



Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»
КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к научно-исследовательской работе студента

на тему

«Исследование систем типов в языках программирования»

Студент РК6-75Б
группа

подпись, дата

Никитин В.Л.
ФИО

Руководитель НИРС

подпись, дата

Соколов А.П.
ФИО

Москва, 2024

РЕФЕРАТ

научно-исследовательская работа студента: 17 с., 1 рис., 1 табл., 6 источн.

ТЕОРИЯ ТИПОВ, ЯЗЫКИ ПРОГРАММИРОВАНИЯ, КОМПИЛЯТОРЫ, ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ, СИСТЕМА ТИПОВ ХИНДЛИ-МИЛНЕРА.

Работа посвящена анализу систем типов современных языков программирования. Программирование выстроено вокруг глубокой математической теории. Благодаря этому появляются возможности для оптимизации, развития и улучшения языков посредством применения математики. Одним из важных применений является теория типов, которая помогает программисту в написании кода. В последнее время все больше и больше языков почерпывают что-то из этой области. Поэтому важно при разработке своего собственного языка программирования сформировать его структуру и идематику, используя те или иные приемы из математической теории.

Тип работы: научно-исследовательская работа студента.

Тема работы: *«Исследование систем типов в языках программирования».*

Объект исследования: системы типов.

Основная задача, на решение которой направлена работа: Анализ систем типов современных языков программирования.

Цели работы: проведение анализа среди существующих систем типов с целью выбрать одну из них для реализации в компилятора языка программирования Kodept

В результате выполнения работы: 1) проведен анализ систем типов некоторых современных языков программирования 2) предложено использовать систему типов Хиндли-Милнера

ОПРЕДЕЛЕНИЯ

Kodept экспериментальный язык программирования, разрабатываемый в рамках обучения на кафедре 2

лямбда-исчисление формальная система, предназначенная для анализа вычислимости 8

система типов набор правил в языке программирования, по которым выражениям назначаются типы 5, 6

типизированное лямбда-исчисление лямбда-исчисление, где лямбда-термам приписаны типы 10

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ	3
ВВЕДЕНИЕ	5
1 Теория типов	6
2 Классификация систем типов	8
2.1 Система типов C	9
2.2 Система типов Kotlin	9
2.3 Система типов ML-подобных языков	10
3 Система типов Хиндли-Милнера	13
ЗАКЛЮЧЕНИЕ	16
Литература	17

ВВЕДЕНИЕ

В современном программировании становится все более важным сколько будет потрачено времени на создание того или иного продукта. В это число входит как время, потраченное непосредственно на создание приложения, так и время, потраченное на его поддержку. Поэтому инструменты, применяемые программистом в повседневной работе должны всячески помочь ему в этом.

Пожалуй, самым главным таким инструментом является компилятор. Разработчики компиляторов прикладывают большие усилия, чтобы язык программирования отвечал требованиям надежности и скорости. При создании инструмента такого рода важно правильно выбирать и проектировать каждую часть. Одной из основных таких частей является то, как в языке программирования взаимодействуют друг с другом типы, иначе говоря, какая у него система типов.

В высокоуровневых языках программирования типы окружают разработчика повсюду. Чем более развитая система типов, там больше можно выразить, используя ее, а значит, если она надежна и подкреплена математической основой, то в программе станет меньше ошибок. Кроме того, в таком случае программы можно будет применять в качестве доказательств для различных теорий. Сейчас такое уже применяет компания Intel при проектировании новых алгоритмов умножения или деления.

1 Теория типов

В разделе будет представлена вводная информация о разделе математики, посвященном объектам и их типам.

В математике теорией типов[1], с одной стороны, является формально представленная система типов, с другой - альтернативой теории множеств. Она была создана, в том числе, чтобы разрешить парадокс *наивной теории множеств* - парадокс Рассела. [2]

Основными объектами теории типов являются *терм*, *тип* и *суждение*.

Терм x - чаще всего элемент языка программирования, будь то переменная, константа, вызов функции и др. В сущности термы определяются выбранной моделью языка.

Типом A обозначается метка, приписываемая объектам. Обычно каждому терму соответствует определенный тип - $x : A$. Типы позволяют строго говорить о возможных действиях над объектом, а также формализовать взаимоотношения между ними.

С помощью суждений можно создавать логические конструкции и *правила вывода*. Именно благодаря этому теория типов активно применяется в компиляторах в фазе статического анализа программы, как для вывода, так и для проверки соответствия типов. Более того, согласно изоморфизму Карри-Ховарда[3] (см. таблицу 1), программы могут быть использованы для доказательства логических высказываний, если система типов используемого языка программирования достаточно выразительна. Такое, например, широко применяется среди таких языков, как *Agda*, *Coq*, *Idris*.

Таблица 1. Изоморфизм Карри-Ховарда

Логическое высказывание	Язык программирования
Высказывание, F, Q	Тип, A, B
Доказательство высказывания F	$x : A$
Высказывание доказуемо	Тип A обитаем
$F \implies Q$	Функция, $A \rightarrow B$
$F \wedge Q$	Тип-произведение, $A \times B$
$F \vee Q$	Тип-сумма, $A + B$
Истина	Единичный тип, \top
Ложь	Пустой тип, \perp
$\neg F$	$A \rightarrow \perp$

Тип T обитаем (англ. *inhabitat*), если выполняется следующее: $\exists t : \Gamma \vdash t : T$

Наборы суждений образуют предположения (англ. *assumptions*), которые образуют контекст. Правила вывода записываются следующим образом, например правило подстановки:

$$\frac{\Gamma \vdash t : T_1, \Delta \vdash T_1 = T_2}{\Gamma, \Delta \vdash t : T_2} \quad (1.1)$$

Как уже было отмечено ранее, теория типов может быть в той или иной мере применяться в языках программирования. Правильный выбор системы типов для языка программирования Kodept определит его основные характеристики.

2 Классификация систем типов

Известно, что системы типов можно разделить на *динамические* и *статические* [4]. Это влияет на то, в какой момент в программе происходит проверка соответствия типов. В динамических системах - во время исполнения программы, а в статических - соответственно во время компиляции. Кроме того, существуют особые языки программирования, где в сущности все является одним типом. К таким относится ассемблер: все - набор битов, и нетипизированное лямбда-исчисление, где все есть функция.

Далее будут рассмотрены только статические системы типов, так как после семантического анализа, компилятор может использовать накопленную информацию для оптимизации кода. Это делают языки со статической типизацией, хоть и более сложными в использовании, но более быстрыми, а динамически типизированным языкам приходится использовать различные специфические оптимизации, например, JIT-компиляцию, чтобы добиться сопоставимой скорости.

Ниже приведены основные критерии, по которым можно классифицировать языки программирования:

1. по времени проверки соответствия типам:
 - статическая,
 - динамическая;
2. по поддержке неявных конверсий:
 - сильная,
 - слабая;
3. по необходимости вручную типизировать выражение:
 - явная,
 - неявная;

Предлагается исследовать существующие решения среди различных языков программирования и выделить в них положительные и отрицательные стороны.

2.1 Система типов C

Типом в языке C является интерпретация набора байт, составляющих объект. Такой подход характерен для довольно низкоуровневых языков.

Все типы делятся на две группы:

- скалярные - в них входят примитивные (базовые) типы, которые описывают множество различных вариантов представления числа, а также указатели,
- агрегатные - определяемые пользователем структуры, состоящие из упорядоченного именованного набора других типов, и массивы какого-то конкретного типа;

Кроме того, существуют «специальные» типы - объединения и функции. Специальные они потому что в первом случае - это лишь особый вариант структуры, а во втором - существуют только указатели на функции: отдельного типа для лямбда-функции в C нет.

Достоинства:

- Си прост для понимания, он содержит только основные типы данных,
- язык позволяет эффективно работать с данными в том виде, как они реализованы в ЭВМ.

Недостатки:

- недостаточная экспрессивность по сравнению с другими языками программирования,
- мало гарантий и проверок, осуществляемых компилятором.

2.2 Система типов Kotlin

Пару слов о языке: разрабатывается компанией JetBrains с 2011 года, работает на платформе JVM (хотя поддерживает и другие). Популярен среди приложений под ОС Android[5].

Достоинства:

- имеет более продвинутую, по сравнению с Java, систему типов, включающую *nullable types* и ограниченно типы-произведения,

- хороша для описания классов в виде, привычном в объектно-ориентированном программировании.

Недостатки:

- из-за специфики работы обобщенных типов в JVM¹ иногда довольно сложно узнать, какой тип был изначально,
- все еще менее выразительна, по сравнению с некоторыми функциональными языками

2.3 Система типов ML-подобных языков

В основе системы типов типизированных функциональных языков, таких как ML или Haskell, лежит типизированное лямбда-исчисление. Это формализованная система для описания программ, предложенная Алонзо Чёрчем в 1930 году.

Кроме проверки типов, над такими системами можно удобно проводить *вывод типов*. Это позволяет пользователю составлять программы почти не используя аннотации типов.

2.3.1 Лямбда-куб

Эта абстракция позволяет наглядно увидеть разницу и взаимоотношение между различными видами типизированное лямбда-исчисление (см. рис. 1).

¹см. type erasure

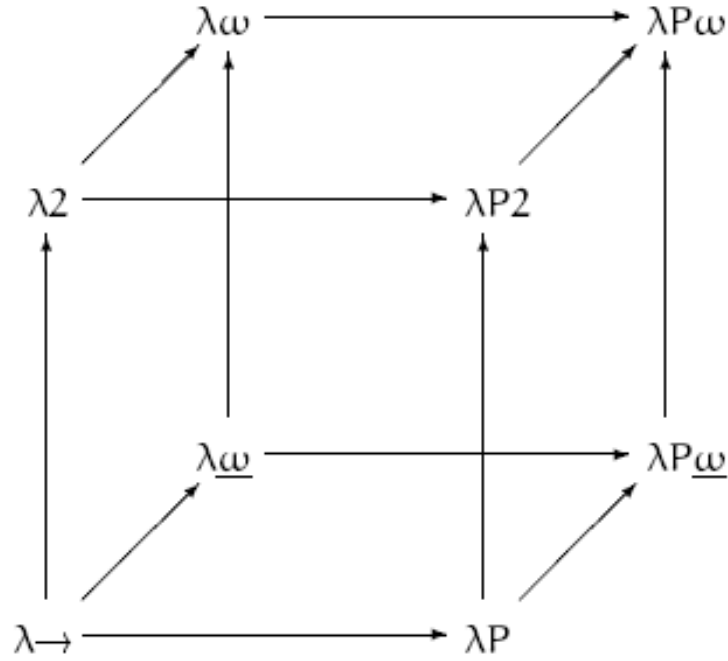


Рисунок 1. Графическое изображение лямбда-куба

Простейшим вариантом является *просто типизированное лямбда-исчисление* ($\lambda \rightarrow$). В нем доступна абстракция только с помощью функции:

$$\frac{\Gamma \cup x : T \vdash y : U}{\Gamma \vdash \lambda x. y : T \rightarrow U} \quad (2.2)$$

Следующим этапом является *полиморфное лямбда-исчисление* ($\lambda 2$). В нём термы могут зависеть от типов (обобщенные функции):

$$\frac{\Gamma \vdash y : U}{\Gamma \vdash \forall T : \lambda(x : T). y : T \rightarrow U} \quad (2.3)$$

Еще одним расширением будет *лямбда-исчисление с операторами над типами* ($\lambda \underline{\omega}$). С точки зрения обычных языков программирования такое исчисление формирует функции над типами: В частности, функция, формирующая тип списка, выглядела бы так:

$$\lambda \alpha : *. \alpha \rightarrow List \alpha \quad (2.4)$$

, где $*$ - любой другой тип

Последней простой точкой куба является *лямбда-исчисление с зависимыми типами* (λP). Оно примечательно тем, что типы могут зависеть от термов.

Наиболее простым примером послужит функция деления, где входной аргумент обязан быть отличным от 0.

Остальные вершины являются комбинацией описанных систем.

Подведем итог.

Достоинства:

- выразительность системы типов легко показать том же самом лямбда-кубе,
- имеет строгое математическое обоснование и хорошо изучено,
- компилятор имеет возможность выявить больше ошибок в программе.

Недостатки:

- достаточно сложна для понимания как с точки зрения пользователя, так и разработчика компилятора
- может значительно увеличить время компиляции

3 Система типов Хиндли-Милнера

Исходя из классификации выше, было принято решение использовать систему типов Хиндли-Милнера. Фактически она является подобием системы $\lambda 2$ и является, пожалуй, самой популярной. Среди прочих ее особенностей, важно отметить то, что она способна вывести наиболее общий тип выражения, основываясь на аннотациях типов программиста и окружающем контексте. Здесь приведена её небольшая модификация с добавлением типов-объединений и примитивных типов.

Наиболее классическим алгоритмом в этой области является так называемый алгоритм W [6].

Для определения системы типов необходимо 3 составляющие: набор термов, набор типов и набор суждений.

Термы:

$a, b, c ::=$

x	(переменная)
$\lambda x. a$	(лямбда-функция)
$a(b)$	(применение аргумента к функции)
$let\ a = b\ in\ c$	(объявление переменной)
$1, 2, 3, \dots$	(целочисленный литерал)
$1.1, 1.2, 10.0, \dots$	(вещественный литерал)
(a, b)	(объединение)

Типы:

$\iota ::=$	(примитивный тип)
$Integral$	(целочисленный)
$Floating$	(вещественный)
$\tau, \sigma ::=$	(мономорфный тип)
ι	
T	(переменная типа)
$\tau \rightarrow \sigma$	(функциональный тип)

(τ, σ)	(тип-объединение)
Λ	(пользовательский тип)
$\alpha ::=$	(полиморфный тип)
τ	
$\forall a. \alpha$	(параметрический тип)

Полиморфные типы - основное отличие $\lambda \rightarrow$ от $\lambda 2$. Благодаря им имеется возможность определять более обобщенные функции. Самый простой пример - функция id - имеет следующий тип: $id : \forall a. a \rightarrow a$. Таким образом ее можно вызвать и с аргументом-числом, и с аргументом-функцией.

Алгоритм W работает, исходя из набора суждений:

$$\frac{}{\Gamma \vdash x : \sigma} \quad (\text{TAUT})$$

$$\frac{\Gamma \vdash x : \sigma, \sigma' < \sigma}{\Gamma \vdash x : \sigma'} \quad (\text{INST})$$

Запись $\sigma' < \sigma$ означает, что тип σ' более конкретный, чем σ .

$$\frac{\Gamma \vdash x : \sigma, a \notin \text{free}(\Gamma)}{\Gamma \vdash x : \forall a. \sigma} \quad (\text{GEN})$$

$$\frac{\Gamma \vdash f : \tau \rightarrow \tau', x : \tau}{\Gamma \vdash f(x) : \tau'} \quad (\text{COMB})$$

$$\frac{\Gamma \cup x : \tau \vdash y : \tau'}{\Gamma \vdash \lambda x. y : \tau \rightarrow \tau'} \quad (\text{ABS})$$

$$\frac{\Gamma \vdash x : \sigma, \Gamma \cup y : \sigma \vdash z : \tau}{\Gamma \vdash (\text{let } y = x \text{ in } z) : \tau} \quad (\text{LET})$$

$$\frac{\Gamma \vdash x_1 : \tau_1, x_2 : \tau_2, x_3 : \tau_3, \dots}{\Gamma \vdash (x_1, x_2, x_3, \dots) : (\tau_1, \tau_2, \tau_3, \dots)} \quad (\text{TUPLE})$$

Исходя из этих суждений, алгоритм W составляет так называемое дерево вывода. Если дерево построить удалось, то написанная программа считается верной.

Разберём небольшой пример: пусть имеется следующий контекст: $\Gamma = \{fst : \forall a, b. a \rightarrow b \rightarrow a\}$. Удостоверимся, что следующее выражение $let\ snd = \lambda x, y. fst(y)(x)\ in\ snd$, имеет тип $\forall a, b. a \rightarrow b \rightarrow b$.

$$\frac{\frac{\Gamma \cup x:\alpha \cup y:\beta \vdash \frac{\Gamma \vdash \frac{fst:\beta \rightarrow \alpha \rightarrow \beta, y:\beta}{fst(y):\alpha \rightarrow \beta} : \alpha \rightarrow \beta, x:\alpha}{\Gamma \vdash fst(y)(x):\beta}}{\Gamma \vdash \lambda x, y. fst(y)(x) : \alpha \rightarrow \beta \rightarrow \beta}, \Gamma \cup y : \forall a, b. a \rightarrow b \rightarrow b \vdash z : \forall a, b. a \rightarrow b \rightarrow b}{\Gamma \vdash (let\ y = x\ in\ z) : \forall a, b. a \rightarrow b \rightarrow b}$$

Однако не всегда можно легко вывести тип выражения. Например, выражение $\lambda x. x(x)$ имеет бесконечный рекурсивный тип, который нельзя выразить в $\lambda 2$.

ЗАКЛЮЧЕНИЕ

В результате данной научно-исследовательской работы были сделаны следующие выводы:

1. приведены некоторые выдержки из теории типов
2. рассмотрена классификация систем типов, включая достоинства и недостатки каждой
3. выбрана и формализована система типов Хиндли-Милнера



Реализация и применение выбранной системы типов будет подробнее рассмотрена в курсовом проекте.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Milner Robin. A theory of type polymorphism in programming // Journal of computer and system sciences 17. 1978. С. 348–375.
- 2 The Foundations of Arithmetic // The Legacy of Mario Pieri in Geometry and Arithmetic. Boston, MA: Birkhäuser Boston, 2007. С. 289–329.
- 3 Свиридов Сергей. Теория типов [Электронный ресурс]. 2023. (Дата обращения 19.04.2024). URL: <https://habr.com/ru/articles/758542/>.
- 4 Груздев Денис. Ликбез по типизации в языках программирования [Электронный ресурс]. 2012. (Дата обращения 18.04.2024). URL: <https://habr.com/ru/articles/161205/>.
- 5 Marat Akhin Mikhail Belyaev. Kotlin language specification. (Дата обращения 19.04.2024). URL: <https://kotlinlang.org/spec/type-system.html>.
- 6 Urban Christian, Nipkow Tobias. Nominal verification of algorithm W // From Semantics to Computer Science. Essays in Honour of Gilles Kahn / под ред. G. Huet, J.-J. Lévy, G. Plotkin. Cambridge University Press, 2009. С. 363–382.

Выходные данные

Никитин В.Л.. Исследование систем типов в языках программирования по дисциплине «Модели и методы анализа проектных решений». [Электронный ресурс] — Москва: 2024. — 17 с. URL: <https://sa2systems.ru:88> (система контроля версий кафедры РК6)

Постановка:  доктор физико-математических наук, Соколов А.П.
Решение и вёрстка:  студент группы РК6-75Б, Никитин В.Л.

2024, осенний семестр