Автономная некоммерческая организация высшего образования
«Университет Иннополис»

АННОТАЦИЯ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ)
ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ
09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

НАПРАВЛЕННОСТЬ (ПРОФИЛЬ) ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ
«ПРОГРАММНАЯ ИНЖЕНЕРИЯ»

Тема	**Руководство по реализации лямбда-исчисления с параметрическим предикативным полиморфизмом произвольного порядка**

Выполнил	**Данько Данила Константинович**	подпись

Иннополис, Innopolis, 2025

Оглавление

Глава 1

Общая характеристика работы

Настоящая диссертация посвящена проектированию и реализации `Arralac` — небольшого функционального языка программирования и сопутствующего ему компилятора. Основная цель проекта — преодолеть педагогический разрыв между сложной академической теорией систем типов и практикой реализации современных компиляторов. Работа не претендует на научную новизну, а фокусируется на применении известных, но передовых инженерных подходов для создания прозрачного, модульного и интерактивного инструмента, предназначенного для изучения внутреннего устройства компилятора.

Центральный тезис работы заключается в том, что сфокусированная, современная и интерактивная реализация, сознательно отходящая от старых алгоритмических моделей, может служить более эффективным средством обучения, чем изолиро-

ванное изучение фундаментальных научных статей или исходного кода промышленных компиляторов.

Работа представляет собой полный цикл разработки: от теоретического обоснования и проектирования архитектуры до реализации, тестирования и анализа полученной системы. Особое внимание уделяется двум ключевым аспектам современной инженерии компиляторов: архитектуре вывода типов на основе ограничений (constraint-based) и представлению Абстрактного синтаксического дерева (АСД) с помощью паттерна «Растущие деревья» (Trees That Grow).

Глава 2

Актуальность темы

Продвинутые системы типов являются краеугольным камнем современного функционального программирования. Одной из таких мощных возможностей является **полиморфизм произвольных рангов** (arbitrary-rank polymorphism), который позволяет функциям принимать в качестве аргументов другие полиморфные функции. Эта возможность лежит в основе реализации таких продвинутых языковых средств, как обобщённые алгебраические типы данных (GADT), программирование на уровне типов и сложные формы обобщённого программирования.

Теоретические основы для реализации этой функции в практическом компиляторе были заложены в основополагающей статье Пейтона Джонса и др. «Practical type inference for arbitrary-rank types» [1]. Самой известной реализацией этих идей является компилятор языка Haskell — Glasgow Haskell Compiler (GHC) [2]. Для многих разработчиков, включая автора данной работы, внутреннее устройство GHC представляет собой вершину компиляторных технологий — достаточно развитую технологию, ко-

торая может казаться магией.

Несмотря на наличие как фундаментальной теории, так и промышленной реализации, существует значительный педагогический разрыв. Начинающие разработчики языков и студенты сталкиваются с крутой кривой обучения при попытке понять, как элегантная теория полиморфизма произвольных рангов преобразуется в практический код. Этот разрыв обусловлен несколькими факторами:

- **Академическая литература**, включая [1], представляет теоретически плотную систему, построенную на сложном взаимодействии **субсумпции** (subsumption), **глубокой сколемизации** (deep skolemization) и **двунаправленной проверки типов** (bidirectional checking). Хотя система формально корректна [3], описание алгоритма **«жадного» объединения** (eager unification) в статье опускает множество практических инженерных деталей, необходимых для построения надёжной системы.

- **Исходный код GHC**, будучи бесценным ресурсом, представляет собой огромную, высокооптимизированную промышленную систему. Его современная архитектура вывода типов, **основанная на ограничениях** (constraint-based), является значительной эволюцией по сравнению с «жадной» моделью, описанной в основополагающих статьях. Это затрудняет прослеживание связи между теорией и реализацией для новичка. Современные аналоги, такие как MicroHs [4], [5], также являются крупными проектами.

- **Существующие учебные компиляторы** для Haskell, такие как Hugs [6], устарели или не включают эти современные архитектурные паттерны, оставляя студентов без моста от базовых принципов к современному состоянию дел.

Таким образом, возникает очевидная потребность в ресурсе, который преодолевает этот разрыв, — «промежуточном звене», более конкретном, чем статья, но более сфокусированном и доступном, чем полноценный промышленный компилятор, и который явно демонстрирует архитектурную эволюцию от «жадной» модели к выводу типов на основе ограничений.

Глава 3

Цель, предмет и объект исследования

Основной **целью** данной работы является создание дидактического компилятора `Arralac`, который служит учебным пособием по реализации полиморфизма произвольных рангов с использованием современных архитектурных практик.

Объектом исследования является процесс проектирования и реализации компиляторов для функциональных языков программирования с продвинутыми системами типов.

Предметом исследования является архитектура и реализация компилятора `Arralac` как дидактического инструмента для изучения полиморфизма произвольных рангов, в частности, модель вывода типов на основе ограничений, паттерн «Растущие деревья» и интеграция с инструментами интерактивной разработки.

Для достижения поставленной цели были сформулированы

следующие **задачи**:

1. Реализовать ключевые алгоритмы двунаправленной системы вывода типов для полиморфизма произвольных рангов, включая субсумпцию и глубокую сколемизацию, в ясной и модульной манере, пригодной для образовательных целей.

2. Продемонстрировать на практике, что модель вывода типов, основанная на ограничениях (разделение генерации и решения), служит более понятным педагогическим инструментом для объяснения вывода типов, чем модель «жадного» объединения, представленная в фундаментальной литературе.

3. Использовать протокол языкового сервера (LSP) для создания интерактивной среды разработки, которая делает поведение и результаты конвейера вывода типов языка прозрачными и исследуемыми, превращая абстрактные правила в конкретную обратную связь.

4. Создать общедоступный репозиторий с документированным исходным кодом [7] в качестве ресурса для сообщества для обучения и экспериментов.

Глава 4

Методы исследования

Для решения поставленных задач в работе использовался комплексный подход, включающий следующие методы:

- **Теоретические методы:** анализ и синтез научно-технической литературы по теориям систем типов, алгоритмам вывода типов и архитектуре компиляторов; изучение основополагающих работ (в частности, [1]) и современных исследований в области.
- **Методы системного проектирования:** применение модульного подхода и конвейерной архитектуры для декомпозиции сложной задачи компиляции на управляемые этапы (парсинг, переименование, проверка типов, решение ограничений, оценка).
- **Методы программной инженерии:** реализация системы на языке функционального программирования Haskell; применение современных архитектурных паттернов, таких как

«Растущие деревья» (Trees That Grow), для создания расширяемого и типобезопасного АСД; использование системы управления зависимостями Nix для обеспечения воспроизводимости сборки.

- **Эмпирические методы:** разработка и использование набора целевых тестовых примеров для верификации корректности реализации ключевых механизмов системы типов (обработка полиморфизма высших рангов, обнаружение утечки сколемов); ручное тестирование интерактивных функций языкового сервера.

Глава 5

Научные и практические результаты

В ходе выполнения диссертационной работы были получены следующие основные результаты:

1. **Разработан и реализован компилятор** `Arralac`, представляющий собой функциональный, хорошо структурированный и нетривиальный учебный проект. Система успешно компилирует и выполняет программы на языке с полиморфизмом произвольных рангов.

2. **Продемонстрирована эффективность двухфазной архитектуры вывода типов.** В отличие от «жадной» модели из [1], `Arralac` разделяет генерацию ограничений и их решение. Это позволило создать более модульную систему, где

логика проверки типов и логика унификации полностью разделены, а также заложить основу для более качественной диагностики ошибок.

3. **Реализовано представление АСД с использованием паттерна «Растущие деревья».** Данный подход обеспечил типобезопасное добавление аннотаций (таких как выведенные типы) на разных стадиях конвейера компиляции, что является необходимым условием для создания современных инструментов разработки и делает эволюцию структуры данных наглядной.

4. **Создан интерактивный инструментарий на основе LSP.** Реализация языкового сервера, который предоставляет информацию о выведенных типах при наведении курсора и сообщает об ошибках в реальном времени, превращает компилятор из «чёрного ящика» в интерактивный инструмент для изучения его работы.

5. **Создан общедоступный образовательный ресурс.** Весь исходный код проекта, включая подробные комментарии, опубликован в открытом доступе и снабжён средствами для быстрой и воспроизводимой установки, что позволяет использовать его в качестве учебного пособия.

Глава 6

Структура и краткое содержание работы

Диссертация состоит из введения, четырёх глав, заключения и списка литературы.

Во введении обосновывается актуальность темы, определяется проблема педагогического разрыва между теорией и практикой в области компиляторов, формулируются цели и задачи исследования, а также описывается вклад работы в решение этой проблемы.

В первой главе, «Обзор литературы», представлены теоретические основы работы. Глава прослеживает эволюцию полиморфных систем типов: от фундаментальных ограничений Простого типизированного лямбда-исчисления, через теоретический идеал System F и практический компромисс системы типов Хиндли-

Милнера, к «пробелу N-го ранга» (Rank-N gap). Завершается глава детальным обзором двунаправленного подхода для типов произвольных рангов, предложенного в [1], который является теоретическим ядром диссертации.

Во второй главе, «Проектирование и методология», подробно описывается архитектура и реализация языка `Arralac`. Рассматривается полный конвейер компиляции. Особый акцент сделан на ключевых архитектурных решениях: выборе паттерна «Растущие деревья» для представления АСД, а также на сознательном отходе от «жадной» унификации в пользу двухфазной архитектуры проверки типов на основе ограничений, вдохновлённой GHC. Описывается использование уровней (TcLevel) для контроля области видимости полиморфных переменных.

В третьей главе, «Реализация и результаты», демонстрируется практическая реализация спроектированной системы на языке Haskell. Описывается структура данных для АСД и её инстанцирование для разных фаз компиляции. Детально рассматривается конвейер вывода типов: генерация ограничений, их решение отдельным модулем-решателем и финализация (зонкинг). Глава завершается демонстрацией работы компилятора на конкретных примерах: успешная проверка типов для программы с полиморфизмом высших рангов, корректное обнаружение ошибки (утечка сколема) и работа интерактивных инструментов (LSP).

В четвёртой главе, «Анализ и обсуждение», проводится критический анализ полученной системы. Оценивается, насколь-

ко успешно выбранные архитектурные решения (вывод типов на ограничениях, «Растущие деревья») способствовали достижению педагогических целей. Обсуждаются компромиссы принятых решений, а также качественные характеристики системы, такие как модульность и анализируемость. В заключительной части главы определяются ограничения текущей реализации (отсутствие let-генерализации, неспособность решателя сообщать о нескольких ошибках сразу) и намечаются конкретные направления для будущих исследований.

В заключении подводятся итоги работы, делается вывод о достижении поставленных целей. Пересматривается центральный тезис в свете полученных результатов и подтверждается, что современная, сфокусированная реализация является эффективным средством для демистификации сложных компиляторных технологий. Намечаются дальнейшие шаги по развитию проекта.

Список литературы

[1] S. P. Jones, D. Vytiniotis, S. Weirich и M. Shields, «Practical type inference for arbitrary-rank types», *J. Funct. Prog.*, т. 17, № 1, с. 1—82, янв. 2007, ISSN: 0956-7968, 1469-7653. DOI: 10.1017/S0956796806006034. дата обр. 3 апр. 2025. url: https://www.cambridge.org/core/product/identifier/S0956796806006034/type/journal_article.

[2] «The Glasgow Haskell Compiler», дата обр. 27 апр. 2025. url: https://www.haskell.org/ghc/.

[3] D. Vytiniotis, S. Weirich и S. Peyton-Jones, «Practical type inference for arbitrary-rank types - technical appendix», url: https://repository.upenn.edu/server/api/core/bitstreams/7c1dc678-93c6-4516-98fd-f82b384eb75d/content.

[4] L. Augustsson, «MicroHs: A Small Compiler for Haskell», в *Proceedings of the 17th ACM SIGPLAN International Haskell Symposium*, сер. Haskell 2024, New York, NY, USA: Association for Computing Machinery, 28 авг. 2024, с. 120—124, ISBN: 979-8-4007-1102-2. DOI: 10.1145/3677999.3678280. дата обр. 15 июля 2025. url: https://dl.acm.org/doi/10.1145/3677999.3678280.

[5] «augustss/MicroHs: Haskell implemented with combinators», дата обр. 20 мая 2025. url: https://github.com/augustss/MicroHs.

[6] «Hugs 98», дата обр. 16 июля 2025. url: https://www.haskell.org/hugs/.

[7] D. Danko, *deemp/arbitrary-rank-tutorial*, original-date: 2024-11-11T16:24:27Z, 27 апр. 2025. дата обр. 27 апр. 2025. url: https://github.com/deemp/arbitrary-rank-tutorial.