Acknowledgments

Thanks your peoples here.

Statement of Integrity

Resumo

Abstract em português.

Palavras-chave 3 a 5 palavras-chave, ordenadas alfabeticamente e separadas por vírgulas

Abstract

Your abstract here.

Keywords 3-5 keywords alphabetically ordered and comma-separated.

"We adore chaos because we love to produce order."

M. C. Escher

Contents

Chapter 1

Introduction

1.1 Motivation

There is no motivation, yet we need to write one.

1.2 Objectives

Formalise results about λ-calculus variants in *Coq*.

1.3 Document Structure

List your chapters here, with a very brief description of each one.

Chapter 2

Background

In the following chapter we will introduce some background to aid the reading of this dissertation. First, we introduce the λ-calculus and some basic knowledge around it. Then, we introduce some theory related to the mechanisation part of this work. These concepts are introduced and motivated by the task of formalising the λ-calculus system introduced.

2.1 Lambda Calculus

2.1.1 Syntax

[3] [1]

Definition 1 (λ-terms)**.** *The λ-terms are defined by the following grammar:*

$$M, N \ ::= \ x \mid (\lambda x.M) \mid (MN)$$

where x denotes any variable, typically in the range of x, y, z.

Notation. *We will assume the usual notation conventions on λ-terms:*

1. *Outermost parenthesis are omitted.*
2. *Multiple abstractions can be abreviated as $\lambda xyz.M$ instead of $\lambda x.(\lambda y.(\lambda z.M))$.*
3. *Multiple applications can be abreviated as MN_1N_2 instead of $(MN_1)N_2$.*

Definition 2 (Free variables)**.** *For every λ-term M, we recursively define the set of free variables in M, $FV(M)$, as follows:*

1. $FV(x) = \{x\}$
2. $FV(\lambda x.M) = FV(M) - \{x\}$
3. $FV(MN) = FV(M) \cup FV(N)$

When a variable occurring in a term is not free it is said to be bound.

Definition 3 (α-equality)**.** *We say that two λ-terms are α-equal when they only differ in the name of their bound variables.*

Remark. *The previous informal definition lets us take advantage of a variable naming convention. With this convention, the definition of substitution over λ-terms and meta-discussion of our syntax will be simplified. After defining the substitution operation we may introduce later a better and formal definition of this α-equality.*

Convention. *We will use the variable convention introduced in [1], using only λ-terms that are chosen (via α-equality) to have bound variables with different names from free variables.*

Definition 4 (Substitution)**.** *For every λ-term M, we recursively define the substitution of the free variable x by N in M, $M[x := N]$, as follows:*

1. $x[x := N] = N$
2. $y[x := N] = y$ *(when $x \neq y$)*
3. $(\lambda y.M)[x := N] = \lambda y.(M[x := N])$
4. $(M_1 M_2)[x := N] = (M_1[x := N])(M_2[x := N])$

Definition 5 (Compatible Relation)**.** *We say that a binary relation in λ-terms, R, is compatible if it satisfies:*

$$\frac{(M_1, M_2) \in R}{(\lambda x.M_1, \lambda x.M_2) \in R} \qquad \frac{(M_1, M_2) \in R}{(NM_1, NM_2) \in R} \qquad \frac{(M_1, M_2) \in R}{(M_1 N, M_2 N) \in R}$$

Definition 6 (α-conversion)**.**

Definition 7 (β-reduction)**.**

2.1.2 Types

[**?**]

Definition 8 (Simple Types)**.**

Definition 9 (Tipification Rules)**.**

2.2 Mechanising Meta-theory in Coq

The formalisation we aim at is dependent on the theory provided by the *Coq* proof assistant - the Calculus of Inductive Constructions. We will follow assuming a basic knowledge on *Coq* and its syntax to define inductive types and proof techniques.

2.2.1 How to Denote Variables?

If we were to formalise such a system like the λ-calculus introduced above, maybe we would create an inductive type like the following, in *Coq*.

```
Inductive term : Type :=
| Var (x: var)
| Lam (x: var) (t: term)
| App (s: term) (t: term).
```

The question that every similar definition imposes is the definition of the var type. Following the usual pen and paper aproach, this type would be something similar to a subset of the string type, where a variable is just a placeholder for a name.

Of course this is fine when dealing with pen and paper proofs and definitions. Even, we can take advantage of conventions, like the one referenced above by Barendregt. However, this is becomes rather exhausting when it comes to rigorously defining all this syntactical aspects.

2.2.2 De Bruijn Syntax

In the 1970s, de Bruijn started working on the *AUTOMATH* proof assistant and proposed a simplified syntax to deal with generic binders [2]. This approach is claimed to be good for meta-lingual discussion and for the computer and computer programme. In contrast, this syntax is farther away from the human reader.

The main idea is to treat variables as indices (represented by natural numbers) and to interpret these indices as the distance to the respective binder. This way, we do not decide the names for the bound variables in a term - we are obliged to use the specific indices that bind a variable to a certain binder.

Definition 10 (nameless λ-terms)**.** *The nameless λ-terms are defined by the following grammar:*

$$M, N \ ::= \ i \mid (\lambda.M) \mid (MN)$$

where i ranges over the natural numbers.

2.2.3 Explicit Substitutions and the Sigma Calculus

2.2.4 The Autosubst Library

Chapter 3

The Multiary Lambda Calculus and Its Canonical Fragment

3.1 The Multiary Lambda Calculus (λm)

3.1.1 A Sequent Calculus Fragment

3.2 The Canonical Fragment ($\vec{\lambda} m$)

3.2.1 ???

3.2.2 Conservativeness

3.3 Comments on the Formalisation

3.3.1 A Nested Inductive Type

3.3.2 Formalising a Subsystem

Chapter 4

The Isomorphic Fragment of Lambda Calculus in λm

Chapter 5

Discussion

Chapter 6

Conclusions

Final chapter, present your conclusions.

6.1 Summary of Findings

Highlight per-chapter content here, with a general conclusion in the end.

6.2 Future Work

There’s no lack of future work.

Appendix A

Example

This is what an Appendix looks like.

Bibliography

[1] H. P. Barendregt. *The lambda calculus*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, London, England, 2 edition, Oct. 1987.

[2] N. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972. ISSN 1385-7258. doi: https://doi.org/10.1016/1385-7258(72)90034-0. URL https://www.sciencedirect.com/science/article/pii/1385725872900340.

[3] J. R. Hindley. *Basic Simple Type Theory*. Cambridge University Press, Cambridge, July 1997.

[4] T. Winterhalter. *Formalisation and meta-theory of type theory*. PhD thesis, Université de Nantes, 2020.