Acknowledgments

Thanks your peoples here.

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

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

The following chapter introduces essential background to aid the reading of this dissertation. First, we introduce the λ-calculus and ~~some basic knowledge around it~~. Then, we delve into the theory on mechanisation of meta-theory, specifically in the context of our 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

[5] [3]

Definition (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 (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 (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 notion of α-equality, the definition of substitution over λ-terms and meta-discussion of our syntax will be simplified. After defining the substitution operation we will rigorously introduce the definition for α-conversion.*

Convention. *We will use the variable convention introduced in [3]. Every λ-term that we refer from now on is chosen (via α-equality) to have bound variables with different names from free variables.*

Definition (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. $y[x := N] = y$, with $x \neq y$
3. $(\lambda y.M_1)[x := N] = \lambda y.(M_1[x := N])$
4. $(M_1 M_2)[x := N] = (M_1[x := N])(M_2[x := N])$

Remark. *Is is important to notice that by variable convention, the substitution operation described is capture-avoiding - bound variables will not be substituted ($x \in FV(M)$) and the free variables in N will not be affected by the binders in M, as they are chosen to have different names.*

Definition (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 (α-conversion)**.** *Consider the following binary relation on λ-terms:*

$$\lambda x.M =_\alpha \lambda y.M[x := y]. \tag{α}$$

We define the α-conversion as the least compatible, reflexive and transitive relation that satisfies (α).

Definition (β-reduction)**.** *Consider the following binary relation on λ-terms:*

$$(\lambda x.M)N \rightarrow_\beta M[x := N]. \tag{β}$$

We define the β-reduction as the least compatible relation that satisfies (β).

2.1.2 Types

[2]

Definition (Simple Types)**.** *The simple types are defined by the following grammar:*

$$A, B, C ::= p \mid (A \supset B)$$

where p denotes any atomic variable, typically in the range of p, q, r.

Notation. *We will assume the usual notation conventions on simple types:*

1. *Outermost parenthesis are omitted.*

2. *Types associate to the right. Therefore, the type* $A \supset (B \supset C)$ *may often be written simply as* $A \supset B \supset C$*.*

Definition (Context)**.** *A context,* $\Gamma, \Delta, \ldots$*, is a partial function from the variables of* λ*-terms to simple types.*

Notation.

1. *We may often think of the partial function of as the set of pairs* (x, A) *written as* $x : A$*.*

2. *We will also simplify the set notation of contexts as follow:*

$$\begin{aligned} &\mapsto \{\} \\ x : A &\mapsto \{x : A\} \\ x : A, \Gamma &\mapsto \{x : A\} \cup \Gamma \end{aligned}$$

Definition (Typing Rules for λ-terms)**.** *A type-assignment or sequent is a triple,* $\Gamma \vdash M : A$*, that is inductively defined by the following inference rules (or typing rules):*

$$\frac{}{x : A, \Gamma \vdash x : A}\ \textit{Var} \qquad \frac{x : A, \Gamma \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ \textit{Abs} \qquad \frac{\Gamma \vdash M : A \supset B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ \textit{App}$$

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, we would probably 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 approach, this type would be a subset of a 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. To simplify this, we can even take advantage of conventions, like the one referenced above by Barendregt. However, this variable definition can get rather exhausting when it comes to rigorously define all this syntactical aspects and substitution operations.

There are several alternatives described in the literature of mechanisation of meta-theory. This topic of binding was even proposed in the POPLmark challenge [**?**] as a way to discuss the potential of proof assistants.

The *AUTOSUBST* library for the *Coq* proof assistant stood out as a great solution for our case. This library uses a combination of de Bruijn indices and explicit substitutions to ~~tackle this "problem"~~.

2.2.2 De Bruijn Syntax

[4] [6]

In the 1970s, de Bruijn started working on the *AUTOMATH* proof assistant and proposed a simplified syntax to deal with generic binders [4]. This approach is claimed to be good for meta-lingual discussion and for the computer and computer programme. In contrast, this syntax is further 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. ~~Therefore, we will call these λ-terms the nameless λ-terms.~~

Definition (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.

Remark. *Nameless λ-terms have no α-conversion since there is no freedom to choose the names of bound variables.*

2.2.3 Explicit Parallel Substitutions

[6]

We refer to explicit substitutions [1] when the substitution operation is part of the syntax calculus. Often, substitutions are dealt at a meta level. However, a definition of capture-avoiding substitution over nameless λ-terms will require some manipulations over these explicit objects [4].

Having a reserved syntax for substitutions motivates us to generalise this notion. Previously, the substitution object was a term that was going to take the place of some free variable. A parallel substitution generalizes this concept to replace every free variable.

Now, we present a calculus with explicit substitutions. We can see these explicit substitutions as sequences of terms, $\sigma = (M_i)_{i \in \mathbb{N}}$, where the term M_i will replace the index i.

Definition ($\lambda\sigma$-terms)**.** *The (nameless) $\lambda\sigma$-terms are defined by the following grammar:*

$$M, N ::= \ i \mid (\lambda.M) \mid (MN) \mid M[\sigma]$$
$$\sigma, \tau ::= \ id \mid \uparrow \mid M \cdot \sigma \mid \sigma \circ \tau$$

Definition (Reductions over substitutions)**.**

Chapter 3

The Multiary Lambda Calculus and Its Canonical Fragment

3.1 The Multiary Lambda Calculus (λm)

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

$$\begin{aligned} t, u \ &::= \ x \mid \lambda x.t \mid t(u, l) \\ l \ &::= \ [] \mid u :: l \end{aligned}$$

Definition (Concatenation of λm-lists). *The concatenation of two λm-lists, $l + l'$ is defined as follows:*

1. $[] + l' = l'$
2. $(u :: l) + l' = u :: (l + l')$.

Definition (Substitution for λm-terms). *The substitution over a λm-term is mutually defined with the substitution over a λm-list as follows:*

1. $x[x := v] = v$
2. $y[x := v] = y$, *with* $x \neq y$
3. $(\lambda y.t)[x := v] = \lambda y.(t[x := v])$
4. $t(u, l)[x := v] = t[x := v](u[x := v], l[x := v])$
5. $([])[x := v] = []$
6. $(u :: l)[x := v] = u[x := v] :: l[x := v]$.

Definition (Reduction rules for λm-terms).

$$(\lambda x.t)(u, []) \to t[x := u] \qquad (\beta_1)$$

$$(\lambda x.t)(u, v :: l) \to t[x := u](v, l) \qquad (\beta_2)$$

$$t(u, l)(u', l') \to t(u, l + (u' :: l')) \qquad (h)$$

Definition (Typing Rules for λm-terms).

$$\frac{}{x : A, \Gamma \vdash x : A}\ \textit{Var} \qquad \frac{x : A, \Gamma \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\ \textit{Abs}$$

$$\frac{\Gamma \vdash t : A \supset B \quad \Gamma \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma \vdash t(u, l) : C}\ \textit{mApp}$$

$$\frac{}{\Gamma; A \vdash [] : A}\ \textit{Nil} \qquad \frac{\Gamma \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash u :: l : C}\ \textit{Cons}$$

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

Definition ($\vec{\lambda}m$-terms)**.** *The $\vec{\lambda}m$-terms are defined by the following grammar:*

$$\begin{aligned} t, u \; &::= \; x \mid \lambda x.t \mid x(u, l) \mid (\lambda x.t)(u, l) \\ l \; &::= \; [] \mid u :: l \end{aligned}$$

Definition (Substitution for λm-terms)**.** *The substitution over a $\vec{\lambda}m$-term is mutually defined with the substitution over a $\vec{\lambda}m$-list as follows:*

1. $x[x := v] = v$
2. $y[x := v] = y$, *with* $x \neq y$
3. $(\lambda y.t)[x := v] = \lambda y.(t[x := v])$
4. $x(u, l)[x := v] = v@(u[x := v], l[x := v])$
5. $y(u, l)[x := v] = y(u[x := v], l[x := v])$, *with* $x \neq y$
6. $(\lambda y.t)(u, l)[x := v] = (\lambda y.t[x := v])(u[x := v], l[x := v])$
7. $([])[x := v] = []$
8. $(u :: l)[x := v] = u[x := v] :: l[x := v]$.

Definition (Typing Rules for $\vec{\lambda}m$-terms)**.**

$$\frac{}{x : A, \Gamma \vdash x : A}\;\textit{Var} \qquad \frac{x : A, \Gamma \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\;\textit{Abs}$$

$$\frac{\Gamma, x : A \supset B \vdash u : A \qquad \Gamma, x : A \supset B; B \vdash l : C}{\Gamma, x : A \supset B \vdash x(u, l) : C}\;\textit{VarApp}$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A \qquad \Gamma; B \vdash l : C}{\Gamma \vdash (\lambda x.t)(u, l) : C}\;\textit{LamApp}$$

$$\frac{}{\Gamma; A \vdash [] : A}\;\textit{Nil} \qquad \frac{\Gamma \vdash u : A \qquad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash u :: l : C}\;\textit{Cons}$$

3.3 Formalised Results

Theorem (Subject Reduction)**.**

Theorem (Conservativeness)**.**

3.4 Comments on the Formalisation

3.4.1 A Nested Inductive Type

3.4.2 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] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 31–46, 1989.

[2] H. Barendregt, W. Dekkers, and R. Statman. *Perspectives in logic: Lambda calculus with types*. Perspectives in logic. Cambridge University Press, Cambridge, England, June 2013.

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

[4] 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.

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

[6] S. Schäfer, T. Tebbi, and G. Smolka. Autosubst: Reasoning with de bruijn terms and parallel substitutions. In C. Urban and X. Zhang, editors, *Interactive Theorem Proving*, pages 359–374, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22102-1.