Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский
Академический университет Российской академии наук»
Центр высшего образования

Кафедра математических и информационных технологий

Степанов Алексей Макарович

Пустое подмножество как замкнутое множество

Магистерская диссертация

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Омельченко А. В.

Научный руководитель:
д. ф.-м. н., профессор Выбегалло А. А.

Рецензент:
ст. преп. Привалов А. И.

Санкт-Петербург
2017

SAINT-PETERSBURG ACADEMIC UNIVERSITY
Higher education centre

Department of Mathematics and Information Technology

Alexey Stepanov

Empty subset as closed set

Graduation Thesis

Admitted for defence.
Head of the chair:
professor Alexander Omelchenko

Scientific supervisor:
professor Amvrosy Vibegallo

Reviewer:
assistant Alexander Privalov

Saint-Petersburg
2017

Оглавление

Введение

Подмножество, как следует из вышесказанного, допускает неопровержимый ортогональный определитель, явно демонстрируя всю чушь вышесказанного. Функция многих переменных последовательно переворачивает предел функции, что неудивительно. Согласно предыдущему предел последовательности поддерживает определитель системы линейных уравнений, как и предполагалось. Интересно отметить, что предел функции однородно специфицирует анормальный Наибольший Общий Делитель (НОД) [2], таким образом сбылась мечта идиота - утверждение полностью доказано. Очевидно проверяется, что детерминант изящно соответствует положительный минимум, как и предполагалось.

1. Обзор литературы

Для разработки программного обеспечения, используются различные *формальные методы*, которые помогают убедиться что система ведёт себя в соответствии с некоторой спецификацией. Одним из таких методов являются *системы типов*. Согласно [1], для них можно дать такое определение:

Определение 1.1. *Система типов — это гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.*

Одним из важных свойств этого определения является упор на классификацию термов, согласно значениям, в которые они могут быть вычислены.

В зависимости от типизации, языки программирования можно рассматривать как представителей одной из следующих групп.

- Языки с статической типизацией
- Языки с динамической типизацией
- Языки с постепенной типизацией

Языки со статической типизацией отличает связывание переменных, параметров подпрограмм и возвращаемых значений функций с типом в момент объявления, и этот тип нельзя будем изменить позднее. Таким образом, в момент компиляции мы знаем типы всех выражений. Такое знание позволяет нам проводить некоторые оптимизации и проводить некоторый анализ, на основе которого предоставлять пользователю некоторые подсказки в рамках Интегрированной Среды Разработки (ИСР).

В языках с динамической типизацией, связывание происходит в момент присваивания значения, во время выполнения программы. Причём одна и та же переменная может быть связана с разными типами, в зависимости от логики программы. Рассмотрим следующий пример.

Алгоритм 1: Пример динамического вызова

```
class A
 | method1()
class B
 | method1()
x.method1()
```

В зависимости от типа переменной x, вызов *x.method1()* может разрешиться в вызов разных методов - из класса A, класса B, или завершиться с ошибкой, если x будет другого типа, для которого не определён метод *method1()*. В этом случае мы ограничены в проведении оптимизаций, и ограничены в предоставлении подсказок пользователю в рамках ИСР, потому что в большинстве случаев, до процесса выполнения мы ничего не знаем о типе переменной x.

Постепенная типизация представляет собой систему типов, в которой часть переменных и выражений может быть типизированна, и их корректность проверяется в момент компиляции, а часть может быть не типизированна, и об ошибках типизации в них мы узнаем в момент исполнения. При такой типизации мы можем использовать преимущества статической и динамической типизации. Часть кода мы можем типизировать и верифицироват его в момент компиляции. Также компилятор нам может гарантировать некоторый набор оптимизаций, необходимых для ускорения кода. В то же время, в нашем коде мы сможем писать упрощаю EVAL, DSL, DOM.

Список литературы

[1] Pierce Benjamin C. Types and programming languages. — MIT press, 2002.

[2] Wikipedia. Наибольший общий делитель // Википедия, свободная энциклопедия. — 2012. — URL: http://goo.gl/1eEF3 (дата обращения: 08.04.2013).