.pdl	Таблица описания системных операторов
errors.pdl	Таблица ошибок, найденных при чтении и анализе файла описания задачи
descriptions.pdl	Таблица описанных задач и субзадач
variables.pdl	Таблица описанных переменных
arrays.pdl	Таблица описанных массивов
objects.pdl	Таблица описанных объектов
attributes.pdl	Таблица описанных атрибутов
parameters.pdl	Таблица описанных параметров
composite.pdl	Таблица описанных композитных операторов
cycles.pdl	Таблица описанных циклов
internal.pdl	Таблица внутренних и системных переменных
primitives.pdl	Таблица использованных примитивов
perl.pdl	Таблица вставок на перле, добавленных в описание

Работа на фазе синтеза в данном трансляторе существенно отличается от аналогичной фазы других трансляторов [115-120]. На ней последовательно рассматриваются описания задач и с использованием имеющихся таблиц для каждого описания формируется текст программы на языке Perl.

Фаза синтеза может работать 3-мя способами:

1. Если в описании задачи не упоминается ни один из операторов действия, но имеются имена примитивов.

2. Если в описании присутствует хотя бы один оператор действия и нужно проводить поиск в банке алгоритмов (хранилище описаний алгоритмов), [121,122].

3. Если при тех же условиях что в пункте 2 в банке алгоритмов не найдено нужного модуля. В этом случае необходимый модуль придётся частично или полностью писать самому программисту.

В первом случае используются примитивы. Примитив – программный модуль, имеющий одну и только одну функцию, например, вычисление факториала или анализ записей журнального файла по какому-то параметру. Примитивы образуют библиотеки, каждая из них снабжается описанием (например, libr1.pm). Может существовать несколько тематических библиотек примитивов.

Программа, сгенерированная с помощью синтеза 1-го типа, может представлять собой список имен примитивов, между которыми могут добавляться короткие вставки кода на Perl. Желательно, чтобы примитивы и модули в банке алгоритмов были выполнены с соблюдением правил Хольцмана [6]. Это снизит трудоёмкость выявления и исправления программных ошибок.

Далее могут следовать описания входных параметров, а также характеристики и форматы выходных данных. Примитив обязательно должен содержаться в банке алгоритмов. Результирующая программа будет формироваться из суперпозиции примитивов.

Среди примитивов могут быть модули подпрограмм, например, для вычисления полинома любого порядка, или факториала, обращения матрицы, вычисление определителя матрицы произвольного порядка, упорядочивания списка по возрастанию или убыванию, вычисления среднего значения или дисперсии (для заданного списка результатов измерения) и т. д.

Имена примитивов надо отличать от программных модулей, записанных в банках. Поэтому было решено начинать их с **p**_(например **p_ip_to_num)**.

Мощность множества примитивов будет существенно меньше мощности множества программных модулей, хранящихся в банке алгоритмов.

Когда код формируется из примитивов и макросов, в этом случае CDTL выполняет функцию транслятора с макроязыка – транслирует описание задачи в код. Там, где в описании помещено имя примитива, в код вставляется обращение к примитиву, а там, где присутствует имя макроса, вставляется текст макроса. В этом варианте в описании задачи содержатся элементы алгоритма.

Во втором случае, как уже было сказано, необходимо провести поиск в банке алгоритмов. Поиск в банке алгоритмов проводится диалоговым методом, описанным в статье [121]. Главным преимуществом данного поиска является то, что он выдаёт не более одного результата. Если для решения проблемы нужны HTML-страницы, то они могут быть извлечены из банка алгоритмов, или написаны самим программистом.

В данном трансляторе скрипты error.pl, pars_access_2.pl и secur_cur.pl синтезируются 1-м способом, а скрипты filter.pl и proc.pl - 2-м способом.

3.7 Выводы по главе

1. Создан банк алгоритмов с помощью которого можно снизить число программных ошибок методом «Множества глаз». В данный банк заносятся описания программных модулей, сведения об авторах и т. д., а также даны ссылки на их код. Для создания данной базы была использована специальная таблица MySQL. Для занесения описаний в эту таблицу, использовалась разработанная для этого программа “Loader v 1.0” [14]. Данный банк алгоритмов добавлен в открытый доступ и предназначен для использования как людьми, так и программами, в том числе транслятором с языка CDTL на язык программирования высокого уровня.
2. Разработан и позднее усовершенствован алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных. Он может применяться для простого определения контекста слов и текстовых файлов с помощью ЭВМ без использования как без

использования нейросетей, так и в сочетании с ними. В этой диссертационной работе данный алгоритм был применён для анализа отзывов внешних пользователей на модули из банка алгоритмов, описанного в пункте 1.

3. Реализована проверка формул определения контекста слова с помощью неравенства Чебышева и метода Монте-Карло, которая показала высокий процент его эффективности [2,3]. Эти формулы были разработаны для внешней верификации программных модулей в базах данных. Таким образом, метод внешней верификации программных модулей в базах данных проверен с помощью программ, на которые были получены авторские свидетельства [11,12]. Проверка показала, что данный метод даёт верный результат в более чем 95% случаев.
4. Для апробации метаязыка описания задач предложена структура специального транслятора. Каждая из фаз его работы имеет свои особенности, что делает структуру такого транслятора уникальной

4. Экспериментальное исследование методов снижения числа программных ошибок на основе сокращения участия программиста

4.1 Апробация метаязыка описания задачи CDTL с помощью транслятора при генерации пакета программ для защиты от атак WEB-сервера

На данный момент программное обеспечение всё более и более внедряется в нашу жизнь. Поэтому серьёзность последствий программных ошибок со временем только возрастает и сокращение их количества в программном обеспечении способно во многом улучшить качество нашей жизни. Уже сейчас в мире произошла масса катастроф, причинами которых стали ошибки в программах [103]. В ядре операционной системы Windows 0,5 ошибки на 1000 строк кода. Оценка сделана методом экстраполяции и анализа временных рядов, но исходные данные недоступны. Но даже одна небольшая ошибка может стать причиной серьёзных последствий.

Для улучшения качества кода программ, мной были разработаны методы снижения числа программных ошибок и метаязык CDTL, описанные в главе 2.

Алгоритм динамического выбора состава модулей для решения коллектива задач одного типа был применен при создании библиотеки для gnuplot gnu.pm. gnu.pm также использует для своей работы библиотеку Chart::Graph::Gnuplot, которая находится на сайте http://cpan.org/ в открытом доступе. В библиотеку включен код стандартной программы для построения графика. Параметры для данной библиотеки задаются внутри программы, её использующей, и передаются ей обозначенным выше способом. В результате чего в самой программе прописываются только значения параметров для библиотеки и строка их передачи. Библиотеке gnuplot передаются следующие параметры: название всей картинки, название осей графика, название линий графика, файл данных для графика. В программах безопасности для защиты Web-сервера библиотека используется 15 раз. При её использовании каждый программный модуль составляет 10 строк кода и общая их сумма равна 150. Сама библиотека

включает в себя 23 строки кода. Суммарно библиотека и все программные модули дают 173 строки кода (150+23). Размер такого же модуля без использования библиотеки составляет 14 строк кода. А общее их количество без использования библиотеки gnu.pm получилось равным 210. В итоге при использовании метода описанного в статье [1] строк кода получается меньше на 17,6 %. А чем меньше будет написано программного кода, тем меньшее количество ошибок может быть совершено.

Для *апробации метаязыка CDTL* был создан транслятор (см. рис. 4.1.), который читает описание задачи (расширение имени файла .pdl) строка за строкой и формирует таблицы для: descriptions, variables, arrays, operators, primitives и т.д. и формирует на выходе файл программы на perl - name.pl (name представляет в данном случае имя формируемой программы).

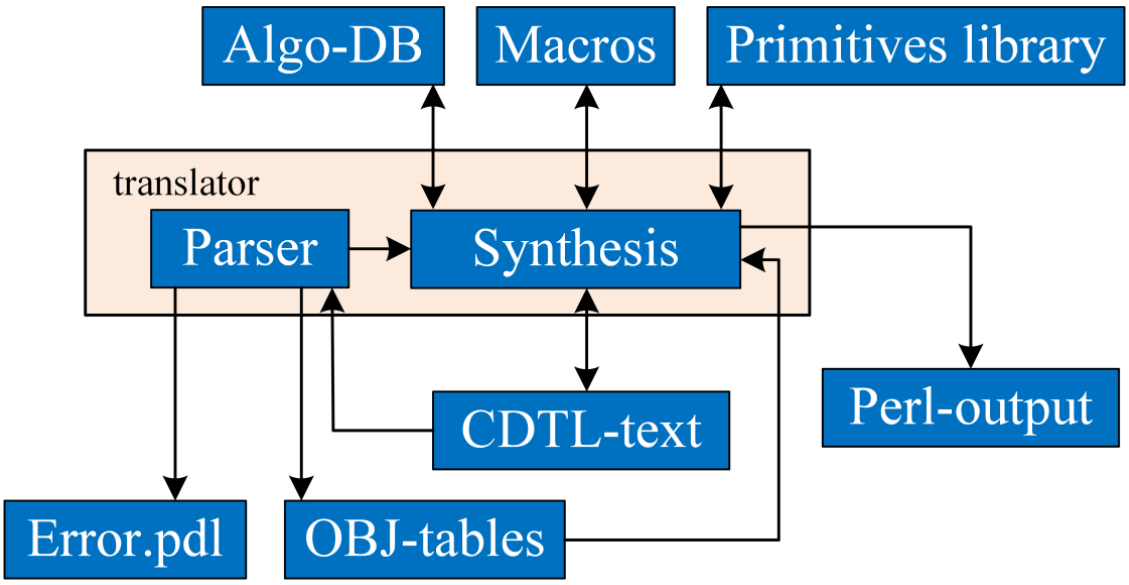

Рис. 4.1. Структурная схема динамического транслятора описания задачи в язык Perl. CDTL-text - текст описания задачи на CDTL; ALGO-DB - банк описаний алгоритмов; Perl-output - программа, формируемая транслятором (результат); error.pdl - журнал ошибок, выявленных парсером; OBJ-tables - таблицы результатов анализа текста описания; Primitives library – библиотека примитивов; Macros - библиотек макросов

Синтаксис записей в строках проверяется по мере их чтения. При выявлении ошибок производится запись в журнальный файл ошибок (errors.pdl). Заносится номер строки, сама строка с ошибкой и указывается слово, которое

содержит ошибку. Каждому типу ошибки присваивается определенный код, который будет также занесен в файл errors.pdl.

Каждое из слов проверяется на наличие его в таблицах, в частности, в таблице operators. Каждое слово в таблицах характеризуется его принадлежностью к определенному классу: actions, objects, attributes, parameters, operators, composite operators и т. д. В транслятор встроены разные таблицы для разных классов.

В результате работы транслятора генерируется специальный комплекс программ на языке Perl, обеспечивающий безопасность Web-сервера.

Данная программа должна анализировать журнальные файлы (access_log, error_log, secure и др.), выявлять случаи атак и блокировать доступ к WEB-узлу с соответствующего IP-адреса, если число атак превысило пороговый уровень.

Таким образом разработана структура динамического транслятора описания задачи в язык Perl, отличающаяся автоматическим учетом контекста задачи и обеспечивающая проектирование программы решения задачи на языке высокого уровня.

Задание порогов атак отдельно для каждого типа, имя базы данных, таблицы (если имеются), имена журнальных файлов для каждого типа атак, требования к формированию графики производится в темном блоке на рис. 4.2 “Зона обработки результата”).

Предполагается, что в банке описаний алгоритмов имеются модули, соответствующие субзадачам, названным в вертикальных прямоугольниках рис. 4.2. Может выбираться один вариант или любая их комбинация с помощью диалогового алгоритма, описанного в главе 2. После завершения диалога программа формирует из имеющихся модулей пакет программ, решающих данную проблему самостоятельно.

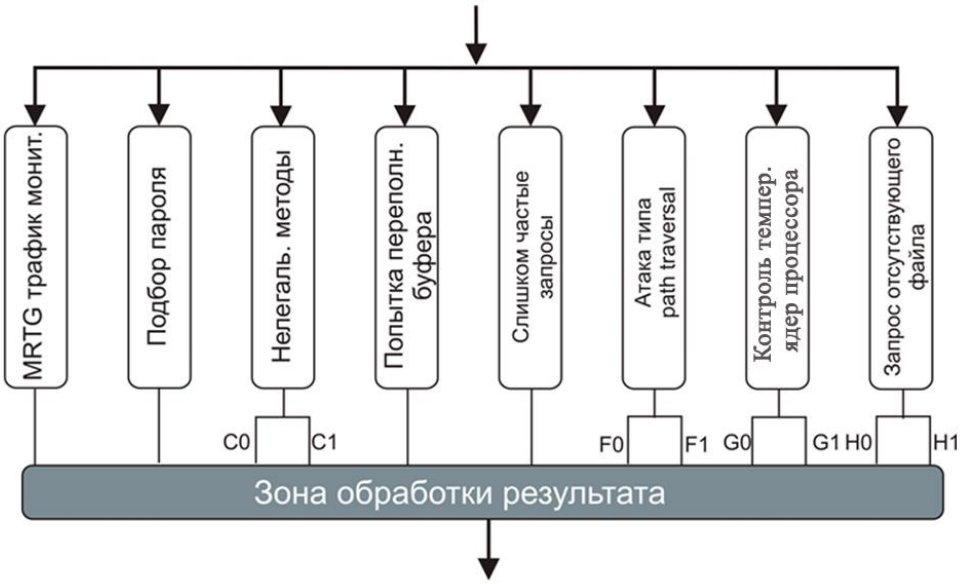

Рис. 4.2. Структура конфигурации программы безопасности WEB-сервера
Обозначения: C0 – использование методов put или delete. C1 – использование методов post и т.д.. F0/1 – детектирование в журнальном файле access_log случаев "../" , "/../../../../", %2e%2e%2f, ..%c0%af или %2e%2e/ отдельно.

Этот пакет программ мониторирует входящий и исходящий информационный трафик, детектирует атаки переполнения буфера, попытки DoS-атак и path traversal, а также случаи попыток использования нелегальных HTTP-методов (put, delete и пр.) и некоторых других атак (см. рис. 4.3).

Рис. 4.3 Полное меню пакета программ защиты Web-сервера

Программа не реже раза в неделю блокирует доступ к WEB-серверу для одного и более подозрительных ip-адресов без участия человека (процедуры записываются в журнальный файл). Чтобы оказаться заблокированным, атакующий должен совершить не менее 5 попыток. Всего выявлено и автоматически заблокировано уже несколько тысяч источников угроз. Их активность фиксируется в специальной базе данных. Сопоставимые результаты можно получить, используя какой-либо IDS-IPS (например, SNORT), но подобные системы требуют больших вычислительных ресурсов, которыми мы не располагали (число сигнатур атак давно превышает 50 млн). Система защиты Web-сервера ежемесячно публикует на сайте список топ-10 самых активных ip-адресов, с которых осуществляются попытки взлома нашего сервера, с указанием времени, IP и типа атак.

Данный программный пакет включает 8 модулей безопасности. Каждый из них защищает сайт от определённого типа атак. Вот список этих модулей:

1. *MRTG-monitoring.*
2. *Pass guessing.*
3. *Illegal method.*
4. *Buffer overflow attack.*
5. *High request frequency.*
6. *Path traversal attack.*
7. *Temper-monitoring.*
8. *Missing file request.*

Подробное описание каждого модуля дано в разделе 4.2.

4.2 Характеристика программных модулей, используемых при тестировании метаязыка описания задач

4.2.1. Модуль “mrtg-monitoring”

Модуль “mrtg-monitoring” сейчас особенно актуален для диагностики атак, поскольку с помощью него измеряется входящий и исходящий трафик. Атака может быть выявлена по резкому скачку трафика. Программа MRTG [123], которую создал Тобиас Оетикер, позволяет измерять трафик с помощью MIB переменных через протокол SNMP [124]. Например, с помощью MRTG можно

отследить загрузку канала в байтах или в пакетах в секунду. Примеры таких графиков предоставлены ниже на рисунках 4.4 и 4.5.

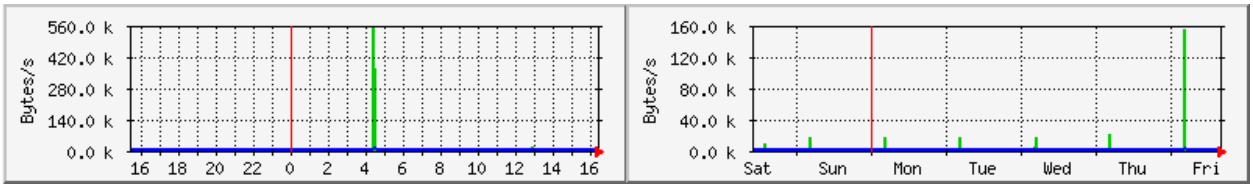

Рис. 4.4 Входящие и исходящие байты за сутки и за неделю

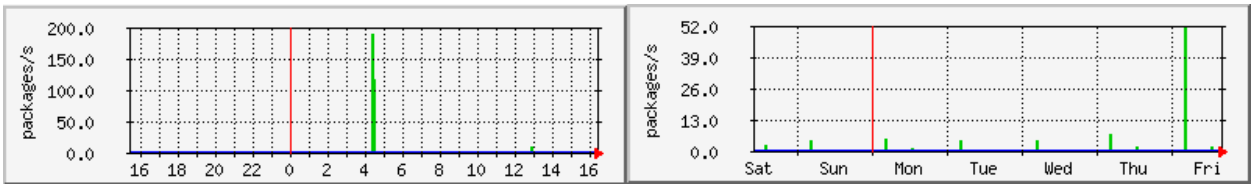

Рис. 4.5 - посланные (не повторно) и повторно пересылаемые сегменты за сутки и за неделю

По пикам во временном распределении с использованием MRTG выявляются IP-адреса ботов. В локальной секции сети это позволяет выявить mac-flood атаки с последующим выявлением попыток перехвата вводимых паролей, а также перехвата паролей .htaccess. По пикам трафика можно детектировать попытки хакеров подобрать ISN с последующим перехватом TCP-сессии (уже после авторизации легального пользователя) для вторжения или чтения чужой почты. Можно пытаться детектировать случаи установки RPC-демонов.

Анализируя всплески трафика, можно детектировать попытки провокации занесения почтового адреса жертвы в репутационный список и блокировки любых его почтовых обменов. На некоторых MRTG-диаграммах на сервере Saturn предусмотрена возможность кликанья мышкой на пике трафика и получения IP-адресов инициаторов.

Кроме того, измерены средние значения потоков для разных протоколов и номеров портов и значимые отклонения от этих средних значений вызывают сигнал тревоги.

4.2.2. Модуль “Pass guessing”

Модуль “Pass guessing” создан для обнаружения атак подбора пароля [125]. Такая атака осуществляется путём угадывания пароля в том числе и программным методом (путём перебора различных вариантов пароля). В рамках данного модуля в пакете программ для защиты web-сервера работает файл secur_cur.pl.

Он проводит анализ записей внутри /var/log/secure и строит распределение атак подбора пароля в текущем времени. На рисунке 4.6 показано такое распределение по ip-адресам для web-сервера saturn.itep.ru

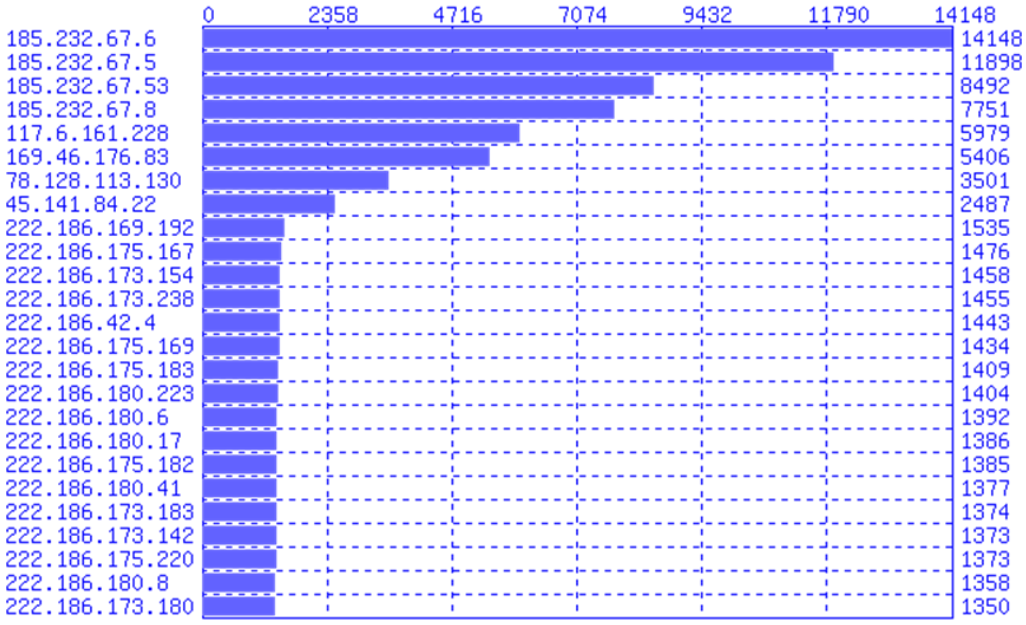

Рис. 4.6 Статистика атак подбора пароля по ip-адресам

Ещё эта программа создаёт распределение для атак подбора пароля за сутки. Для web-сервера saturn.itep.ru оно выглядит так (см. рис. 4.7):

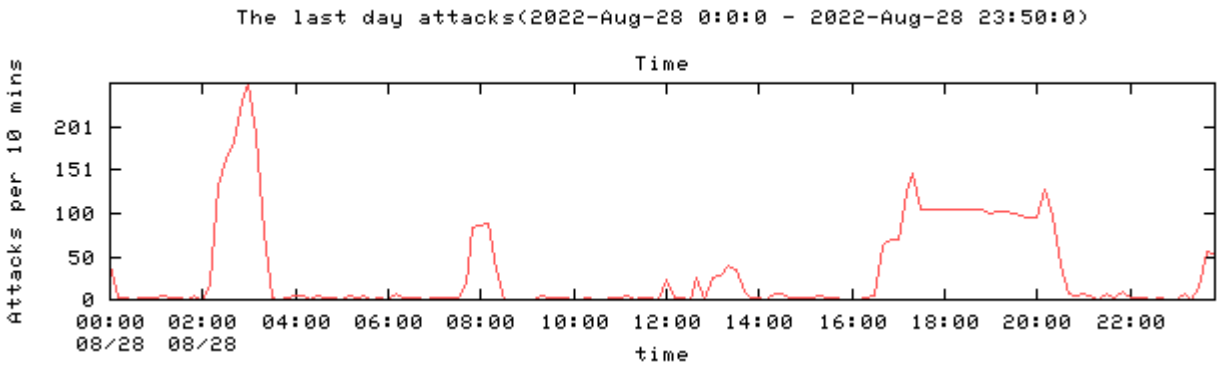

Рис. 4.7 Статистика атак подбора пароля за сутки

4.2.3. Модуль “Illegal method”

Модуль “Illegal method” предназначен для выявления использования нелегальных методов с целью взлома сайта. С помощью программ данного модуля выявляются строки запросов, содержащие put, options, delete, link и PROFIND.

В рамках данного модуля в пакете программ для защиты web-сервера частично используется pars_access2.pl. Данная программа в процессе своей работы анализирует файл access_log записывает случаи нелегальных WEB-методов и запросов в файл cgi.txt. Запросы, содержащие put, options, delete, link и PROPFIND данная программа передаёт в файл illegal.txt

Методы put, options, delete, link и PROFIND описаны в RFC. Сейчас сайт book.itep.ru генерируется с помощью базы данных. Был такой случай, что с 11 часов вечера до 3 часов ночи студент МФТИ из общежития в Долгопрудном стер вручную несколько сотен записей на сайте. Он не знал, что все процедуры пишутся в журнал. Но с тех пор было решено перестраховаться и объявлять методы put, delete и др. нелегальными, их использование воспринимается программой как попытка атаки. Система детектирует не менее 3-12 таких попыток в сутки на сервере Saturn.itep.ru.

4.2.4. Модуль “ Buffer overflow attack”

Модуль “Buffer overflow attack” создан для выявления атак переполнения буфера [126]. Для осуществления атаки такого вида необходимо превысить объём памяти, отведённый для временного хранения данных. Если удаётся это сделать, атакующий получает возможность выполнять на взломанной машине любые операции. Сейчас такие атаки считаются одними из наиболее распространённых.

В пакете программ для защиты web-сервера в рамках данного модуля частично используются программа pars_access2.pl. С помощью неё создаётся распределение атак по IP с длиной запроса более 250 байт.

Хотелось бы добавить, что данная система безопасности WEB-сервера не является универсальной, а настроена под определённые задачи. Система безопасности должна реализовать все возможные ограничения по любым параметрам, поэтому была разработана функция ограничения длинны запросов. Были проанализированы легальные и нелегальные запросы за несколько лет и после этого выработаны критерии отбора. Передача файлов в качестве параметра в данной системе запрещена.

Такое распределение для сервера saturn.itep.ru показано на рисунке 4.8. В данном случае оно строится только для длинных запросов и указывает на то, что скорее всего это была атака переполнения буфера.

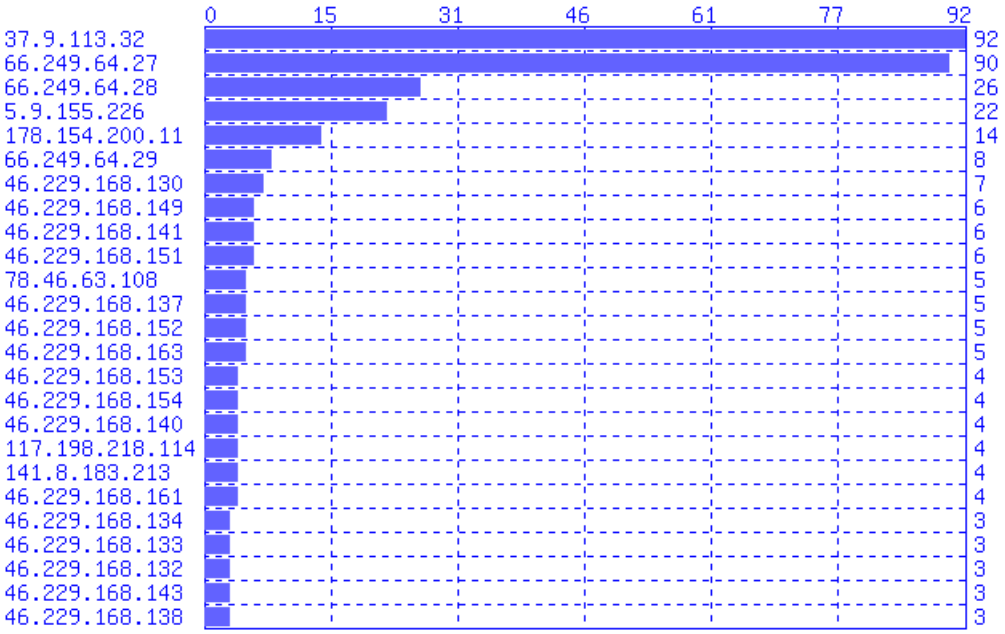

Рис. 4.8 Распределение запросов длинной более 250 байт по ip

На этом рисунке есть и ip-адреса, с которых поступали данные запросы.

4.2.5. Модуль “High request frequency”

Модуль “High request frequency” создан для выявления атак, использующих слишком частые запросы или DDoS-атак [127]. Атака данного вида осуществляется путём отправления на веб-ресурс будущей жертвы множества запросов за короткое время. Это делается для того, чтобы случился отказ в обслуживании запросов других пользователей из-за невозможности их обработки. В результате такой атаки сайт жертвы перестаёт функционировать.

В пакете программ для защиты web-сервера, в рамках данного модуля, частично используется программа pars_access2.pl. Она создаёт файл atk_dens.txt. В него записываются ip-адреса компьютеров с которых поступало 3 и более запросов в секунду. Рядом с ip-адресами указывается количество поступивших запросов.

4.2.6. Модуль “Path traversal attack”

Модуль “Path traversal attack” разработан с целью предотвращения доступа сторонних лиц к файлам и каталогам сервера, находящимся вне досягаемости для них [128, 129].

Атаку можно классифицировать как path traversal или slash, если злоумышленник пытался получить доступ к файлам и каталогам, находящимся вне пределов, определенных конфигурацией (web root folder) [128]. Просматривая какое-нибудь приложение атакующий пытается узнать абсолютные пути к файлам и каталогам, находящимся на машине предполагаемой жертвы. При атаке данного вида нарушитель добавляет в свои запросы последовательности типа “../”, пытаясь войти в корневой каталог. При такой атаке может использоваться вредоносный код, добавленный в запрос.

В пакете программ, описанном в этой диссертационной работе, в рамках данного модуля частично используется программа pars_access2.pl. Данная программа создаёт файл atk_ip.txt и записывает в него ip-адрес компьютера, с которого поступают запросы с “////” и более и проставляет количество этих атак.

4.2.7. Модуль “Temper-monitoring”

Ещё для безопасности Web-сервера необходимо следить за тем, чтобы ядра процессора не перегревались, иначе это может привести к поломке компьютера. Для решения этой проблемы был создан отдельный программный модуль “Программа для защиты от перегрева ядер процессора ЭВМ v 1.0” [13].

Он предназначен для получения информации о температуре ядер процессора и записи её в отдельные текстовые файлы для каждого ядра с указанием времени события. Помимо этого, данный модуль выполняет

автоматическое построение графика температуры ядер процессора за последние сутки с помощью программы Gnuplot и отображает его на специальной HTML странице. Такой график за сутки для сервера Saturn.itep.ru показан на рисунке 4.9.

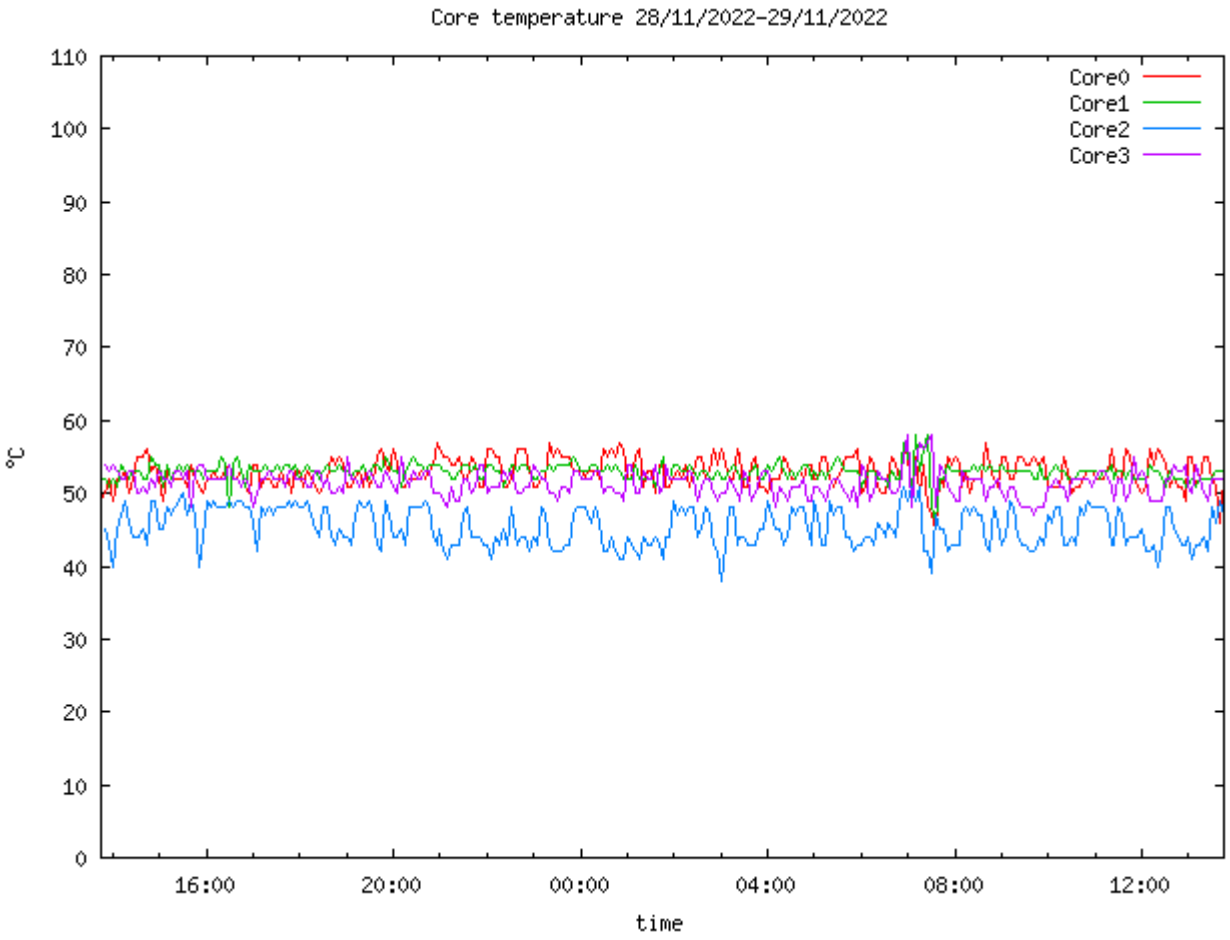

Рис. 4.9 График температуры ядер процессора сервера Saturn.itep.ru за сутки

Кулеры могут ломаться, например если задеть защелку. Система мониторинга температуры ядер процессора может быть полезна тем, что присылает сообщение с уведомлением о повышении температуры и временной меткой, что мгновенно решает проблему. Это позволяет своевременно зафиксировать ухудшение работы кулеров и исключить необходимость прихода на работу в нерабочее время.

Ещё такая программа может использоваться для диагностики скрытого майнинга [130,131]. Майнеры предпочитают использовать чужие компьютеры

потому что для самой процедуры добычи криптовалюты требуется много ресурсов процессора и электроэнергии. И чтобы не тратить на это деньги, майнеры создают большую сеть из заражённых соответствующим вирусом чужих компьютеров. Но дело в том, что при интенсивном использовании процессора, температура ядер его возрастает, момент изменения их температур программа зафиксирует на графике (см. рис. 4.9). При резком изменении температур ядер процессора без известной на то причины можно подозревать активность вредоносных программ.

Так “Программа для защиты от перегрева ядер процессора ЭВМ v 1.0” может помочь обнаружить влияние на сервер не только майнеров, но и других программ злоумышленников, пытающихся использовать процессор.

4.2.8. Модуль “Missing file request”

Модуль “Missing file request” отслеживает поиск несуществующего файла. Примером такого поиска может быть попытка найти приложение PhpMyAdmin [132] на сайте, где он не установлен или переименован. Это означает, что хакер знает уязвимости данной программы и специально пытается их найти, чтобы взломать компьютер. Поскольку данное приложение является популярным и им много кто пользуется, велика вероятность найти компьютеры с этим приложением и их взломать. Для защиты от подобной атаки, если PhpMyAdmin установлен, можно например переименовать ссылку на него или закрыть доступ к данной странице из интернета.

Для обнаружения таких атак в пакете программ для защиты web-сервера используется программа error.pl. С помощью неё информация о попытках найти несуществующую страницу извлекается из файла “error_log”. Помимо этого данная программа создаёт распределение по времени для подобных попыток. На рисунке 4.10 такое распределение для сервера saturn.itep.ru показано в виде графика.

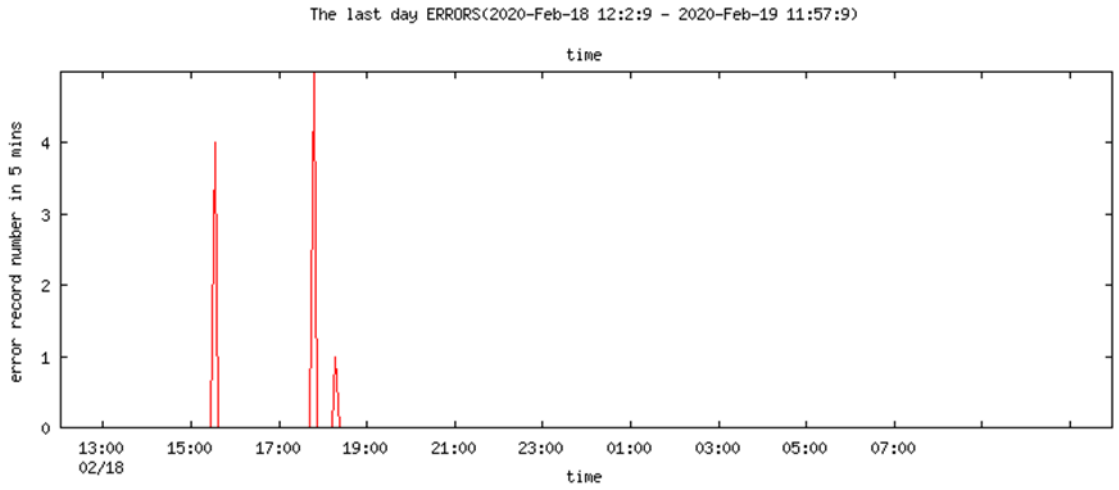

Рис. 4.10 Распределение попыток запроса отсутствующего файла при обращении к серверу за последние сутки

4.3 Экспериментальная оценка методов контекстного анализа слова и файла

Для того, чтобы проверить функциональность данной формулы, были написаны специальные программы и по одной из них было получено авторское свидетельство [12]. С помощью этой программы проверялась работа формулы на текстах, а также вариации и валидность значения C в зависимости от ширины площадки. Вводится площадка шириной **a**, в зоне которой $f(L_i)$ принимает значение 1. $f(L_i)$ от окончания площадки работает также, как она бы работала от L=1 без неё. Например если $f(L_i) = 1/L_i$, а ширина площадки a=100, то при L_i=101 $f(L_i)$=1, а при L_i=102 $f(L_i)$=0,5.

Для того, чтобы сопоставить работу формул 3.1 и 3.3 были взяты 100 произвольных отрывков текста с количеством слов не менее 500. В каждом из этих отрывков было одно слово “программа” (W_1) и одно слово “компьютер” (W_2) и произвольное количество слов W_3, связанных с ним. Для формулы 2 были также взяты 100 отрывков текста с количеством слов не менее 500. В каждом из этих отрывков текста было одно слово “программа” (W_1) и несколько слов “компьютер” (W_2) и произвольное количество слов W_3.

Были проведены расчёты дисперсии, среднеквадратичного отклонения σ, среднего значения C и отношения σ/C_{avr} для весовых функций 1/L, 1,

1/(1+Log(L)) и для формул 3.1 и 3.3 при значениях площадок a=100 и 300. Рис. 4.11 поясняет смысл весовых функций и их параметров. f(L=1) для всех вариантов =1.

Результаты расчётов представлены ниже в таблицах 4.1 и 4.2. В таблице 4.1 приведены результаты для формулы 3.1. В таблице 4.2 соответственно собраны результаты для формулы 3.3. В каждой из таблиц рассмотрены различные варианты весовой функции.

В первом варианте применена функция $1/L_i$. Во втором случае она тоже равна $1/L_i$, но при малых значениях L была введена ширина площадки a, в зоне которой принимает значение 1 (см. рис. 4.11). Были проведены расчёты для a=100 и a=300 (см. табл. 4.1 и 4.2).

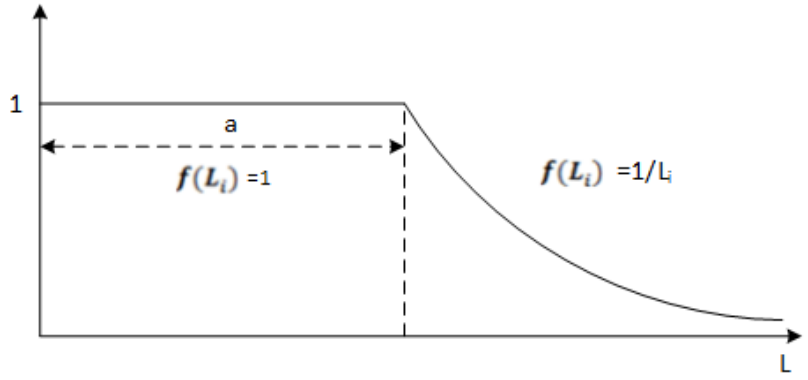

Рис. 4.11 Возможные формы весовой функции

Такие же расчёты с площадкой были проведены для $f(L_i)$ = 1/(1+log(L_i)). Данный вариант был введён для того, чтобы уменьшить влияние больших расстояний L_i на значение C.

Самым удачным вариантом оказалась функция $f(L_i)$ = 1/(1+log(L_i)) с площадкой равной 300. Причём как для формулы 3.3, так и для 3.1. Это связано с тем, что увеличивается влияние далеко стоящих слов близких по контексту. Это даёт преимущество по сравнению с вариантами, где $f(L_i)$ = 1/L.

Однако при этом расстояние всё равно учитывается, и если слово находится слишком уж далеко, оно не будет сильно влиять, но это возможно только в случае очень больших файлов.

Таблица 4.1 для формулы 3.1

Версия весовой функции	σ	C_{avr}	σ / C_{avr}
$1/L_i$	13,34	7,22	1,85
$1/L_i$ a=100	89,11	58,17	1,53
$1/L_i$ a=300	198,80	149,87	1,33
$1/(1+\log(L_i))$	278,02	221,41	1,26
$1/(1+\log(L_i))$ a=100	300,93	253,45	1,19
$1/(1+\log(L_i))$ a=300	369,72	317,29	**1,17**
1	1061,53	774,58	1,37

Таблица 4.2 для формулы 3.3

Версия весовой функции	σ	C_{avr}	σ/C_{avr}
$1/L_i$	9,66	7,64	1,26
$1/L_i$ a=100	108,35	83,19	1,30
$1/L_i$ a=300	245,36	183,66	1,34
$1/(1+\log(L_i))$	328,40	276,49	1,19
$1/(1+\log(L_i))$ a=100	351,13	324,26	1,08
$1/(1+\log(L_i))$ a=300	411,93	394,16	**1,05**
1	1434,12	1000,43	1,43

Наименьшее соотношение σ/C_{avr} получилось для второй формулы (см. табл. 4.1 и 4.2). Чем оно меньше, тем меньше шансов, что программа неверно определит контекст слова. Поэтому можно сказать, что во второй формуле получился лучший результат, чем в первой. Преимущества второй, формулы связаны с тем, что в ней учитываются расстояния между W1 и W2.

Для разных видов формулы были построены распределения плотности вероятности. При построении этих распределений выбран вариант с наименьшим соотношением σ/C_{avr} .

На графиках по оси абсцисс отложены значения C, а по оси ординат указана вероятность, с которой C может попасть в тот или иной интервал значений. Видно, что вероятность чуть-чуть увеличивается от нуля и дальше падает.

Поэтому можно сказать, что такие распределения имеют гауссоподобную форму с длинным хвостом в сторону больших значений C. Подобное распределение было получено и при моделировании значений C на рисунке 3.6. Это значит, что результаты расчёта значений C на реальных текстах подтверждают данные моделирования.

Из рисунков 4.12 и 4.13 видно, что дисперсия C велика. Этому способствуют вариации размеров файлов, чисел слов W2 и W3, а также их распределение по тексту. На рисунке 3.6 было показано, что даже при такой дисперсии вероятность правильного определения контекста выше 90%. Тем не менее, любые методы снижения дисперсии должны приветствоваться.

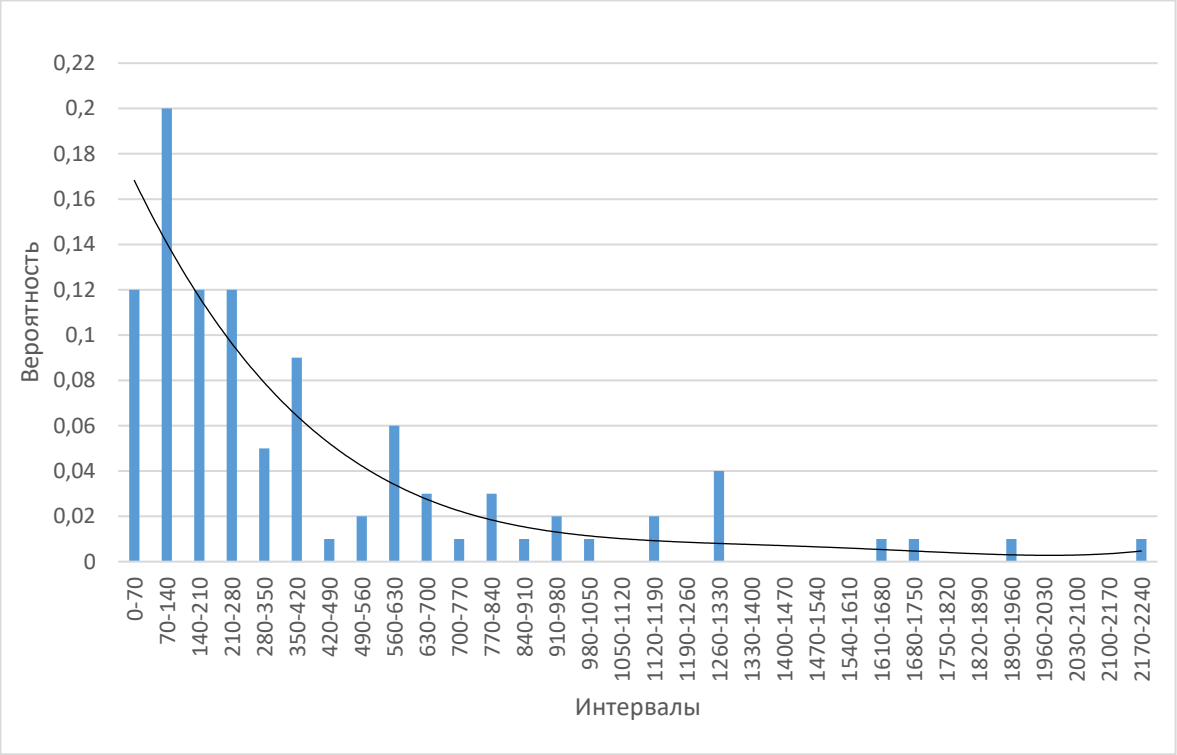

Рис. 4.12. Распределение плотности вероятности для C (формула 3.3) при весовой функции 1/(1+log(L)) и ширине площадки a=300

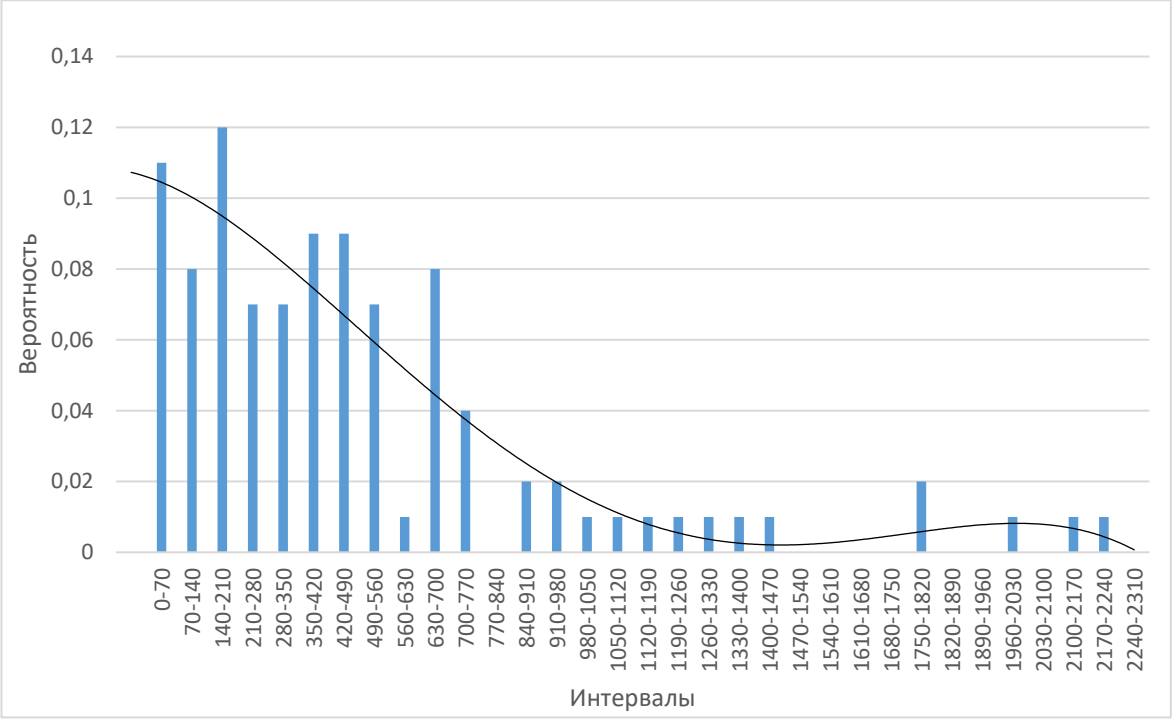

Рис. 4.13. Распределение плотности вероятности для C (формула 3.1) при весовой функции 1/(1+log(L)) и ширине площадки a=300

На рис. 3.6 при моделировании предполагалось однородное распределение слов в файле и даже в этом варианте дисперсия была значительной. На самом деле вариации L могут быть очень большими. На рис. 4.14 показано распределение L для слов W2-W3 для одного из файлов. По оси ординат отложено число случаев с данным расстоянием L для L=300. Файл содержал 75041 байт.

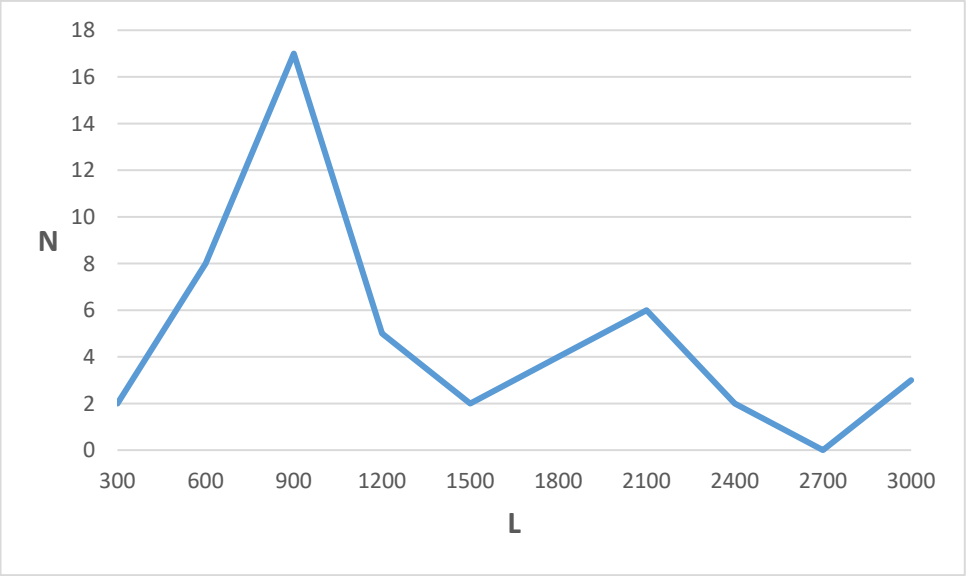

Рис. 4.14. Распределение расстояний W2-W3 для одного из файлов

Разброс L приводит к дополнительному росту дисперсии, что частично компенсируется за счет усреднения.

Было проведено исследование зависимости отношения σ/C_{avr} от среднего размера текстового файла. Зависимость оказалась слабой.

Как показали исследования, предложенная формула улучшает результаты формулы 3.1 (на которую был получен патент [7]). Удалось выбрать оптимальную весовую функцию. Данная техника может найти применение при анализе смысла текстов, и для снижения числа программных ошибок методом описания задачи, а не алгоритма [1].

4.4 Оценка сокращения сложности разработки программ на CDTL с помощью метрики Холстеда и Метрики Джилба

Использование метрик программного кода даёт возможность оценивать разнообразные характеристики ПО. С помощью них можно рассчитать сложность или надежность кода программы [86]. Это очень важно, потому что из-за большой сложности могут серьёзно увеличиваться время и усилия, затраченные на разработку программного продукта.

Чем проще написать программный код, тем меньше человек в нём сделает ошибок. Сложность программ защиты Web-сервера на метаязыке CDTL и на Perl, была вычислена с помощью 2-х метрик разного вида. Это метрика Джилба и метрика Холстеда [86]. Они являются количественными, поэтому предназначены для работы с исходным кодом и сложность с их помощью вычислить достаточно просто. Данные метрики были подробно описаны в главе 1, разделе 1.4. Сложность программ на Perl по метрике Холстеда равна 104,92, что выше аналогичного показателя программ CDTL (23,61) на 77,5%. Сложность программ на Perl по метрике Джилба равна 153, что выше аналогичного показателя программ CDTL (4) на 97,39%.

Увеличение разницы в значениях при вычислениях с помощью метрики Джилба связано с тем, что там сравнивается количество операторов типа IF-

THEN-ELSE, а в метрике Холстеда используются любые операторы и операнды. Это говорит о том, что в метаязыке CDTL меньше операторов IF-THEN-ELSE, где вероятнее совершить ошибки программисту. Из этих показателей следует, что программный код на CDTL написать проще, чем на Perl. Поэтому при использовании CDTL человек будет совершать меньше ошибок.

4.5 Оценка сокращения количества ошибок в программах при применении метаязыка CDTL с помощью метрики Холстеда и простой интуитивной модели.

Основным преимуществом CDTL по сравнению с алгоритмическими языками является минимизация человеческого участия в генерации кода программы [81, 95, 121, 133, 134]. Благодаря этому сокращается и количество ошибок, которые могут возникнуть, а также время, которое затрачивается на написание программы. Снижаётся трудоёмкость процесса написания программного кода, благодаря чему большее количество людей могут принимать в нём участие.

Снижения числа ошибок при использовании CDTL происходит в основном за счёт сокращения количества программного кода. Поэтому для определения сокращения количества программных ошибок, которое должно произойти благодаря применению CDTL, наиболее удобно использовать простую интуитивную модель и метрику Холстеда, которые описаны в первой главе данной диссертации (раздел 1.5).

Правда надо заметить, что Метрика Холстеда имеет небольшой недостаток, в ней учитываются только те ошибки, которые пользователь создаёт в описании. Но данный недостаток устраняется за счёт использования интуитивной модели. Она была применена для дополнительной проверки результатов метрики Холстеда. С помощью неё сравнивался код на Perl, написанный вручную, с кодом на Perl, сгенерированным транслятором. Такое сравнение было необходимо, чтобы убедиться, что в коде программ, сгенерированных из

описаний на CDTL не будет столько же ошибок, как и в написанных на Perl. Ошибки могут быть в программных модулях, примитивах или самом трансляторе. И если их много, они могут повлиять на результаты работы программы. Чтобы проверить, насколько существенно такое влияние, была использована простая интуитивная модель.

Сокращение ошибок, при расчёте с помощью интуитивной модели получилось даже более существенным, чем при использовании метрики Холстеда. Однако отличия небольшие. Результаты расчетов по данной методике представлены в таблице 4.3.

Таблица 4.3.

Расчёт сокращения количества ошибок с помощью простой интуитивной модели и метрики Холстеда в пакете программ защиты Web - сервера при использовании CDTL.

Количество строк кода		Объём в битах		Приблизительное количество ошибок, шт.		Сокращение ошибок в % при использовании CDTL	
в описании задачи на CDTL	в программе на Perl	Описания проблемы на CDTL	в программе на Perl	в описании задачи на CDTL	в программе на Perl	Метрика Холстеда	Простая интуитивная модель
52	335	29600	105064	9,87	35,02	71,8	86,7

Конечно возникает такой вопрос, а что будет, если программные модули, примитивы или сам транслятор будут написаны с большим количеством ошибок? Я уверена, что даже такая ситуация будет со временем решена, поскольку исходные коды будут добавляться в банк алгоритмов, который будет находиться в открытом доступе. Следовательно, этими модулями смогут пользоваться все желающие и сообщать о найденных ошибках разработчику. И так ошибки будут сокращены до минимума.

4.6 Выводы по главе

1. Разработан прототип транслятора CDTL=>Perl. Его структура представлена на рис. 4.1 В результате работы транслятора CDTL=>Perl создано ПО для защиты от атак Web-сервера [123]. Программное обеспечение для защиты Web-сервера может быть сгенерировано рядовым пользователем. Виды атак, с которыми работает данное ПО легко могут быть настроены пользователем в режиме диалога с программой. В данном ПО, в том числе, реализованы технологии снижения числа программных ошибок в процессе создания кода и алгоритм динамического выбора состава модулей для решения коллектива задач одного типа описанные в главе 2.
2. Эффективность усовершенствованной формулы определения контекста слова и файла удалось подтвердить экспериментально. Была выбрана оптимальная весовая функция и ширина площадки. Данная техника может найти применение при анализе смысла текстов, и для анализа типа отзывов на модули, находящиеся в базе данных алгоритмов.
3. В результате проверки сложности разработки кода и количества ошибок в нём было выяснено, что при использовании CDTL эти показатели существенно сокращаются. Для расчёта сокращения сложности написания программ на CDTL по сравнению с языком Perl в процентах использовались специальные метрики: Метрика Холстеда (сокращение на 77,5%) и метрика Джилба (сокращение на 97,39%).
4. Было рассчитано сокращение ошибок при использовании CDTL. Для этого также использовались две специально подобранные метрики. Это метрика Холстеда (сокращение на 71,8%) и простая интуитивная модель (сокращение на 86,7%). Снижения числа ошибок при использовании CDTL происходит в основном за счёт снижения трудоёмкости создания и уменьшения объёма текста программы.

Заключение

Цель работы - разработка математического и программного обеспечения динамически проектируемого транслятора со специального метаязыка описания задачи на язык высокого уровня для снижения трудоемкости генерации кода.

В процессе выполнения диссертационного исследования получены следующие основные результаты:

1. Разработан синтаксис метаязыка описания задачи, обеспечивающий снижение трудоёмкости генерации кода на языке высокого уровня.

2. Разработан алгоритм динамического выбора состава модулей для решения коллектива задач одного типа, обеспечивающий рациональный состав модулей.

3. Предложен и проверен алгоритм идентификации формализованных результатов внешней верификации программных модулей в базах данных, позволяющий в более чем 95% случаев получить корректный результат, при котором не требуется сложных вычислений и больших затрат ресурсов.

4. Создан прототип динамического транслятора описания задачи в язык Perl, обеспечивающий проектирование программы решения задачи на языке высокого уровня.

5. Разработанное программное обеспечение защищено патентом РФ, получено 11 свидетельств о государственной регистрации в Роспатенте.

В приложениях представлены: патент на изобретение, 11 свидетельств о регистрации программ для ЭВМ и акт о внедрении результатов диссертации.

Список литературы

1. Доренская Е. А. О технологии программирования, ориентированной на минимизацию программных ошибок / Е.А. Доренская, Ю.А. Семенов // Современные информационные технологии и ИТ-образование. – 2017. –Т. 13. – № 2. – Стр. 50-56, DOI: https://doi.org/10.25559/SITITO.2017.2.226
2. Доренская Е. А. Метод определения контекстных значений слов и документов / Е.А. Доренская, Ю.А. Семенов // Современные информационные технологии и ИТ-образование. – 2018. – Т. 14. – №4. – Стр. 896-902, DOI: https://doi.org/10.25559/SITITO.14.201804.896-902
3. Доренская Е. А. Улучшенный алгоритм вычисления контекстного значения слов в тексте/ Доренская Е. А., Семенов Ю.А. // Современные информационные технологии и ИТ-образование. – 2019. – Т. 15. – №4. – Стр. 954-960, https://doi.org/10.25559/SITITO.15.201904.954-960
4. Доренская Е. А. Язык описания проблемы и исследование его возможностей/ Е.А. Доренская, Ю.А. Семенов, А.А. Куликовская // Современные информационные технологии и ИТ-образование. – 2020. – Т. 16 – №3. – Стр. 653-663, DOI: https://doi.org/10.25559/SITITO.16.202003.653-663
5. Куликовская А. А. Количественные характеристики безопасности программ / А. А. Куликовская, Е.А. Доренская, Ю.А. Семенов // Современные информационные технологии и ИТ-образование. – 2022. – Т. 18. – №4. – Стр. 855-860, DOI: https://doi.org/10.25559/SITITO.18.202204.855-860
6. Dorenskaya E.A., New methods of minimizing the errors in the software / E.A. Dorenskaya, Y.A. Semenov // Proceedings of the VIII International Conference «Distributed Computing and Grid-technologies in Science and Education» (GRID 2018), Dubna, Moscow region, Russia. – 2018. – Vol 2267, Pp. 150-154 , URL: http://ceur-ws.org/Vol-2267/150-154-paper-27.pdf

7. Доренская Е. А., Семенов Ю.А. Способ определения контекста слова и текстового файла. Патент на изобретение № 2685044 от 16.04.2019. М.: ФИПС, 2019.
8. Доренская Е. А. Формирование регулярного выражения посредством диалога. – Свидетельство о государственной регистрации программы для ЭВМ № 2017619074 от 15.08.2017. М: ФИПС, 2017.
9. Доренская Е. А. Организация запросов для генерации регулярных выражений с помощью специальных слов. – Свидетельство о государственной регистрации программы для ЭВМ № 2017663254 от 28.11.2017. М: ФИПС, 2017.
10. Доренская Е. А. Автоматическая генерация программ для построения графиков с помощью библиотеки gnuplot. – Свидетельство о государственной регистрации программы для ЭВМ № 2017663858 от 13.12.2017. М: ФИПС, 2017.
11. Доренская Е. А. Расчёт контекстного значения заданного слова в текстовом файле. – Свидетельство о государственной регистрации программы для ЭВМ № 2018615758 от 16.05.2018. М: ФИПС, 2018.
12. Доренская Е. А. Программа для защиты от перегрева ядер процессора V 1.0. – Свидетельство о государственной регистрации программы для ЭВМ № 2019616336 от 22.05.2019. М: ФИПС, 2019.
13. Доренская Е. А. “Wordcontext” v1.0 для расчета контекстного значения слова с учетом расстояний. – Свидетельство о государственной регистрации программы для ЭВМ № 2019660440 от 06.08.2019. М: ФИПС, 2019.
14. Доренская Е. А. Программа “Loader v 1.0” для загрузки описаний в базу данных алгоритмов. – Свидетельство о государственной регистрации программы для ЭВМ № 2019663880 от 24.10.2019. М: ФИПС, 2019.
15. Доренская Е. А. Универсальный конфигуратор ПО для пакета программ защиты Web-сервера. – Свидетельство о государственной регистрации программы для ЭВМ № 2020616122 от 10.06.2020. М: ФИПС, 2020.

16. Доренская Е. А. Анализ описаний PDL и трансляция с помощью примитивов. – Свидетельство о государственной регистрации программы для ЭВМ № 2020662902 от 20.10.2020. М: ФИПС, 2020.
17. Доренская Е. А. Программа “Databcont1” определение контекста слов с помощью специальной базы данных. – Свидетельство о государственной регистрации программы для ЭВМ № 2021666997 от 22.10.2021. М: ФИПС, 2021.
18. Доренская Е. А. Программа оценки качества программных модулей банка алгоритмов. – Свидетельство о государственной регистрации программы для ЭВМ, № 2023619401 от 11.05.2023. М: ФИПС, 2023.
19. Доренская Е. А. Методы создания изображений и программ с использованием описаний и функций // Решение: матер. 11-й Всеросс. НПК. – Пермь: Изд-во Перм. нац. исслед. политехн. ун-та. – 2022. – стр. 206-209.
20. Доренская Е. А. Подход к минимизации количества программных ошибок в алгоритмических модулях, разработка «языка описания проблем» / Е. А. Доренская, А.В. Бычков // Труды 60-й Всероссийской научной конференции МФТИ Т. Нано-, био-, информационные, когнитивные и социогуманитарные науки и технологии. – 2017. – стр. 7-8. URL: https://abitu.net/public/admin/mipt-conference/INBIKST.pdf
21. Канер Сэм, Фолк Джек, Нгуен Енг Кек, Тестирование программного обеспечения // издательство «Диа Софт». – 2001. – 543 с.
22. Чуканов С. Н., Рефакторинг и технологии управления программным кодом. Серия внутривузовских методических указаний // Издательско – полиграфический комплекс СибАДИ. – 2018. – 29 с..
23. Макаров А. Н. Поиск программных ошибок в алгоритмах обработки сложно-структурированных данных // Прикладная дискретная математика. – 2009 Приложение № 1. – стр. 75-76 URL: http://www.mathnet.ru/links/de515cefa7fcb318be68dc2032f44c82/pdm113.pdf

24. J.A. Rawlinson, Report on the Therac-25, OCTRF//OCI Physicists Meeting, Kingston, Ont., Canada. – 1987, URL: http://sunnyday.mit.edu/papers/therac.pdf
25. Ниязова Ж. М., Цыб А.Ф., Гулидов И.А., Глазырин А.М., О культуре безопасности в радиологических центрах страны // Альманах клинической медицины. – 2006. – № 12. – стр. 91 URL: https://cyberleninka.ru/article/n/o-kulture-bezopasnosti-v-radiologicheskih-tsentrah-strany/viewer
26. Investigation of an accidental exposure of radiotherapy patients in Panama, Report of a Team of Experts, 26 May–1 June 2001 [электронный ресурс] URL: https://www-pub.iaea.org/MTCD/publications/PDF/Pub1114_scr.pdf (дата обращения: 27.12.2022).
27. Mars Climate Orbiter Mishap Investigation Board Phase I Report, November 10, 1999 [электронный ресурс] URL: http://sunnyday.mit.edu/accidents/MCO_report.pdf (дата обращения: 27.12.2022).
28. Mars Polar Lander [электронный ресурс] URL: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1999-001A (дата обращения: 27.12.2022).
29. Le Lann, Gérard, An Analysis of the Ariane 5 Flight 501 Failure – A System Engineering Perspective // Proceedings of the 1997 international conference on Engineering of computer-based systems (ECBS'97). IEEE Computer Society, Chesnay cedex, France. – 1997. – Pp 339-346. DOI: https://doi.org/10.1109/ECBS.1997.581900
30. Верхотуров Д. Н., Ядерная война: уничтожить друг друга // Selfpub.ru, 2016
31. Charles B. Perrow, Normal Accidents: Living with High Risk Technologies - Updated Edition // Princeton University Press, Princeton. – 1999. – 386 p.
32. Robert N. Charette, This Car Runs on Code [электронный ресурс] URL: https://spectrum.ieee.org/transportation/systems/this-car-runs-on-code (дата обращения: 30.12.2022)

33. US Department of Transportation, National Highway Traffic Safety Administration, Report of Investigation PE 16-007, [электронный ресурс]. – URL: https://static.nhtsa.gov/odi/inv/2016/INCLA-PE16007-7876.pdf
34. Knight Capital Says Trading Glitch Cost It $440 Million [электронный ресурс] URL: https://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/ (дата обращения: 27.12.2022).
35. Иванников В. П., Системное программирование: состояние и тенденции // Сборник докладов Международной научной конференции, посвященной 80-летию со дня рождения академика В. А. Мельникова. – 2009. – с. 3-6. URL: http://www.besm-6.su/documents/%D0%9C%D0%9D%D0%9A80%D0%92%D0%90%D0%9C%20%D0%A1%D0%B1%D0%BE%D1%80%D0%BD%D0%B8%D0%BA%20%D0%B4%D0%BE%D0%BA%D0%BB%D0%B0%D0%B4%D0%BE%D0%B2.pdf
36. Отчёт о проверке безопасности открытого и проприетарного кода за 2011-ый год [электронный ресурс] URL: https://www.linux.org.ru/news/security/7461868 (дата обращения: 27.12.2022).
37. Holzmann G.J., The power of 10: rules for developing safety-critical code // Computer. – 2006. – Vol. 39. – № 9. – Pp 95 – 99, DOI: https://doi.org/10.1109/MC.2006.212
38. Иванников В. П., Белеванцев А. А., Бородин А. Е., Игнатьев В. Н., Журихин Д. М., Аветисян А. И., Леонов М. И. Статический анализатор Svace для поиска дефектов в исходном коде программ // Труды Института системного программирования РАН. – 2014. – Т. 26. – № 1. – стр. 231-250 DOI: https://doi.org/10.15514/ISPRAS-2014-26(1)-7
39. Герасимов А. Ю., Обзор подходов к улучшению качества результатов статического анализа программ // Труды Института системного программирования РАН. – 2017. – Т. 29. – № 3. – стр. 75-98 DOI: https://doi.org/10.15514/ISPRAS-2017-29(3)-6

40. Бородин А. Е., Горемыкин А. В., Вартанов С. П., Белеванцев А. А., Поиск уязвимостей небезопасного использования помеченных данных в статическом анализаторе Svace // Труды Института системного программирования РАН. – 2021. – Т. 33. – № 1. – стр. 7-31, DOI: https://doi.org/10.15514/ISPRAS-2021-33(1)-1
41. Меньшиков М.А., Подходы к отладке и обеспечению качества статического анализатора // Труды Института системного программирования РАН. – 2020. – Т. 32. – № 3. – стр. 33-47, DOI: https://doi.org/10.15514/ISPRAS-2020-32(3)-3
42. Беляев М.В., Романенков Е.С., Игнатьев Н.В, Моделирование библиотечных функций в промышленном статическом анализаторе кода // Труды Института системного программирования РАН. – 2020. – Т. 32. – № 3. – стр. 21-31, DOI: https://doi.org/10.15514/ISPRAS-2020-32(3)-2
43. Dimovski, A.S., Apel, S., Legay, A., Several lifted abstract domains for static analysis of numerical program families // Science of Computer Programming. – 2022. – Vol 213. – №1, DOI: https://doi.org/10.1016/j.scico.2021.102725
44. Chen, T., Heo, K., Raghothaman, M., Boosting static analysis accuracy with instrumented test executions // 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE, USA. – 2021. – Pp 1154–1165 DOI: https://doi.org/10.1145/3468264.3468626
45. Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow, DeepCoder: Learning to Write Programs //Proceedings of ICLR. – 2017, URL: https://arxiv.org/abs/1611.01989
46. Бутенко В. В., Бутенко Д. С., Механизмы отладки кода // Актуальные вопросы технических наук: материалы III Междунар. науч. конф., Пермь: Зебра. – 2015. – стр. 12-14, URL: https://moluch.ru/conf/tech/archive/125/7395/
47. Debugging (Unmanaged API Reference) [электронный ресурс] URL: https://docs.microsoft.com/en-us/dotnet/framework/unmanaged-api/debugging/ (дата обращения: 27.12.2022)

48. Бородин А.Е., Белеванцев А.А., Статический анализатор Svace как коллекция анализаторов разных уровней сложности // Труды ИСП РАН. – 2015. – Т. 27. – № 6, , с. 111-134 DOI: https://doi.org/10.15514/ISPRAS-2015-27(6)-8
49. Сайт института системного программирования им. В.П. Иванникова РАН [электронный ресурс] URL: https://www.ispras.ru/ (дата обращения: 29.12.2022)
50. Плаксин М. А., Тестирование и отладка программ для профессионалов будущих и настоящих 2-е издание (электронное) // БИНОМ. Лаборатория знаний. – 2013. – 166 с.
51. Mel Llaguno, 2017 Coverity Scan Report [Электронный ресурс]. – Режим доступа: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-software-scan-report.html (дата обращения: 30.12.2022).
52. The Linux Foundation [электронный ресурс] URL: https://www.linuxfoundation.org/projects/linux/ (дата обращения: 27.12.2022).
53. Проекты с открытым исходным кодом, поддерживаемые компанией Linux [электронный ресурс] URL: https://www.linuxfoundation.org/projects/ (дата обращения: 30.12.2022)
54. Благодатских В.А., Волнин В.А., Поскакалов К.Ф., Стандартизация разработки программных средств // Финансы и статистика. – 2005. – 288 с.
55. Вичугова А. А., Автоматизация процесса разработки программного обеспечения: методы и средства // Прикладная информатика. – 2016. – Т. 11. – № 3. – стр. 63-75, URL: http://earchive.tpu.ru/bitstream/11683/37406/1/reprint-nw-15418.pdf
56. Веб-сайт Machine Learning Lab, University of Freiburg [электронный ресурс] URL: https://www.automl.org/ (дата обращения: 30.12.2022)
57. Большакова Е.И., Воронцов К.В., Ефремова Н.Э., Клышинский Э.С., Лукашевич Н.В., Сапин А.С. Автоматическая обработка текстов на естественном языке и анализ данных // учебное пособие НИУ ВШЭ. – 2017.

– 269 стр. – URL: https://www.hse.ru/data/2017/08/12/1174382135/NLP_and_DA.pdf

58. Бермудес С.Х.Г. Керимова С.У. О методе определения текстовой близости основанном на семантических классах // Инженерный вестник Дона, Ростов-на-Дону. – № 4 (43) . – 2016, URL: https://cyberleninka.ru/article/n/o-metode-opredeleniya-tekstovoy-blizosti-osnovannom-na-semanticheskih-klassah/viewer
59. The New Hacker's Dictionary version 4.2.2 by Various editors, 2002, URL: https://www.fulltextarchive.com/pdfs/The-New-Hacker-s-Dictionary-version-4-2.pdf
60. Розенталь Д. Э., Теленкова М. А., Словарь-справочник лингвистических терминов // Просвещение.– 1976. – 543 стр.
61. Чефранова М. А., Роль полисемии в современном русском языке и речи // Молодой ученый. – 2014. – №4. – стр. 1132-1134, URL: https://moluch.ru/archive/63/10077/
62. Розенталь Д.Э., Голуб И.Б., Теленкова М.А., Современный русский язык // Высшая школа. – 1991. – 559 с.
63. Турдаков Д. Ю. Методы и программные средства разрешения лексической многозначности терминов на основе сетей документов // диссертация на соискание уч. степени кандидата физико-математических наук, МГУ, 2010, 138 стр. URL: http://www.ispras.ru/upload/iblock/3ea/3ea4b70757395519f5799222c1189fe9.pdf
64. Кондрашова Д. А. Насыров Р. В., Сравнение эффективности методов автоматической классификации текстов // Труды VII Всероссийской научной конференции (с приглашением зарубежных ученых) Т.. 1, ГОУ ВПО "Уфимский государственный авиационный технический университет". – 2019. – стр. 146-149 URL: http://itids.ugatu.su/index.php/itids/itids2019/paper/viewFile/159/133
65. Усталов Д.А. Модели, методы и алгоритмы построения семантической сети слов для задач обработки естественного языка // диссертация на соискание

уч. степени кандидата физико-математических наук, Институт математики и механики им. Н. Н. Красовского УО РАН. – 2017. – 129 стр., URL: http://www.susu.ru/sites/default/files/dissertation/dissertation_0.pdf

66. Лесников С.В., Холмогоров Д. В., Лесников А. В., Лесников Г. С., Мозымов А. Г., Архитектура гипертекстового информационно-поискового тезауруса метаязыка науки // Современные информационные технологии и ИТ-образование. – № 7. – 2011. – стр. 696-705. – URL: https://cyberleninka.ru/article/n/arhitektura-gipertekstovogo-informatsionno-poiskovogo-tezaurusa-metayazyka-nauki/viewer
67. Roussopoulos N.D. A semantic network model of data bases // Department of Computer Science, University of Toronto. – 1976. – TR № 104
68. Tingting Wei, Yonghe Lu, Huiyou Chang, Qiang Zhou, Xianyu Bao, A semantic approach for text clustering using WordNet and lexical chains // Expert Systems with Applications. – 2015. – Vol. 42. – № 4. – Pp 2264-2275, DOI: https://doi.org/10.1016/j.eswa.2014.10.023
69. Бондарчук Д.В. Определение семантической близости термов с помощью контекстного множества // Сборник научных трудов по материалам I Международной конференции Уральский федеральный университет имени первого Президента России Б.Н. Ельцина. – 2016. – стр. 175-179 URL: http://elar.urfu.ru/bitstream/10995/43751/1/cai-2016-41.pdf
70. Зеркина Н. Н., Костина Н. Н., Этапы и направления изучения семантики история и современность // Образование и наука в современном мире. Инновации. – 2015. – №1. – стр. 70-74, URL: https://elibrary.ru/item.asp?id=25801922
71. Загидулин И., Методы и средства автоматической текстовой категоризации // Магистерская диссертация, Математико-механический факультет УрГУ, Екатеринбург. – 2008. – 65 стр.
72. Сапожников А. В., Чеканин А. И., Антонов Е. С., Способ и система предоставления контекстуальной информации. Патент на изобретение RU 2

632 126, дата публикации 02.10.2017 URL: https://www1.fips.ru/registers-doc-view/fips_servlet?DB=RUPAT&DocNumber=2632126&TypeFile=html

73. Johnny AZZI, Romeo ISSA, Walid SABA, Eddy TOUMA Techniques for understanding the aboutness of text based on semantic analysis. Патент на изобретение US 9,632,999 B2, дата публикации 25.04.2017, URL: https://patentimages.storage.googleapis.com/2b/4f/88/8a017878f75f47/US9632999.pdf
74. Бродский Ю. И., Мягков А. Н. Декларативное и императивное программирование в имитационном моделировании сложных многокомпонентных систем // вестник Московского Государственного Технического Университета им. Н.Э. Баумана. – 2012. – № 2. – стр. 178-188 URL: https://cyberleninka.ru/article/n/deklarativnoe-i-imperativnoe-programmirovanie-v-imitatsionnom-modelirovanii-slozhnyh-mnogokomponentnyh-sistem
75. Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Гладкова Е.Н., Ефимкин К.Н. Язык НОРМА // Препринты ИПМ им. М.В. Келдыша. – 2019. – № 132. – 48 с. DOI: http://doi.org/10.20948/prepr-2019-132
76. Залогова Л.А., Принципы функционального программирования // Пермский государственный национальный исследовательский университет. – 2020. – Т. 49. – № 2. – стр. 54-68, DOI: https://cyberleninka.ru/article/n/printsipy-funktsionalnogo-programmirovaniya/viewer
77. Большакова Е.И., Груздева Н. В. Основы программирования на языке ЛИСП. Учебное пособие // Издательский отдел факультета ВМК МГУ имени М.В. Ломоносова. – 2010, URL: http://www.recyclebin.ru/BMK/LISP/PosobieLisp.pdf
78. Wolfram Language & System documentation center [электронный ресурс] URL: https://reference.wolfram.com/language/ (дата обращения: 27.12.2022).
79. Язык Wolfram Language [электронный ресурс] URL: https://www.wolfram.com/language/for-experts/ (дата обращения: 27.12.2022).

80. Скрипкин С. К., Ворожцова Т. Н. Современные методы метапрограммирования и их распределенные системы технологии разработки перспективы // Вестник Иркутского государственного технического университета. – 2006. – Т. 26. – № 2-3. – стр. 90-97, URL: https://cyberleninka.ru/article/n/sovremennye-metody-metaprogrammirovaniya-i-ih-raspredelennye-sistemy-tehnologii-razrabotki-perspektivy
81. Василенко Н. В., Макаров В. А., Модели оценки надежности программного обеспечения // Вестник Новгородского государственного университета им. Ярослава Мудрого. – 2004. – № 28. – стр. 126-132, URL: https://cyberleninka.ru/article/n/modeli-otsenki-nadezhnosti-programmnogo-obespecheniya
82. Грузенкин Д. В., Чучунева А. А., Павлушкина Л. В., Методы оценки надежности программного обеспечения, Новая наука: от идеи к результату // Агентство международных исследований. – 2016. – № 12-3. – стр. 175-178, URL: https://ami.im/sbornik/MNPK-120-3.pdf
83. Монахов Ю. М., Функциональная устойчивость информационных систем, часть 1. Надежность программного обеспечения // Издательство Владимирского государственного университета. – 2011. – 60 с. , URL: https://op.vlsu.ru/fileadmin/Programmy/Specialitet/10.05.04/Metod_doc/2016/Metod_FuncustIS_100504_29122016.pdf
84. Дубовой Н. Д. Основы метрологии, стандартизации и сертификации // Издательский Дом ФОРУМ. – 2014, 255 с.
85. Аверьянов А. В., Кошель И. Н., Кузнецов В. В., Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ // «Известия высших учебных заведений. Приборостроение. – 2019. – Т. 62. – № 11. – стр. 970-975, DOI: https://doi.org/10.17586/0021-3454-2019-62-11-970-975
86. Ледовских И. Н., Метрики сложности кода // Технический отчет ИСП РАН [электронный ресурс] – 2012, URL:

https://www.ispras.ru/preprints/docs/prep_25_2013.pdf (дата обращения: 11.05.2022).

87. Варенница В. В., Проблемы вычисления метрик сложности программного обеспечения при проведении аудита безопасности кода методом ручного рецензирования // Вестник Московского государственного технического университета им. Н.Э. Баумана. Серия «Приборостроение». – 2011. – № S1. – стр. 79-84
88. Семенов Ю.А., Экономика в 2016 году и через 10 лет // Экономические стратегии. – 2017. – Т. 19. – №1. – стр. 126-135, URL: http://www.inesnet.ru/wp-content/mag_archive/2017_01/es2017-01-126-135_Yury_Semenov.pdf
89. Ильин А. В., Конструирование разрешающих структур на задачных графах системы знаний о программируемых задачах // Информационные технологии и вычислительные системы. – 2007. – №3. – стр. 30-36.
90. Ильин В. Д., Система порождения программ // Наука. – 1989. – 264 стр.
91. Тыугу Э. Х., Концептуальное программирование // Наука. – 1984. – 256 стр.
92. Ильин А. В., Ильин В. Д. Систематизация знаний о программируемых задачах // Системы и средства информатики. – 2014. – Т. 24. – № 3. – стр. 192-203.
93. Bernard A., How attackers are adapting to a post-macro world // [электронный ресурс] URL: https://www.techrepublic.com/article/how-attackers-are-adapting-to-a-post-macro-world/ (дата обращения: 27.08.2022)
94. Семенов Ю.А., Овсянников А.П., Овсянникова Т.В., Разработка банка алгоритмов и основ языка описания проблем с целью минимизации числа программных ошибок // Труды НИИСИ РАН. – 2016. – Т. 6. – №2. – стр. 96-100, URL: https://elibrary.ru/item.asp?id=29798446
95. Лебедева Т. Н., Носова Л. С., Формализация данных в языке программирования 1С // Вестник Астраханского государственного технического университета. – 2015. – №3. – стр. 113-121, URL:

https://cyberleninka.ru/article/n/formalizatsiya-dannyh-v-yazyke-programmirovaniya-1s

96. Архив документации и программного обеспечения, написанного на языке программирования Perl [электронный ресурс] URL: https://www.cpan.org/ (дата обращения: 27.12.2022).
97. Смирнов А.Е., Язык программирования PERL // Педагогический университетский вестник Алтая. – 2004. – №1, стр. 114-119 URL: https://www.elibrary.ru/item.asp?id=21453645
98. Код программы lm-sensors [электронный ресурс] URL: https://github.com/lm-sensors/lm-sensors/blob/14856e37e84be6eeb8310c939dceeb7438437079/README.thinkpad (дата обращения: 29.12.2022)
99. Сайт программы для построения графиков gnuplot [электронный ресурс] URL: http://www.gnuplot.info/ (дата обращения: 29.12.2022)
100. Семенов Ю.А. Управляющая база данных MIB [электронный ресурс] URL: http://book.itep.ru/4/44/mib44131.htm (дата обращения: 27.12.2022).
101. Сервер Сатурн ИТЭФ [электронный ресурс] URL: http://saturn.itep.ru/ (дата обращения: 27.12.2022).
102. The New Hacker's Dictionary version 4.2.2 by Various editors [электронный ресурс]. – 2002 URL: https://www.fulltextarchive.com/pdfs/The-New-Hacker-s-Dictionary-version-4-2.pdf (дата обращения: 27.12.2022).
103. Катастрофические последствия программных ошибок, блог компании mail.ru group [электронный ресурс] URL: https://habr.com/ru/company/mailru/blog/370153/ (дата обращения: 27.07.2022).
104. Сайт Scientific Linux [электронный ресурс] URL: https://scientificlinux.org/ (дата обращения: 27.12.2022).
105. Chelf B., "Measuring Software Quality - A Study of Open Source Software", Coverity [электронный ресурс]. – 2011, URL:

http://book.itep.ru/depository/security/software_security/open_source_quality_report.pdf (дата обращения: 27.12.2022).

106. Пахунов А. В., Языки программирования: классификация, особенности, критерии выбора // Современная наука. – 2015. – №4. – стр. 89-91. – URL: https://cyberleninka.ru/article/n/yazyki-programmirovaniya-klassifikatsiya-osobennosti-kriterii-vybora

107. Добрынин В.Ю., Клюев В.В., Некрестьянов И.С. Оценка тематического подобия текстовых документов // Сборник докладов Второй Всероссийской научной конференции" Электронные библиотеки: Перспективные Методы и Технологии, Электронные коллекции". – Протвино. – 2000. – стр. 204-210 URL: http://web.ihep.su/library/pubs/aconf00/dconf00/ps/069.pdf

108. Ильвовский Д.А. Модели, алгоритмы и программные комплексы обработки текстовых данных на основе решеток замкнутых описаний // диссертация на соискание уч. степени к. т. н., НИУ ВШЭ. – 2014. – 158 стр. URL: https://www.hse.ru/data/2014/09/24/1315819304/dis.pdf

109. Малахов Д.А., Серебряков В.А., Модель семантического поиска на базе тезауруса // Труды XIX Международной конференции «Аналитика и управление данными в областях с интенсивным использованием данных» (DAMDID/RCDL'2017), 10–13 октября 2017. – стр. 191-196, URL: http://ceur-ws.org/Vol-2022/paper32.pdf

110. Воронина И.Е., Кретов А.А., Попова И.В., Алгоритмы определения семантической близости ключевых слов по их окружению в тексте // Вестник ВГУ, серия: системный анализ и информационные технологии. – 2010. – №1. – стр. 148-153, URL: http://www.vestnik.vsu.ru/pdf/analiz/2010/01/2010-01-25.pdf

111. Крейнес М.Г., Модели текстов и текстовых коллекций для поиска и анализа информации // труды МФТИ. Математическое моделирование эколого-экономических систем: экономика. – 2017. – Т. 9. – №3. – стр. 132-142, URL: https://mipt.ru/upload/medialibrary/067/16_kreines_132_142.pdf

112. Семенов Ю.А. Технология blockchain [электронный ресурс] URL: http://book.itep.ru//4/6/blockchain.htm (дата обращения: 27.12.2022).
113. Прохоров Ю.В., Розанов Ю.А., Теория вероятностей. Основные понятия, предельные теоремы, случайные процессы // изд. "Наука". – 1967. – 495 стр.
114. Семенов Ю.А. SET и другие системы осуществления платежей [электронный ресурс], URL: http://book.itep.ru/4/6/set_66.htm (дата обращения: 27.12.2022).
115. Wang, J., Pang, J., Liu, X., Yue, F., Tan, J., Fu, L., Dynamic Translation Optimization Method Based on Static Pre-Translation // IEEE Access, Vol 7, 2019, Pp 21491-21501, DOI: https://doi.org/10.1109/ACCESS.2019.2897611
116. Ferrara, P., Cortesi, A., Spoto, F., From CIL to Java bytecode: Semantics-based translation for static analysis leveraging // Science of Computer Programming. – 2020. – Vol 191. – № 102392, DOI: https://doi.org/10.1016/j.scico.2020.102392
117. Zaytsev, V., Modelling of language syntax and semantics: The case of the assembler compiler // Journal of Object Technology. – 2020. – Volume 19. – № 2. – Pp 1-22, DOI: http://dx.doi.org/10.5381/jot.2020.19.2.a5
118. Стась А. Н., Методика обучения разработке трансляторов // Вестник Томского государственного педагогического университета. – 2015. – № 8. – стр. 76-81, URL: https://cyberleninka.ru/article/n/metodika-obucheniya-razrabotke-translyatorov
119. Хохлов Д.Г., Кирпичников А.П., Халид Г.Х., Интерпретатор языка С для электронного обучения программированию // Вестник Технологического университета. – 2017. – Т. 20. – № 14. – стр. 109-111, URL: https://cyberleninka.ru/article/n/interpretator-yazyka-s-dlya-elektronnogo-obucheniya-programmirovaniyu
120. Новичкова М. И., Трофимов Ю. А., Разработка транслятора языка программирования высокого уровня // Образование и наука в современных

условиях. – 2015. – № 3. – стр. 213-214, URL: https://interactive-plus.ru/e-articles/142/Action142-10642.pdf

121. Куликовская А. А., Доренская Е.А., Семенов Ю.А., Разработка банка алгоритмов и метода поиска программ в соответствии с требованиями пользователей // Современные информационные технологии и ИТ-образование. – 2020. – Т. 16. – №1. – стр. 81-89, DOI: https://doi.org/10.25559/SITITO.16.202001.81-89

122. Котов Э. М., Целых А. Н., Исследование моделей информационного поиска // Известия Южного федерального университета. Серия: Технические науки. – 2009. – № 4 (93). – стр. 163-168, URL: https://cyberleninka.ru/article/n/issledovanie-modeley-informatsionnogo-poiska

123. Tobi Oetiker's MRTG - The Multi Router Traffic Grapher [электронный ресурс] URL: https://oss.oetiker.ch/mrtg/ (дата обращения: 27.12.2022).

124. Евстропов Д.Е., Добржинский Ю.В., SNMP - протокол управления и наблюдения ЛВС // Вологдинские чтения. – 2009. – № 73. – стр. 63-64, URL: https://cyberleninka.ru/article/n/snmp-protokol-upravleniya-i-nablyudeniya-lvs

125. Noller, Y.,Kersten, R., Păsăreanu, C.S., Badger: Complexity analysis with fuzzing and symbolic execution // ISSTA 2018: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. – 2018. – Pp 322–332, DOI: https://doi.org/10.1145/3213846.3213868

126. Боршевников А. Е. Сетевые атаки. Виды. Способы борьбы // Современные тенденции технических наук: материалы Международной научной конференции. – 2011. – стр. 8-13. — URL: https://moluch.ru/conf/tech/archive/5/1115

127. Семенов Ю.А. Атаки переполнения [электронный ресурс] URL: http://book.itep.ru/6/intrusion.htm#1000 (дата обращения: 27.12.2022).

128. Семенов Ю.А. Path Traversal (slash-атаки) [электронный ресурс] URL: http://book.itep.ru/6/intrusion.htm#192 (дата обращения: 27.12.2022).
129. Егоров М. А., Методика оценки безопасности программного кода корпоративных приложений // Вестник МГТУ им. Н.Э. Баумана. – № S1. – 2011. – стр. 67-78, URL: https://cyberleninka.ru/article/n/metodika-otsenki-bezopasnosti-programmnogo-koda-korporativnyh-prilozheniy
130. Гаврилов А. Г., Арзина И. Ю. Майнинг криптовалюты // Аллея Науки. – 2017. – Т. 2. – № 16. – стр. 355-361
131. Чалин. В.А. Способы обнаружения скрытых процессов, связанных с майнингом криптовалют // Материалы всероссийской научно-технической конференции “Автоматизированные системы управления и информационные технологии”. – 2018. – Т. 2. –стр. 339-343
132. Сайт веб-приложения phpmyadmin [электронный ресурс] URL: https://www.phpmyadmin.net / (дата обращения: 29.12.2022)
133. Казакова А. Е., Особенности семантики языков программирования // Вестник Московского университета. Серия 7: Философия. – 2007. – № 6. – стр. 69-75 URL: https://cyberleninka.ru/article/n/osobennosti-semantiki-yazykov-programmirovaniya
134. Наумов Р. В., Актуальные языки программирования // ACADEMY. – 2016, №1. – стр. – 49-50, URL: https://cyberleninka.ru/article/n/aktualnye-yazyki-programmirovaniya

ПРИЛОЖЕНИЕ

Свидетельства о регистрации программных продуктов, патент на изобретение и акт внедрения

РОССИЙСКАЯ ФЕДЕРАЦИЯ

ПАТЕНТ

НА ИЗОБРЕТЕНИЕ

№ 2685044

СПОСОБ ОПРЕДЕЛЕНИЯ КОНТЕКСТА СЛОВА И ТЕКСТОВОГО ФАЙЛА

Патентообладатель: ***Федеральное государственное бюджетное учреждение "Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра "Курчатовский институт" (НИЦ "Курчатовский институт"- ИТЭФ) (RU)***

Авторы: ***Доренская Елизавета Александровна (RU), Семенов Юрий Алексеевич (RU)***

Заявка № **2018124219**
Приоритет изобретения **03 июля 2018 г.**
Дата государственной регистрации в Государственном реестре изобретений Российской Федерации **16 апреля 2019 г.**
Срок действия исключительного права на изобретение истекает **03 июля 2038 г.**

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017619074

ФОРМИРОВАНИЕ РЕГУЛЯРНОГО ВЫРАЖЕНИЯ ПОСРЕДСТВОМ ДИАЛОГА

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2017615855**
Дата поступления **20 июня 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***15 августа 2017 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017663254

ОРГАНИЗАЦИЯ ЗАПРОСОВ ДЛЯ ГЕНЕРАЦИИ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ С ПОМОЩЬЮ СПЕЦИАЛЬНЫХ СЛОВ

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2017619873**
Дата поступления **03 октября 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***28 ноября 2017 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017663858

АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ПРОГРАММ ДЛЯ ПОСТРОЕНИЯ ГРАФИКОВ С ПОМОЩЬЮ БИБЛИОТЕКИ GNUPLOT

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2017660685**
Дата поступления **24 октября 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***13 декабря 2017 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018615758

РАСЧЁТ КОНТЕКСТНОГО ЗНАЧЕНИЯ ЗАДАННОГО СЛОВА В ТЕКСТОВОМ ФАЙЛЕ

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2018613237**
Дата поступления **03 апреля 2018 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***16 мая 2018 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019616336

ПРОГРАММА ДЛЯ ЗАЩИТЫ ОТ ПЕРЕГРЕВА ЯДЕР ПРОЦЕССОРА ЭВМ V 1.0

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2019614990**
Дата поступления **07 мая 2019 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***22 мая 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019660440

«Word context» v 1.0 ДЛЯ РАСЧЁТА КОНТЕКСТНОГО ЗНАЧЕНИЯ СЛОВА С УЧЁТОМ РАССТОЯНИЙ

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2019619485**
Дата поступления **22 июля 2019 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***06 августа 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019663880

Программа Loader v 1.0 для загрузки описаний в базу данных алгоритмов

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2019662734**
Дата поступления **15 октября 2019 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***24 октября 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2020616122

Универсальный конфигуратор ПО для пакета программ защиты Web-сервера

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2020615340**
Дата поступления **03 июня 2020 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***10 июня 2020 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2020662902

Анализ описаний на языке PDL и трансляция с помощью примитивов

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор: ***Доренская Елизавета Александровна (RU)***

Заявка № **2020662211**
Дата поступления **14 октября 2020 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***20 октября 2020 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021666997

Databcont V 1.0 - определение контекста слов с помощью специальной базы данных

Правообладатель: ***Федеральное государственное бюджетное учреждение «Институт теоретической и экспериментальной физики имени А.И. Алиханова Национального исследовательского центра «Курчатовский институт» (RU)***

Автор(ы): ***Доренская Елизавета Александровна (RU)***

Заявка № **2021665962**
Дата поступления **12 октября 2021 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***22 октября 2021 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2023619401

«Программа оценки качества программных модулей банка алгоритмов»

Правообладатель: ***Федеральное государственное бюджетное учреждение «Национальный исследовательский центр «Курчатовский институт» (RU)***

Автор(ы): ***Доренская Елизавета Александровна (RU)***

Заявка № **2023618428**
Дата поступления **28 апреля 2023 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***11 мая 2023 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Ю.С. Зубов

НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ ЦЕНТР
«КУРЧАТОВСКИЙ ИНСТИТУТ»

Федеральное государственное бюджетное учреждение
«Институт теоретической и экспериментальной
физики имени А.И. Алиханова Национального
исследовательского центра «Курчатовский институт»
(НИЦ «Курчатовский институт» – ИТЭФ)

ул. Большая Черемушкинская, д. 25, г. Москва, 117218
тел.: (499) 125-32-97, факс: (499) 127-08-33

15.11.2021 № 900-000/4-87

На № _____

АКТ

О внедрении результатов диссертационной работы Доренской Елизаветы Александровны, «Математические и программные методы автоматизации программирования и сокращения числа ошибок»

г. Москва 15.11.2021

Настоящим актом удостоверяется, что основные теоретические результаты и комплекс программ, представленные в диссертационной работе на соискание учёной степени кандидата технических наук Доренской Е. А. «Математические и программные методы автоматизации программирования и сокращения числа ошибок» были использованы в НИЦ "Курчатовский институт" – Институт Теоретической и Экспериментальной Физики (ИТЭФ), отдел № 242, Группа технической поддержки компьютерных линий связи в период с «17» мая 2020 г. по «17» декабря 2020 г. в рамках выполнения работ по теме «Исследование возможности автоматической генерации программ на примере защиты WEB-серверов с целью сокращения числа программных ошибок».

Использование результатов диссертации позволило повысить эффективность защиты серверов от компьютерных атак, в частности сервера saturn.itep.ru. На этом сервере находится интернет-ресурс по телекоммуникационным технологиям http://book.itep.ru (7516 страниц).

Первый заместитель директора
по научной работе
НИЦ «Курчатовский институт» - ИТЭФ _____ / В.Ю. Егорычев /