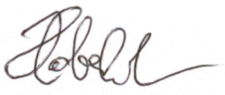
На правах рукописи



**Хаберланд Рене**

# ЛОГИЧЕСКИЙ ЯЗЫК ПРОГРАММИРОВАНИЯ КАК ИНСТРУМЕНТ СПЕЦИФИКАЦИИ И ВЕРИФИКАЦИИ ДИНАМИЧЕСКОЙ ПАМЯТИ

Специальность:
05.13.11 – Математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей
АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург — 2020

Работа выполнена на кафедре «Математического обеспечения и применения ЭВМ» государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В.И. Ульянова (Ленина)» (СПбГЭТУ «ЛЭТИ»).

| | |
|---|---|
| **Научный руководитель:** | кандидат технических наук, доцент<br>Кирилл Владимирович Кринкин |
| **Официальнные оппоненты:** | доктор физико-математических наук,<br>Андрей Николаевич Терехов<br>Профессор, заведующий кафедрой системного программирования, «Санкт-Петербургский Государственный Университет», г. Санкт-Петербург |
| | кандидат технических наук, доцент<br>Дмитрий Ильич Муромцев<br>доцент факультета программной инженерии и компьютерной техники, «Санкт-Петербургский Национальный Исследовательский Университет Информационных Технологий, Механики и Оптики» (Университет ИТМО), г. Санкт-Петербург |
| **Ведущая организация:** | Федеральное государственное бюджетное учреждение науки «Санкт-Петербургский институт информатики и автоматизации Российской академии наук» (СПИИРАН), г. Санкт-Петербург |

Защита состоится «08» апреля 2020 г. в 17:00 часов на заседании Диссертационного Совета Д212.238.01 при СПбГЭТУ «ЛЭТИ» по адресу: 197376, Россия, Санкт-Петербург, ул. Профессора Попова, дом 5 .

С диссертацией можно ознакомится в библиотеке ФГАОУ ВО «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)» и на сайте университета `http://www.etu.ru`.

Автореферат разослан «07» февраля 2020 г.

| | |
|---|---|
| Ученый секретарь<br>Диссертационного совета: | кандидат технических наук, доцент<br>Пазников Алексей Александрович |

# 1 Общая характеристика работы

## 1.1 Актуальность

Ошибки использования динамической памяти являются одними из самых дорогостоящих при разработке программного обеспечения на протяжении уже нескольких десятилетий. Локализация таких ошибок является трудной задачей, поскольку, довольно часто, видимое проявление некорректного поведения программы и реальный участок исходного кода его вызвавшего отстоят далеко друг от друга. Следовательно, разработка подходов и методов, упрощающих поиск и предотвращение ситуаций некорректного использования динамической памяти является актуальной задачей.

Выразимость утверждений, полнота правил, и автоматизация в большой степени определяют успех верификации динамической памяти. Выразимость утверждений касается формализма утверждений и правил верификации. В существующей на текущий момент практике, описания динамической памяти сложны и часто неинтуитивны. Возможности логического вывода зачастую ограничиваются корректностью и полнотой правил верификации и представляют собой лишь «*островные реализации*», которые нельзя использовать на практике. Из-за большого количества соглашений(конвенций) в языках спецификации и верификации возможности верификации динамической памяти сильно ограничены.

Верификация часто осуществляется нетривиально, объяснения принятия или отказа доказательства, как правило, контр-интуитивны.

Диссертационная работа опирается на фундаметальные и прикладные исследования таких ученых как: Floyd R.W., Hoare C.A.R., Burstall R.M., Kowalski R.A., Clarke E.M. Jr., Apt K.R., Suzuki N., Warren D.H.D., Horwitz S., Miller B.P., Fredriksen, L. So, B., Steinbach J., Tofte M., Talpin J.-P., Abadi M., Hutton G., Reynolds J.C., Berdine J., Calcagno C., O'Hearn P.W. Clarke E.M. Jr., Parkinson M.J., Burstall R.M., Hurlin C., Distefano D., Birkedal L., Matthews C., Bertot Y., Gregoire B., Leroy X., Dodds M., Jacobs B., Smans J., Philippaerts P., Vogels F., Penninck W., Piessens F. Bornat R., Meyer B., Parduhn S., Wilhelm R. Leino K.R.M.

## 1.2 Цель работы

Повышение степени выразимости средств описания вариантов использования динамической памяти за счёт исключения многозначности между языками спецификации и верификации, а также разработка программных средств автоматической верификации.

Работа была осуществлена при поддержке программы №2.136.2014/К от министерства образования и науки Российской Федерации.

## 1.3 Задачи исследования

Для достижения поставленной цели в диссертационной работе решаются следующие задачи:

1 Исследование и анализ существующих ограничений выразимости и выявление причин несоответствия языков спецификации и верификации динамической памяти.

2 Исследование декларативности в утверждениях, которые основываются на предикатной логике без избыточных конвенций и анализ реляционной модели, основанной на ужесточении пространственных операторов и абстрактных предикатов, для описания куч (от англ. *Heap*; в работе используется русское название структуры данных „куча“, предложенной Дж. Вильямсом ( J. W. J. Williams) в 1964). Сравнение реляционной модели предикатов с другими моделями.

3 Разработка архитектуры верификатора динамической памяти и его программная реализация.

4 Исследование критериев реализуемости верификации динамической памяти на основе выбранного логического языка программирования (использующего абстрактную машину Уоррена) и автоматизированного вывода над абстрактными предикатами. Исследование возможностей и ограничений при представлении куч, а также правил верификации с помощью термов.

5 Исследование представления куч в таком унифицированном виде, который даёт возможность обнаружить синтаксически различные, но семантически одинаковые представления и свести число необходимых сравнений к минимуму. Исследование возможности отделения выражений куч от остальных логических правил и возможность подключения «$SAT$»-решателей для решения отдельных теорий.

6 Исследование возможности автоматического сравнения динамической памяти с имеющимся экземпляром памяти при условии изменения входных условий или представления модели памяти.

## 1.4 Методы исследования

Автоматизированное доказательство теорем, вычисление Хора, формальная логика, вычислимые логики, семантики программ, теория типов, теория объектов, логическое программирование, функциональное программирование, объектно-ориентированное программирование, формальные языки, теория языков программирования, теория программирования, теория графов, $\lambda$-вычисления, абстрактная алгебра, теория вычислимости, языковые процессоры, теория компиляции, логика распределенной памяти, архитектуры процессоров ЭВМ, организация памяти процессов ОС, встроенные системы, современные технологии моделирования, представление и обработка знаний.

## 1.5 Научная новизна работы

### 1.5.1 В области автоматизации доказательств:

1. Впервые исследован разрыв между языками спецификации и верификации динамической памяти. Впервые предложен логический язык программирования как инструмент, который способен преодолеть все выявленные проблемы выразимости, автоматизации и полноты с помощью унификации языков.

2. Впервые проведены сравнения выразимости при трансформации термов в логическом и функциональном представлениях, выявлено с помощью множества примеров при использовании метрик, что логический язык практически без исключений общности превосходит представления функционалов. Альтернативно к функционалам рассматривались императивные программные операторы с побочными эффектами. Оба случаи встречаются в имеющихся подходах от Reynolds, Parkinson, Berdine, Hurlin, Jacobs, Tofte, Hutton и Bertot, они приводили к разрыву языков спецификации и верификации.

3. Впервые предложено, термы диалекта языка Пролог использовать непосредственно для представления данных полного конвейера статического анализа динамической памяти, в отличие от используемого в настоящее время комбинированного под- хода Meyer, Leroy, либо подхода основанного на 3-адресном представление от Reynolds/Berdine, O'Hearn, Bornat, либо подходы основаны на ассемблере как у Parkinson.

4. Предложен новый метод автоматизации процесса верификации памяти, как синтаксического перебора абстрактных предикатов, при использовании генеричных интерпретаторов основанных на пошаговой обработке граней графа кучи, в отличие от (полу-)ручного преобразования структур куч как у Berdine, O'Hearn, а также в отличие от введения определений узкого круга тактик верификации Bertot и Jacobs.

### 1.5.2 В области выразимости языков спецификации и верификации:

1. Сняты ограничения выразимости переменных символов, термов и рекурсивных предикатов, удовлетворяющих правилам Хора для спецификации и верификации динамической памяти, в отличие от Reynolds, Berdine, Bornat и Parkinson/Hurlin. Ограничения связаны, например, с использованием как императивной переменной вместо логического символома, а также всвязи с логическим представлением правил и его выводом как доказательством теоремы над динамической памятью. Кроме того, сняты ограничения вывода встроенных предикатов, таким образом, могут произвольно анализироваться определённые абстрактные предикаты во время запуска программы.

2. Повышена выразимость утверждений в отношении структуры куч за счёт ужесточения операций над кучами, использовавшиеся ранее и имеющие неоднозначные трактовки в процессе доказательства, в отличие например от Reynolds, Jones/Hosking, Cormen/Leiserson, Atallah, Khedker, Muchnik и Pavlu.

   Впервые установлены свойства моноида и группы, позволяющие производить вычисления над кучами одним проходом и без анализа потенциально всех под-выражений динамической памяти, в отличие от Suzuki, Leino и Berdine, а также в отличие от Abadi и Cardelli, которые не вводят указателей и чьи вычисления строятся на мало интуитивных алгебраических кольцах с недостатком неполноты и проблемой локальности. Предложено расширить «*UML/OCL*» с указателями с ужесточённой моделью.

3. Достигнуто повышение выразимости куч и полноты правил, за счёт введения частичного оператора «_», которое заменяет переменную или любую часть терма кучи, в отличие от интерпретируемых предикатов «$\underline{false}$» и «$\underline{true}$» от Reynolds, Berdine/O"Hearn, а также в отличие от предложенных изменений входного языка Clarke, Apt и Tofte/Talpin.

## 1.6 Теоретическая значимость работы

1 Разработана обобщенная архитектура верификатора динамической памяти на основе конвейера, которая может быть использована и подключена в существующие анализы на основе Пролог-термов как центральное промежуточное представление.
2 Верификация динамической памяти на основе логического языка программирования является более адекватным представлением.
3 Абстрактные предикаты могут распознаваться автоматически, т.е. с помощью обобщенного подхода и без применения тактик.
4 Предложено определение единичной кучи для сокращения многозначности. Предложено вычисление для лучшего сравнения куч, а также для улучшения качества программы на более ранней стадии для возможного включения в моделирования с UML.
5 Сводимость языков верификации и спецификации динамической памяти.

## 1.7 Практическая значимость работы

При автоматизированной поддержке вывода с повышенной выразимостью теоретически обоснованный и разработанный прототип на основе диалекта Пролога позволяет проводить верификацию проще и короче.

С практической точки зрения, добавление новых языковых возможностей языка программирования приводит к новым проблемам между спецификацией и верификацией куч. таким образом: (1) необходимо проводить спецификации исключительно на логическом/декларативном языке без побочных эффектов, например близком к Прологу; (2) любое представление на языке спецификации должно обрабатываться элементами языка верификации, какова бы ни была система логического вывода; (3) языки спецификации

и верификации имеют сильные пересечения, и поэтому унификация обоих языков упрощает оба процесса.

Разработанные в рамках диссертации решения (платформа для верификации, далее – Платформа) разрешают добавлять новые фазы по обработке динамической памяти, менять существующие и переводить данные Си-программы в термовое представление. В качестве входного языка может быть использован любой другой язык, в том числе формально пустой язык. К термовому представлению, которое также может меняться, могут добавляться новые элементы. Платформа, предложенная на основе термового представления является открытой.

Теории о кучах можно добавлять произвольно, в том случае если они позволяют синтаксический перебор. Теории не о кучах также могут быть включены в SAT-решатели.

Предложенное ужесточение куч позволяет решить проблему полноты на основе «неполного определения синтаксиса», сравнивать кучи и сокращать их описания. Правила состоят из троек Хора, которые определяют кучи. Если упростить сравнение соседних куч таким образом, что анализируются только переходные кучи, то правила в общем случае упрощаются. Проверка спецификации также даёт возможность сравнивать входную и минимальную программы за счёт выявленного графа кучи. Если автоматизированное доказательство данной теоремы о куче осуществить не удаётся, то унификация термов Пролога автоматически выявит максимально обобщенный контр-пример без дополнительных затрат. Ужесточение позволяет последовательно анализировать подкучи и соответствующие подформулы без возврата и нового поиска. Отпадает анализ всех конъюнкций. Если данный набор абстрактных предикатов перебирается распознавателям LL(k) или LR(k) и т. д., то имеется (положительное или отрицательное) доказательство. Практическим компромиссом считается переоформление правил, которое допускается ради универсальности подхода.

Разработанный диалект Пролога позволяет повысить выразимость увтерждений куч, автоматизировать абстрактные предикаты и упростить спецификацию куч, что в конечном итоге повысит эффектиность использования разработанного программного средства при решении практических задач.

## 1.8 Степень достоверности результатов

Достоверности результатов обеспечивается логическими, формальными выводами и доказательствами представленных в диссертационной работе теорем, а также выполненной реализацией прототипа. Качество предложенных решений, в том числе универсальных, подтверждается проведённой тщательной экспертизой сравнимости термов на более чем 80 специально подобранных примеров из большого числа типичных заданий.

Кроме использованных формальных методов, корректность логического вывода основывается на семантическом анализе, который исключает возможность недопустимых синтаксических и семантических ошибок при предложенной в работе архитектуре конвейера.

Достоверность основных результатов, полученных в диссертационной работе, подтверждается также их апробацией на различных международных и российских конференциях.

## 1.9 Реализация результатов работы

На основе полученных в диссертационной работе теоретических результатов реализованы прототипы на Прологе с мультипарадигмальным расширением, позволяющие проводить верификацию на основе синтаксического анализа. Для анализа и сравнения существующих подходов реализованы ПО «*shrinker*» для локализации ошибок и ПО «*builder*» вместо ПО «*make*».

Результаты диссертационной работы использованы при разработке учебно-методических материалов по $\lambda$-вычислениям и введению в систему верификации Coq «*Верифика-*

*ция систем программного обеспечения*» на кафедре МО ЭВМ СПбГЭТУ.

### 1.10 Положения выносимые на защиту

1 Диалект языка Пролог, как инструмент для спецификации и верификации динамической памяти.
2 Метод верификации абстрактных предикатов куч на основе распознавания атрибутной транслирующей грамматики.
3 Подход к устранению многозначности описания куч для упрощения их анализа и верификации.
4 Комплекс программных средств для верификации, динамической памяти (Builder, Shrinker, ProLogika).

### 1.11 Апробация работы

Основные результаты работы докладывались научных конференциях докладывались и обсуждались на следующих конференциях:

1 International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP), (Venice, Italy, 2016)
2 18th Conference of Open Innovations (FRUCT), (Saint-Petersburg, Russia, 2016)
3 $EMC^2$ «Технологии Майкрософт в Теории и на Практике Программирования: Новые подходы к разработке программного обеспечения», (Saint-Petersburg, Russia, 2014)
4 Dynamically Allocated Memory Verification in Object-Oriented Programs using Prolog (SYRCoSE), (Saint-Petersburg, Russia, 2014)
5 Advances in Methods of Information and Communication Technology, (Helsinki, Finland / Petrozavodsk, Russia, 2008)
6 A Stricter Heap Separating Points-To Logic, (Moscow, Russia, 2016)
7 International Conference on Control Processes and Stability, (Saint-Petersburg, Russia, 2008)
8 Санкт-Петербургский Электротехнический Университет «ЛЭТИ» (ППС ЛЭТИ), (Saint-Petersburg, 2015)

### 1.12 Публикации

По теме диссертационной работы опубликовано 17 печатных работ, в том числе 2 статьи в рецензируемых журналах, рекомендованных ВАК, 8 индексируемых статей, получены 4 свидетельства о регистрации программы для ЭВМ, 3 работы в международных научно-технических конференциях и других изданиях.

### 1.13 Структура и объем диссертации

Диссертационная работа состоит из введения, пятери глав, заключения, библиографического списка и приложений. Основная часть работы изложена на 267 страницах и содержит 97 рисунков.

## 2 Содержание работы

**Во введении** (первая глава) для построения таксономии проблем и подходов к верификации динамической памяти предлагается «лестница качества»: Корректность $<$ Полнота $<$ Оптимальность. Полнота включает в себя автоматизацию и тотальность. Оптимальность использования ресурсов включает в себя например быстродействие, безопасность и т.д.

Аксиоматическая семантика, основанная на правилах формой $A \Rightarrow B$, вводится как основа вычисления Хора, чьи компоненты содержат пред- и постусловие $P$ и $Q$, а также программный оператор $C$ некоторого императивного языка программирования. Эти

три компоненты называются «*тройка Хора*». Минимальный объем логических правил включает правила «*усиления предусловия*» (STRONGPE) и «*ослабление постусловия*» (WEAKPOST). Минимальный объем операторных правил включает правила «*присвоения*» (ASN), «*цикла*» (LOOP) и «*оператор последовательности*» (SEQ). Инвариант цикла является единой формулой условия, которая содержит все переменные меняющиеся в теле цикла и которая на протяжении выполнения верна. Тройка Хора записывается как $\{P\}C\{Q\}$, и если не гарантировано совершение $C$, то тройка является импликацией $\{P\}C\{Q\} \to (C \neq \bot) \wedge Q$.

Задача верификации заключается в том, чтобы проверить, согласно данному набору правил, выводим ли результат лишь при использовании данных логических правил и аксиом $\models_B$. Структура вывода описывается деревом. Центральная проблема вывода, это эффективный поиск дерева вывода. Главный вопрос заключается в том, как лучше представить все компоненты троек Хора, правила вывода и как лучше подобрать входной императивный язык программирования для того, чтобы верификация была корректной и полной. Далее, сходимость является важным свойством, когда необходимо гарантировать, что верификация продолжит вывод. Обсуждается, что корректность и полнота в зависимости от набора правил или входного языка программирования, часто исключают друг друга.

Любое утверждение в вычислении Хора является логическим предикатом. Предикат решает данное утверждение об состоянии вычисления верное или нет. Представляются три основные принципа вывода в классической логике: дедукция, индукция и абдукция.

Вычисление Хора имеет пределы. Статические и глобальные переменные, со-процедуры, продолжения и ограничения в связи с параметрами приводятся в качестве примеров. В некоторых работах верификация динамически выделенных переменных оценивается как крайне трудная. Проводится анализ существующих ограничений, их теоретическая и практическая роль, а также анализируется система обозначений. Оценивается важность включения описаний в более универсальный императивный язык программирования, который близок к Си со структурами/объектными экземплярами.

Устанавливаются главные критерии: (1) простое состояние и трансформации программы должны сопоставляться простым описанием, (2) простые доказательства должны простым образом доказываться, простые отрицательные доказательства должны простым образом отрицаться, (3) «островные решения» часто не такие хорошие, как кажется на первый взгляд, а наоборот, усложняют и ограничивают более эффективные подходы. Многие авторы в различных ситуациях и по различным причинам замечают, что выразимость – это одна из наиболее сильных преград.

«*Coq*», «*verifast*» и другие ассистенты доказательств используют ряд тактик для ускорения нахождения доказательства. В качестве типов спецификации ассистенты основаны на типизированных лямбда-вычислениях второго порядка $T^{\to}_{\lambda 2}$. Бóльшая гибкость ожидается, если далее типы параметризовать, т. е. предложить $T^{\to}_{\lambda 3}$. Проблема отсутствия автоматизированного свертывания и развертывания индуктивных определений при доказательстве кажется существенной преградой. Решению этой проблемы посвящена часть данной работы.

Логическая формула в предикатной логике первого порядка определятся как:

$$\Phi ::= \underline{true} \mid \underline{false} \mid x \mid REL(f(x)) \mid P(x) \mid \neg\Phi \mid \Phi \circ \Phi \mid \forall x.\Phi[x] \mid \exists x.\Phi[x]$$

где $f$ описывает функтор, $REL$ является некоторой реляции, а $\circ$ бинарный логический оператор.

Выявляется необходимость отделения различных теорий от структурных теории верификации, например, когда речь идет о равенствах целых чисел для достижения эффективного решения вместо раздувания все новых и/или сложных правил. Всязи с этим ставится вопрос об упрощении остальных компонент – программных операторов и описание состояния, как основной подход.

Было показано, что доказательство динамической кучи может быть эффективно реализовано с помощью логического программирования, т.е. с помощью процедур и лемм, автоматизированного метода вывода абстрактных предикатов, термового представления куч и ужесточения пространственных операторов, и т.д.

Процесс верификации определяется как статический метод проверки спецификации. Устанавливается разница в подходах к проверке типов:

| | |
|---|---|
| Проверка типов по Хиндлей-Милнеру: $\vdash_\Gamma e : t$ | вычисление Хора: $\vdash_\Gamma \{P\}C\{Q\} \equiv B$ |
| Проверка типов (type checking), дано: $e, t$: проверить: $\vdash_\Gamma e : t$? | Проверка доказательства (proof checking), где $\overline{A_{i,j}}$ аксиома тройка, $A_{i,j}$ правило: $B \vdash_\Gamma A_{n,0..} \vdash_\Gamma A_{n-1,..} \vdash_\Gamma \overline{A_{0,..}}$, дано: все. |
| Вывод типов (type inference), дано: $e$: найти: $t$ ? | Логический Вывод (inference), дано: $B$, $\Gamma$, найти: $B \vdash_\Gamma A_{n,0..} \vdash_\Gamma A_{n-1,..} \vdash_\Gamma \overline{A_{0,..}}$. |
| Проблема содержимого (inhabitant problem), дано: $t$, найти: $e$? | Проблема содержимого (inhabitant problem), дано: $\{P\}$, $\{Q\}$, найти: $C$ . |

Сравнив имеющиеся объектные вычисления по Абади-Карделли и Абади-Лейно, которые оба не включают указатели, в прототипе предложено и реализовано расширение основаное на объектном вычислении Абади-Карделли, включая указатели и псевдонимы вместе с классами.

Вводятся и утверждаются некоторые конвенции об объектах, которые полезны для исчисления динамических куч. Запрещаются встроенные объекты. Одному указателю положено ссылаться на один объект не более одного раза. Разные указатели могут ссылаться на любые объекты, в том числе, ни на какой, либо все на один. В отличие от структур, объект имеет жизненный цикл, который далее совместим с моделью куч. Подробно обсуждается разница между объектами типизацией и спецификацией. Тип класса определяется всеми полями как: $T_j \to T_{j+1} \to \cdots \to T_k$. По Абади-Лейно объектный экземпляр строится полями и методами: $[f_i = x_i, i = 1..n, m_j = \psi(y_j)b_j, j = 1..m]$ Недостатком является потенциально рекурсивное определение содержимого. Вследствии в общности обработка становится трудной.

Проверка принадлежности классового (под-)типа может быть определена по- элементно следующим образом: $A <: A' \Leftrightarrow [f_i : A_i, i = 1..n + p, m_j : B_j, j = 1..m + q] = A$

$$\wedge[f_i : A_i = 1..n, m_j : B_j, j = 1..m] = A', p, q \in \mathbb{N}_0$$

Показано, что чем сильнее разделение ответственности между объектами, тем меньше имеется связей/зависимостей между указателями, тем проще спецификация динамической памяти, тем меньше имеется исключений, тем полнее и выше качество верификации. Устанавливается аналогия с метрикой МакКейба о сложности графов потоков. Если при верификации обнаруживаются лишние элементы, то спецификация, либо не полная, либо ошибка в программе только что обнаружена. Динамические проблемы не всегда возможно преобразовать в стек из-за неопределенности размера стека и фрагментации.

Альтернативные подходы описания графа кучи включают в себя анализ образов, вычисление регионов, перемещение в стек, или преобразование связей памяти в файловую систему. Например, для сборки мусора общее количество пишущих операций, и связи между ячейками в динамической памяти очень важны для быстрого доступа.

На примере подхода «*ротация указателей*» можно вывести, что компактность записи куч не достаточна. Необходимо исключать неожиданные изменения в описаниях куч,

это можно соблюдать с помощью требования о локальности, которые в основном вводились логикой распределённой памяти и усовершенствуются при строгих операторах делимости кучи, а также модуляризируются с помощью абстрактных предикатов.

Представлены и сравнены альтернативные модели представления динамической памяти, прежде всего — это преобразование в стек, анализ образов и представление куч в файловой системе. Выявлены существенные ограничения альтернативов, а также ради проверки на возможные недостатки и противоречия рассмотрены и косвенные области работы с динамической памятью — это подходы анализа псевдонимов, сбор мусора и интроспекция кода в динамической памяти.

**Вторая глава** уточняет выбранные проблемы из первой главы для верификации динамической памяти. Приводятся мотивирующие примеры и анализ проблем, даётся теоретическое и практическое оправдание актуальности этих проблем.

Обсуждаются причины, примеры проблем, которые могут возникать и их необходимо локализовать: (а) доступ к недоступной памяти, висячие указатели, (б) доступ к не инициализированной памяти, (в) нехватка динамической памяти при востребовании, (г) снижение быстродействия, (д) утечка памяти. В (а) адрес не был выделен, либо ссылка осталась, а содержимое содержит что-то другое/ничего. В (д) происходит переопределение содержимого указателя. Сценарий (в) осуществляется почти всегда после долгого времени запуска и остается без подозрений долгое время.

Предлагается первое неформальное определение общего графа кучи: содержит указатели, которые ссылаются на элементы в динамической памяти. Памятные ячейки могут быть сложными. Граф строится/меняется пошагово согласно выполнению программных операторов. В отличие от стека, куча (динамическая память) неорганизованна и связи между элементами определяются явным образом, а занесение и утилизация проводится специальными операторами.

В работе с динамической памятью проблемы классифицируются и приоризируются на корректность, полноту, быстродействие и адэкватность выбранных представленний. Связанные между этими свойствами критерии критично обсуждаются и формулируются по важности для верификации динамической памяти. Основные существующие теоретические принципы критично анализируются, а их практическое значение оценивается и выделяется.

**Третья глава** посвящается выразимости кучевого представления. Блок видимости определенный для локальных переменных различается от динамически выделенных переменных. Соответствие между началом определения переменной и концом видимости блока отсутствует. Граф кучи меняется по одному элементу в операторах, при этом, изменение должно происходить локально. Опасность «*далекой манипуляции*» нужно включать как усложняющий фактор. Из-за локальности регулярные выражения полностью исключаются. Фрейм определяет динамические кучи до и после вызова процедуры:

$$\frac{\{P\}C\{Q\}}{\{P \star R\}C\{Q \star R\}}$$

Граф кучи определяется как тройка $(V, E, L \times V \cup \{nil\})$, где $V$ множество вершин, $E$ множество граней и $L$ множество наименований указателей. Указатель ссылается, либо на $v \in V$ , либо на $nil$, т.е. не инициализирован. Для представления множеств $V$ и $E$ имеются несколько возможностей, чьи преимущества и недостатки обсуждаются. Выбор делается на представление близкое к «*логике распределённой памяти*». Обыск и обнаружение соседей производится с $O(n)$. Анализируя данные определения куч, выходит, что существующие определения проводятся только строго говоря, множественной формой «*кучи*», единичная форма так и отсутствует. Результатом этого недостатка и неточности в определении влечёт за собой неопределённость, в следствии чего, анализ куч усложняется значительно из-за необходимости полного анализа контекста данной части кучи.

Нетрудно убедиться в том, что явное оправдание даётся лишь за счёт более сложного подхода. На первый взгляд не очевидное замечание приводит к попытке определить граф кучи впервые как:

> Направленный и просто связанный граф, вершины чьи расположены в адресном пространстве динамической части оперативной памяти. Циклы разрешаются. С каждой вершины отходит максимально одна грань, исключением являются объектные экземпляры, поля чьи могут быть указателями. Каждая вершина графа присвоена к некоторому определенному базисному или классному типу, который определят размер вершины. Вершина имеет адрес в динамической памяти и имеет содержимое совместимого значения. Адреса всех вершин различаются, пересечение вершин исключается.

Более формальный в логике распределенной памяти основанное определение далее будет далее рассматриваться в пятом разделе на основе соотношений между кучами.

Обсуждается необходимость ввода предикатов для описания куч. Предикаты параметризируются общими переменными символами для более широкого круга применимости. В отличие от имеющихся подходов, динамические переменные не ограничиваются блоком видимости. Предикаты могут использоваться и ссылаться на любые другие предикаты, в том числе на самих себя. Так как внутренние объекты запрещаются, речь не идет о иерархии куч, а о мелких кучах, которые могут иметь любые разрешаемые указатели.

Отклоняются несколько концепций, т.к. они не рассматриваются уникальными из-за ограниченности в корректности и не являются сопоставимыми простым образом, другими концепциями: рекурсивные спецификации и полиморфизм в классовых методах. Предикаты высшего порядка разрежаются в принципе, но не рассматриваются отдельно при определении куч (см. четвертую главу).

**Четвертая глава** посвящается доказательству через логическое программирование. Правила Хорна определены так:

$$T ::= \begin{cases} x & \text{символ } x \in \overline{X} \text{ из множества допустимых символов} \\ X & \text{символьная переменная } X \in \overline{X} \\ [] & \text{пустой список} \\ [\ T \mid Ts\ ] & \text{список с } T \in \overline{X} \text{ голова терм списка, } Ts \text{ список остаток} \\ [\ T_0\ ,\ \ldots\ ,\ T_n] & \text{список с } T_j \text{ терм, } 0 < j \leq n \\ f(T_0, \ldots, T_n) & f \text{ функтор, } T_j \text{ термы, } 0 \leq j \leq n \\ p(T_0, \ldots, T_n) & p \text{ предикат, } T_j \text{ термы, } 0 \leq j \leq n \end{cases}$$

Функторы также используются для объектных экземпляров классового вида (т.к. в замкнутом списке по определению отсутствует рекурсия). Ради исключения парадоксов, термы проверяются на унифицируемость. Как следует из анализов и сравнений, большинство выявленных ранее проблем спецификации и верификации динамической памяти с помощью Пролог-диалекта могут быть решены. Далее, уровень формализмов снижается до необходимого минимума и семантика диалекта хорошо известного логического языка программирования Пролога не требует полных и всё новых объяснений семантики.

Правило в Прологе состоит из головы и тела. Голова правила может содержать список термов, чьи подтермы могут быть использованы в теле. Тело правила состоит из нуля или более подцелей. Подцель может быть унификация или вызов предиката. Подцели послужат утверждениями, правила послужат абстрактными предикатами.

Далее, следует описание семантики вызовов правил. Тотальность и обратимость предикатов обсуждается в качестве условий унификации спецификации и верификации предикатов над термами, в частности, над кучами. Обратимость предиката рассматривается как ужесточающее свойство, которое полезно для изменения логического запроса. Вызов подцелей для верификации куч проводится слева-направо, что совпадает с семантикой вызовов подцелей Пролога.

Анализируется отсечение решений, которое не только может существенно сократить доказательство, но а также может быть использовано эффективно в кругу распознавателей LL(k). Они выбирают сопоставления образами, как метод сравнения подцепочек входящих токенов в качестве критерия принятия или отклонения данного правила верификации.

Встроенная в Пролог эвристика «поиск с возвратом» рассматривается как точное совпадение с процессом верификации абстрактных предикатов куч. Проблема распознавания левой рекурсии решима полностью с помощью приоритетов правил. Другими словами, правила с конкретными кучами ставятся в начало.

Отрицание предиката может оказаться неопределенно или многозначно, даже если положительный случай определен. Только простые высказывания могут быть отклонены без ограничения общности из-за нерешимости проблемы приостановки (ср. с пятой главой).

Пролог не в состоянии полностью вывести любую формулу предикатной логики, но всегда имеется возможность переписывания правил Хорна так, чтобы это стало возможным.

Возможные логические представления о кучах и правилах верификации обсуждаются к сравнению с функциональными реализациями, которые практически совпадают с соответствующими денотационными семантиками согласно имеющейся реализации на Прологе. Вывод простой: логическое представление более выразимо и проще за счёт более представленных термов и их взаимосвязей.

Первое замечание и формулировка проблемы доказательств о состоянии куч может быть интерпретировано в общем случае, как синтаксический перебор. Оба процесса очень близки, термы предоставляют атрибуты, головы правил и вызовы предикатов нетерминалы, а все остальные интерпретируемые подцели терминалы. Определение подцелей имеет двойную семантику: как классический вызов процедуры, а также как предикат генератор решений. Обе семантики обоснуются на общей реализации в машине Уоррена. Новое при этом, это интерпретация подцелей как вывод согласно данной формальной атрибутируемой грамматики. Параметры связанные с предыдущими и следующими стековыми окнами соответствуют наследованным и синтезируемым атрибутам. Атрибуты являются термами и строятся при необходимости из других существующих термов.

Система верификации основана на теориях, которые задаются данными правилами в Прологе. Правила не подлежат резким ограничениям в связи с изменением или добавлением существующих/новых правил. Отсюда следует модуляризация. С помощью мультипарадигмальной реализации имеющиеся в будущем библиотеки смогут быть подключены, например для реализации всё новых теорий над кучами.

Наблюдается и анализируется, что декларативная семантика Пролога точно соответствует требованиям предикатов куч. Для этого рассматриваются (1) функция переключения между стеками: быстродействие, порядок вычисления, определение семантического контекста и промежуточное представление, а также, (2) функция реляций предикатов: декларативность, использование термов, генерация термов, проверка термов.

Терм как промежуточное представление предлагается, а возможное использование анализируется и принимается.

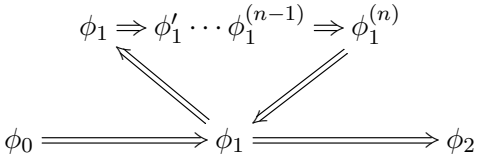
Все то, что было сказано про спецификацию касательно простоты и короткого описания, также распространяется 1:1 на верификацию при использовании одного общего языка. Чем меньше имеется исключений, тем больше растет множество допустимых элементов и тем проще обобщаются предикаты.

Подробный анализ по количеству примеров между логическим и функциональным представлением трансформации термов (полученные из XML документов) с более 80 примерами, показал, что обработка с помощью реляционной модели практически всегда, а в общем арифметическом на 30% превышает функциональный подход по соотношениям операторов к операндам, что представляет собой довольно большую разницу. В среднем, описание Пролога на 50% короче функционального. Функциональное описание

сильно страдает от отсутствия эффективной возможности равномерного расширения и вариабельности языков спецификации и верификации. Императивные подходы описания исключаются с самого начала из-за расследованных причин из первой главы – одной из которых, являются побочные эффекты манипулирующие состояние вычисления.

Процесс верификация куч полностью определяется на основе машины Уоррена, например: порядок вычисления, типизация и категоризация термов, выявление реакций подцелей, символьное использование в термах, а также перегрузка наименований и символов значений. В связи с типизацией замечается, что проверка содержимого типа эквивалентна генерации минимальной программы, которая манипулирует динамическую память.

Предлагается архитектура верификатора динамической памяти на принципах: (1) автоматизация, (2) открытость, (3) расширяемость и (4) обоснованность.




**Пятая глава** диссертации посвящена проблеме многозначности оператора связывания куч в классической логике распределенной памяти. Конъюнкция пространственного оператора $\star$ связывает предположительно разделенные кучи вместе, однако, не исключено согласно неявному определению конъюнкции, что кучи логически не связаны друг с другом, как это может произойти с мостиками. Выходит, что оператор связывания на самом деле, может быть использован как для разделения куч, так и для связывания куч. Это усложняет не только анализ хода верификации, но а также требует раздутые правила верификации, т.к. необходимо покрывать множество случаев. Полное соблюдение случаев требует квадратную сложность и проверку простой кучи со всеми остальными простыми кучами.

Единства синтаксиса, семантики и прагматизма здесь нарушены и далее задача заключается в корректировке дисбаланса без нарушения основных свойств. Ужесточение пространственного оператора не ограничивает выразимость, оно лишь удаляет контекстзависимость. Семантически куча является графом в динамической памяти и зависимость от соседних куч кажется неадекватной и необоснованной. Исключение многозначности по синтаксису приводит к более интуитивному и формальному обозначению и представлению «*кучи*».

Для четких выражений, которые соответствуют различным кучам, необходимо решить проблему изоморфизма графа куч. Изоморфизм граф решается в общем случае экспоненциальной сложностью. Для графов куч сущетсвуют разовые наименования указателей, а указатели не могут имееть одинаковые локации. Далее, лексикографический порядок указателей упрощает сравнение. Таким образом сравнение куч можно проводить за полиномиальное, даже линейное время.

*Конъюнкция куч* $H \circ \alpha \mapsto \beta$ определяется как граф кучи, где $G = (V, E)$ является представлением графа кучи от $H$, $H$ является граф кучи, и где $\alpha \mapsto \beta$ является атомной кучей ссылки:

$$\begin{cases} (V \cup \{\alpha, \beta\} \cup \beta', & \text{если } isFreeIn(\alpha, H) \\ E \cup \{(\alpha, \beta)\} \cup \{(\beta, b) | b \in \beta'\}) & \text{если } H = \underline{emp} \\ & \quad (V = E = \overline{\emptyset}) \\ \underline{\text{false}} & \text{иначе} \end{cases}$$

где $\beta' = vertices(\beta) \subseteq V$ определяет все вершины графа куч, которые указываются из $\beta$

Условие конъюнкции, что подкучи обязательно связаны между собой. Определение тотальное. По умолчанию согласуется, что $H^{-1} \circ \underline{emp} \equiv \underline{emp} \circ H^{-1} \equiv H$ в силе. Из

определения сразу следует формула конъюнкции $\Phi$ в силе, если $\Phi = \Phi_0 \circ a_0 \mapsto b_0 \Leftrightarrow \exists(a_m \mapsto b_m) \in \Phi_0 \wedge (a_m = a_0, b_m \neq b_0 \vee a_m \neq a_0, b_m = b_0)$.

Доказана теорема о конъюнкции, что если даны две кучи $H_1$ и $H_2$, то конъюнкция $\circ$ связывает данные кучи $H_1$ и $H_2$ вместе к одной куче $H_1 \circ H_2$, если существует хотя бы одна общая вершина в обоих их представлениях графов куч, которая является общей. Ради полноты, индуктивно необходимо всегда показывать, что свойства действительны для обобщенных куч.

Теперь можно определить моноид над множеством графов куч и конъюнкцией. Из замкнутости следует $c \in b \wedge c = a : a \mapsto b \circ c \mapsto d \equiv \underline{false}, a \mapsto b \circ a \mapsto d \equiv \underline{false}$. Далее, можно показать, что для данного моноида свойства Абельской группы соблюдаются. Для этого необходимо ввести понятие «*обратимой кучи*» и коммутативность. Обратимость кучи определена как $H \circ H^{-1} \equiv H^{-1} \circ H \equiv \underline{emp}$, вводится соответствующее определение и интуитивно доступное понятие. С помощью показанных свойств группы, теперь можно «*решать*» кучами более удобным способом, например, устанавливать равенства куч, которые могут способствовать упрощению и нормализации куч.

Конъюнкция обратимости может привести к изоляции подкуч, если грань графов куч является мостом. Поэтому, необходимо подключить еще два генеричных шага нормализации, которые заменят при необходимости оставшуюся конъюнкцию на дизъюнкцию. Обратимость кучи означает следующее:

1. удалить грань между $a$ и $b$.
2. удалить вершину $a$, если больше не имеется на нее ссылок в графе $H$ .
3. также удалить вершину $b$, если больше не имеется на нее ссылок в графе $H$.

Гомоморфизм $(g_1 \circ g_2)^{-1} \equiv g_1^{-1} \circ g_2^{-1}$ соблюдается. Он представляет собой полезную эквивалентность, например в поисках нормальной формы.

Дизъюнкция кучи H с простой кучей $a \mapsto b$ определено как две кучи $H$ и куча $a \mapsto b$, которые гарантировано не связанны между собой. Аналогично конъюнкции, для дизъюнкции также можно выявить свойство моноида. Более того, показаны свойства дуальности и дистрибутивности обеих операций.

Свойство строгости конъюнкции и дизъюнкции позволяет анализировать термы последовательно без возврата. Не полностью специфицированные кучи с помощью константных кучевых интерпретаций, например, $\underline{true}$ и $\underline{false}$, могут решить вопрос о полноте данного набора правил через неполную спецификацию.

Объектный экземпляр может быть представлен кучей, которая замкнута. Левые стороны всех полей определяются априори данным классом. Условия в силе для указателей, также распространяются и для объектных экземпляров.

Объектные экземпляры должны соблюдать следующие требования, для полной совместимости с логикой распределенной памяти, с имеющимися условиями представленным ранее и моделированием «*UML/OCL*»: (i) запрет внутренних объектов, (ii) объектные поля отличаются, (iii) размер объектного экземпляра по фиксированному адресу в динамической памяти определен и не может меняться, (iv) массивы в качестве базисного типа запрещаются, (v) в качестве вершины графа кучи могут выступать как простые указатели, так и указатели на объекты, чьи поля могут опять же быть указателями, (vi) объектам пересечения их ячеек памяти запрещено, (vii) указатели на «*четные*» адреса допускаются, на «*нечетные*» адреса запрещаются.

Граф куч в виде граненного представления является согласно четвертой главе главным. Однако, платформа на столь гибкая, что если алгоритм в будущем потребует иное представление граф, то оно может быть будет выведено с помощью встроенных и мультипарадигмальных предикатов. Для этого вводится обсуждение графа представленного вершинами с помощью предиката «*reaches*», который нетрудно сгенерировать, т.к. грани могут быть выявлены с помощью семантики генератора и унификации.

Спецификация объекта производится как: $obj.f1 \mapsto .. \circ obj.f2 \mapsto .. \circ obj.fn \mapsto ...$ $obj$ является переменным символом и может быть использован в частичных предикатах,

например $\underline{true}(obj1)$. Если имеются несколько объектов, то модуляризация соблюдается и любое поле присвоено точно одному объекту.

С помощью «*мемоизатора*», который представляет кэш для вызовов предикатов, можно при анализе конъюнкций и дизъюнкций эффективно выявлять повторяющиеся кучи. Улучшение не измеряется, т.к. принцип кэша хорошо известен и не требует далее исследований. Мемоизатор эффективен для абстрактных предикатов, для предикатов с писанием/чтением в/из файла может иметь смысл его отключать.

**Шестая глава** посвящена автоматизации верификации с абстрактными предикатами куч. Абстрактные предикаты являются основой модуляризации. На практике тяжело установить, когда нужно определение предиката свернуть или наоборот развернуть (P/C), для того, чтобы доказательство довести до успеха или провала. Правила могут даже привести к тому, что невозможно доказательство подтвердить или P/C может оказаться бесконечной. В общем случае проблему можно редуцировать на проблему корреспонденции Поста (для которой была предложена Эмилем Постом абстрактная машина как вычислительная теоретическая модель). Для автоматизации конъюнкции необходимо логическую конъюнкцию и дизъюнкцию куч, а также отсечение преобразовать. Это делается непосредственно в правилах Прологовской программы, затем они интерпретируются как формальная грамматика. Константные интерпретации emp, true, false куч представляют синтаксический сахар, чья роль заключается в принятии всех последовательных простых куч. Если имеется (связанная) куча в виде: $a_0 \circ a_1 \circ \cdots \circ a_n$, при $\forall j.a_j$, то продукт можно сравнивать с имеющейся кучей поэлементно для каждого абстрактного предиката. Абстрактные предикаты имеют следующий вид:

$$\forall n.a :- q_k \times n. \Leftrightarrow a :- q_{k,0}, q_{k,1}, \cdots, q_{k,n}$$

для любых, но определенных целых чисел $k$. Так как, предикаты могут иметь альтернативы, также необходимо их включить к набору предикатов. *Набор предиката* $\Gamma_a$ для некого предиката именем $a \in T$ и $\forall i, j.q_{i,j} \in (T \cup NT)$, где $T$ терминалы, а $NT$ нетерминалы, определен как: $\Gamma_a ::= a :- q_{m \times n}$

$$\begin{array}{rccccccc} & a :- & q_{0,0} & , & q_{0,1} & , & \cdots & , & q_{0,m} \\ \Leftrightarrow & \vdots & \vdots & & \vdots & & \ddots & & \vdots \\ & a :- & q_{m,0} & , & q_{m,1} & , & \cdots & , & q_{m,n} \end{array}$$

К $a$ могут быть приписаны термы содержашие символы (например, когда $m = 0$, $n > 0$). Если $t \in T$, то $t$ имеет вид $loc \mapsto val$, иначе $t \in NT$ означает, предикат под именем $t$, который имеется в $\Gamma$.

При этом, порядок определения предиката остается в силе. Показано, как кучи верифицируются с помощью распознавателей, чьи формальные грамматики представлены правилами Хорна. Абстракция предикатов эквивалентна, при этом параметризацией куч, что эквивалентно атрибутированию данной грамматики.

Преобразование от определения кучи в атрибутированную грамматику можно произвести с помощью денотационной функции C и D следующим образом:

$C(a)[\![a(\vec{y}) :- q(\vec{x}_{k,n})_{k \times n}]\!]\sigma = D[\![q_{k,n}]\!]\sigma(\vec{x}_{k,n}) \circ \cdots \circ D[\![q_{k,1}]\!]\sigma(\vec{x}_{k,1})$.

где $C$ имеет тип $atom \to predicate \to \sigma \to \sigma$, где $D$ имеет тип $subgoal \to \sigma \to \sigma$, где $\sigma$ термовая среда имеет тип $term^* \to term$. Преобразование обратно производится аналогично с помощью $C[\![.]\!]^{-1}$.

Связанность, годность и семантические условия куч проверяются для каждого абстрактного предиката на этапе построения и интерпретации формальной грамматики. Во время распознавания входное слово проверяется со сравниваемой кучей.

Пролог принципиально был предложен, как язык программирования для работы с предикатным представлением. Однако Уоррен, Колвальский и Колмерауер предупреждали, что любые формулы априори не всегда могут быть выводимы. Выводимость зависит от

представления логической формулы и постановления запроса. Для этого имеется два подхода: либо ограничение, либо переоформление правил вывода.

Предикаты высших порядков допускаются, но только не в качестве определения абстрактного предиката куч. Для куч, предикаты высших порядков не обязательно нужны, с другой стороны, далее абстракция предикатов ломала бы деление между нетерминалами и терминалами в соответствующей атрибутированной грамматике.

*Развертывание/Свертывание (P/C) предиката* $a(\vec{\alpha})$ из/в некоторый предикат $a$ для данных предикатов $\Gamma_a$ с настоящими значениями термов $\vec{\alpha}$/подцелей $q_k$ определяется как: $a(\vec{y}) : -q_k$, где $q_k = q_{k,0}(\vec{x}_{k,0}), q_{k,1}(\vec{x}_{k,1}), ..., q_{k,m}(\vec{x}_{k,m})$.
Если $\vec{\alpha} = (\alpha_0, \alpha_1, .., \alpha_A)$ и $\vec{y} = (y_0, y_1, .., y_A)$, то $a(\vec{\alpha}) \Leftrightarrow q_{k,0}(\vec{x}_{k,0}), q_{k,1}(\vec{x}_{k,1}), ..., q_{k,m}(\vec{x}_{k,m})$ с $\alpha_0 \approx y_0, \alpha_1 \approx y_1, ..., \alpha_A \approx y_A$.
Важно заметить, что контекст-свободность распространяется на не повторение одинаковых терминалов, а на терминалы без атрибутов. Атрибутированные терминалы естественно должны различаться. Транслирующие правила соответствуют подцелям, чьи имена не конфликтуют с остальными предикатами.

Проблема распознавания слова теперь можно сформулировать как $\alpha_1, \alpha_2 \in L(G(P))$, где $\alpha_1, \alpha_2$ являются входными словами (одно из которых, цепочка терминалов и нетерминалов, т.е. является спецификацией). $L(G(P))$ является генерируемым языком грамматики, который описывается Прологовскими правилами.

Вопрос решаемости о сходимости верификации редуцируем следующим образом: Если имеются цепочка вывода и правила, то верификация всегда сходима, когда правила детерминированы.

Распознавание слова можно определить с помощью двух операторов вычисления множества наследственных терминалов $\sigma(\alpha)$ и множества началов нетерминалов $\pi(\alpha)$. Таким образом, доказательство редуцируется на синтаксический перебор.

Проблемы многозначности разрешаются на этапе определений правил абстрактных предикатов куч и на этапе синтаксического перебора. Обсуждаются свойства преобразований, процесса синтаксического перебора и концепция «*Доказательство это синтаксический анализ*», а также границы такого подхода и реализации.

## 3 Основные выводы и результаты

Основными результатами работы являются следующие:

1. Выполнен анализ существующих ограничений выразимости и для их устранения предложен унифицированный диалект языка Пролог, позволяющий выполнять одновременную спецификацию и верификацию динамической памяти и, за счёт повышения выразимости, устраняющий многозначность описаний;
2. Разработана обобщенная архитектура верификатора динамической памяти, устраняющая дублирование определений, обеспечивающая расширяемость и вариабельность промежуточного представления (IR). Предложена программная реализация верификатора в виде конвейера;
3. В качестве наиболее эффективной для описания абстрактных предикатов куч предложено использование реляционной модели. Продемонстрирована теоретическая сводимость предложенного диалекта языка к реляционной алгебре;
4. Обоснована возможность построения новых теорий над кучами до запуска абстрактных предикатов. Данные теории могут задаваться непосредственно как абстрактные предикаты в диалекте, либо передаваться произвольно задаваемыми SAT-решателями в мультипарадигменных средах программирования.
5. Предложены и обоснованы расширения языка моделирования «*UML*» и языка «*OCL*» указателями, термами и кучами, показана возможность автоматизированного вывода над абтрактными предикатами; показаны ограничения представления куч и правил верификации с использованием термов.

6 Показано, что представляемые в диалекте абстрактные предикаты над кучами определяют атрибутную грамматику и то, что синтаксический анализ структуры куч совпадает с процессом их верификации, что определяет возможность автоматического сравнения динамической памяти с имеющимся экземпляром памяти при условии изменения входных данных

7 Предложено новое графовое определение единичной кучи с помощью конъюнкции куч и групповых свойств. Достигнуто упрощение спецификации и верификации с целью ужесточения выражений пространственности куч. Предложен новый способ частичной спецификации объектных экземпляров по виду вычисления Абади-Карделли, которая уменьшает риск не полностью определенных правил.

# 4 СПИСОК ПУБЛИКАЦИЙ ПО ТЕМЕ ДИССЕРТАЦИИ

## Публикации в изданиях рекомендуемых ВАК


[1] Хаберланд, Р., Ивановский, С. А., Кринкин, К. В. Верификация Объектно-ориентированных программ с динамической памятью на основе ссылочной модели с помощью Пролога. / Р. Хаберланд // Известия ЛЭТИ, Санкт-Петербургский Электротехнический Университет, Санкт-Петербург. – 2016. №1. С.14–18. ISSN 2017–8985, УДК 004.052.2.

[2] Хаберланд, Р. Сравнительный анализ статических методов верификации динамической памяти. / Р. Хаберланд // Компьютерные инструменты в образовании, Санкт-Петербург. – 2019. №2. С.5-30. DOI: 10.32603/2071-2340-2019-2-5-30, УДК 004.052.2.


## Зарегистрированные программы


[1] Хаберланд, Р., Кринкин К. В., Программа для быстрого построения проектов программного обеспечения (Builder), инструментарий для верификации динамической памяти. Санкт-Петербургский государственный электротехнический университет (ЛЭТИ) — ПрЭВМ в Роспатенте, 18.12.2018г. №2019610438

[2] Хаберланд, Р., Кринкин К. В., Программа для динамического сокращения (слайсинга) программы (Shrinker), инструментарий для верификации динамической памяти. Санкт-Петербургский государственный электротехнический университет (ЛЭТИ) — ПрЭВМ в Роспатенте, 18.12.2018г. №2018664644

[3] Хаберланд, Р., Кринкин К. В,. Программа для верификации динамической памяти (ProLogika), инструментарий для верификации динамической памяти. Санкт-Петербургский государственный электротехнический университет (ЛЭТИ). — ПрЭВМ в Роспатенте, 18.12.2018г. №2019610339

[4] Хаберланд, Р., Кринкин К. В,. Программа для преобразования XML-документов с помощью логического программирования (Prolog-XML). Санкт-Петербургский государственный электротехнический университет (ЛЭТИ). — ПрЭВМ в Роспатенте, 27.11.2019г. №2019666260


## Материалы по теме


[1] René Haberland. Tools: builder, shrinker, ProLogika, Prolog for XML, XML processors in Haskell, dynamic memory tutorials, heap tools, examples, Measure Samples/Code examples. St. Petersburg Electrotechnical University. Nov. 2016. https://bitbucket.org/reneH123.

## Публикации в других изданиях

[1] Хаберланд, Р. Верификация корректности динамической памяти с помощью логического языка программирования. / Р. Хаберланд // Конференция $EMC^2$ «Технологии Майкрософт в Теории и на Практике Программирования. Серия: Новые подходы к разработке программного обеспечения», С.-П. Политехнический Университет, Санкт-Петербург. – 2014. №3. С.56–57.

[2] Хаберланд, Р. Расширяемый Фреймворк для верификации статически типизированных объектно-ориентированных программ, использующих динамическую память. / Р. Хаберланд // Санкт-Петербургский Электротехнический Университет «ЛЭТИ» (ППС ЛЭТИ), Санкт-Петербург. – дек. 2015. №5. УДК 004.052.2.

## Публикации в индексированных изданиях

[1] René Haberland, Kirill Krinkin. *A Non-repetitive Logic for Verification of Dynamic Memory with Explicit Heap Conjunction and Disjunction*. In: International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP). Ed. University of Venice, Italy. Oct. 2016, p.1–9. ISBN 978-1-61208-506-7, ISSN 2308–4498, IARIA XPS Press/ThinkMind.

[2] René Haberland, Kirill Krinkin, Sergey Ivanovskiy. *Abstract Predicate Entailment over Points-To Heaplets is Syntax Recognition*. In: 18th Conference of Open Innovations (FRUCT), ISSN 2305-7254, ISBN 978-952-68397-3-8. Ed. By T. Tyutina S., Balandin A., Levina. 2016, p.66–74. DOI: 10.1109/FRUCT-ISPIT.2016.7561510.

[3] René Haberland. *A Stricter Heap Separating Points-To Logic*. In: 3rd Int. Scientific Symposium Sense Enable (SPITSE2016). Ed. by Russia National Research University Moscow. June 2016, p.103–104. РИНЦ 26444969.

[4] René Haberland, Sergey Ivanovskiy. *Dynamically Allocated Memory Verification in Object-Oriented Programs using Prolog*. In: Proc. of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2014). Ed. by Alexander Kamkin, Alexander Petrenko, and Andrey Terekhov. 2014, p.46–50. ISBN 978-5-91474-020-4, ISSN 2311–7230, DOI: 10.15514/SYRCOSE-2014-8-7.

[5] René Haberland, Igor L. Bratchikov. *Transformation of XML Documents with Prolog*. In: Advances in Methods of Information and Communication Technology. Vol. 10. 2008, p.99–111. ISBN 975-5-8021-1020-1, УДК 519.1+681.3.

[6] René Haberland. *Unification of Template-Expansion and XML-Validation*. In: Proc. of Int. Conf. on Control Processes and Stability. Vol. 34. Saint Petersburg State University, 2008, p.389–394. ISBN 978-5-288-04680-3, ISSN 2313–7304, РИНЦ, УДК 517.51:517.9:518.9.

[7] René Haberland. *Using Prolog for Transforming XML-Documents*, Cornell University, arXiv:1912.10817 [cs.PL], 49p., 2007/2019

[8] René Haberland. *Narrowing Down XML Template Expansion and Schema Validation*, Cornell University, arXiv:1912.10816 [cs.PL], 46p., 2007/2019