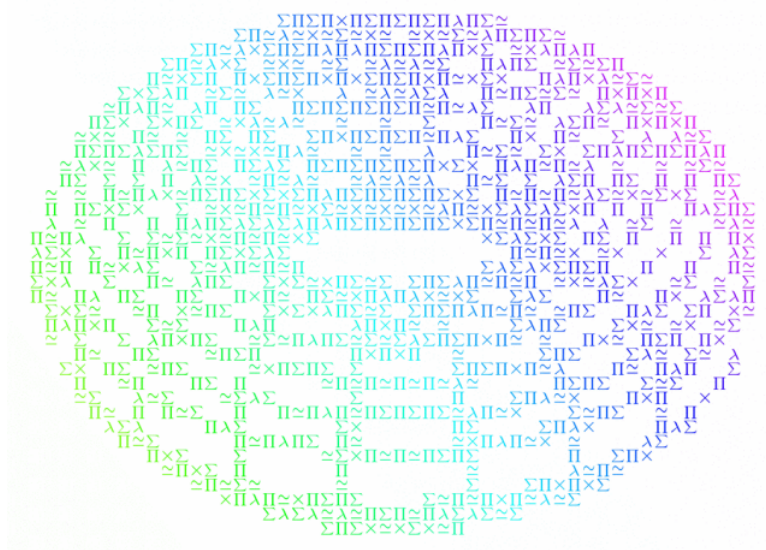


# Quotient Containers in Homotopy Type Theory

Jonathan E. Sherry

*School of Computer Science and Information Technology*  
**University of Nottingham**



Submitted May 2013 in partial fulfilment of the conditions of  
the award of the degree of BSc (Hons) Computer Science.

I hereby declare that this dissertation is all my own work,  
except as indicated in the text:

Signature \_\_\_\_\_

Date \_\_\_\_/\_\_\_\_/\_\_\_\_

# Contents

<b>I</b>	<b>Introduction</b>	<b>2</b>
1	Abstract	3
2	Acknowledgements	4
3	Motivation	5
<b>II</b>	<b>Research</b>	<b>6</b>
<b>4</b>	<b>Martin-Löf Type Theory</b>	<b>7</b>
4.1	Introduction . . . . .	7
4.2	Basic Constructions . . . . .	8
4.3	Type Connectives . . . . .	8
4.3.1	Finite Types . . . . .	8
4.3.2	Product Types . . . . .	8
4.3.3	Coproduct Types . . . . .	8
4.3.4	$\Pi$ -Types . . . . .	9
4.3.5	$\Sigma$ -Types . . . . .	9
4.3.6	Identity Types . . . . .	9
4.3.7	Inductive Types . . . . .	10
4.3.8	Universes . . . . .	10
4.3.9	Propositions as Types . . . . .	10
4.4	Equality proofs form a groupoid? . . . . .	10
<b>5</b>	<b>Homotopy Type Theory</b>	<b>11</b>
5.1	Introduction . . . . .	11
5.2	Key Concepts in Homotopy Type Theory . . . . .	12
5.2.1	Types and their Inhabitants . . . . .	12
5.2.2	Equality . . . . .	12
5.2.3	Functions . . . . .	13
5.2.4	Dependent types . . . . .	13
5.2.5	Higher Inductive Types . . . . .	13
5.3	Summary . . . . .	13
<b>6</b>	<b>Containers</b>	<b>15</b>
<b>7</b>	<b><math>\delta</math> for Data</b>	<b>16</b>

8	Agda	17
III	Quotient Containers in Homotopy Type Theory	18
9	Implementation	19
IV	Conclusions	20
10	Evaluation	21
11	Conclusion	22
12	Further Work	23
	Bibliography	24
	Appendix	25
13	Appendix	25

# Part I

## Introduction

# Chapter 1

## Abstract

lorem ipsum pgiuhfsgpoiuhygpoughpouh

## Chapter 2

## Acknowledgements

## Chapter 3

### Motivation

# Part II

## Research



## Chapter 4

# Martin-Löf Type Theory

### 4.1 Introduction

Martin-Löf type theory is a form of constructive, intuitionistic type theory that is both a programming language and an alternative foundation of mathematics. Devised by Swedish mathematician and philosopher Per Martin-Löf in 1972, Martin-Löf type theory (MLTT) serves to provide a set of formal rules and type-theoretic connectives to perform mathematical reasoning. MLTT is said to be *constructive* and *intuitionistic* as it replaces the classical concept of Truth, with the notion of constructive provability. That is, construction of a mathematical object is proof of its existence. In MLTT, objects are classified by the basal notion of a *type* primitive — objects can have a certain type and are said to inhabit that type. These structured types can be used to provide a specification for the elements within it, providing a means to reason about objects of that type. For example, from a type:

$$A \rightarrow B$$

, the type of functions from type  $A$  to type  $B$ , instructions on how to construct an object are implicitly known. In this case, an object of the type  $A \rightarrow B$  is formed from an object of type  $B$ , parametrised by an object of type  $A$ .

The rigorous rules and predictable nature of objects and types within MLTT afford it strength in formal reasoning. This strength has led to its implementation as a base for a number of languages and proof assistants including Agda (*see chapter 8*), Epigram and Coq, among others.

This chapter constitutes a brief introduction to key concepts and ideas within Martin-Löf type theory required for the rest of the text.

## 4.2 Basic Constructions

$A$ Type	Declaration of a type $A$ .
$a : A$	Object $a$ exists and inhabits type $A$ .
$A \equiv B$	Type $A$ is judgementally equal to type $B$ .
$a \equiv b : A$	$a$ and $b$ are judgementally equal terms of type $A$ .
$\Gamma \vdash A$	$A$ is well-formed in the context $\Gamma$ .
$\Gamma \vdash a : A$	$a$ is well-formed term of the type $A$ in the context $\Gamma$ .

## 4.3 Type Connectives

### 4.3.1 Finite Types

Finite types are those that have a strictly finite, enumerable number of inhabitants. Examples include:

- the **0**-type. Otherwise known as the *empty* or *bottom* type, it has no inhabitants and invoking the Curry-Howard isomorphism<sup>1</sup>, it represents False. An inhabitant of the **0**-type may be referred to as a *contradiction* as there is no way to prove a contradiction and nor can the **0**-type be constructed.
- the **1**-type. Also called the *unit* type. It has only one inhabitant and invoking the Curry-Howard isomorphism, it represents True.
- the **2**-type. Intended to have exactly two inhabitants:  $0_2$  and  $1_2$ , it could be constructed as  $\mathbf{1} + \mathbf{1}$ . This is the type of Booleans.

### 4.3.2 Product Types

Given types  $A, B : \mathcal{U}$ ,  $A \times B : \mathcal{U}$  is the type of their cartesian product where elements of  $A \times B$  are pairs  $(a, b) : A \times B$  such that  $a : A$  and  $b : B$ .

### 4.3.3 Coproduct Types

Given types  $A, B : \mathcal{U}$ ,  $A + B : \mathcal{U}$  is the type of their coproduct. Inhabitants of  $A + B$  can be constructed with  $\text{inl}(a) : A + B$  for  $a : A$  or  $\text{inr}(b) : A + B$  for  $b : B$ .

---

<sup>1</sup>

Discovered by Haskell Curry and William Alvin Howard, the Curry-Howard isomorphism states that there is an equivalence between logic and programming. More formally, there is a syntactic analogy between formal logic and computational calculi.

### 4.3.4 $\Pi$ -Types

$\Pi$ -types, also known as dependent product type, is a type whose inhabitants are functions whose codomain is dependent on the domain to which the function is applied. It can be regarded as the cartesian product over a type. Given a type  $A : \mathcal{U}$  and a family  $B : A \rightarrow \mathcal{U}$ , we can construct the type of dependent functions:

$$\prod_{(x:A)} B(x) : \mathcal{U}$$

For a constant family  $B$ , this is judgementally equal to a function  $A \rightarrow B$ . Polymorphic functions are an example of a dependent type element. A polymorphic function would be of the type:

$$\prod_{(A:\mathcal{U})} A \rightarrow B$$

Invoking the Curry-Howard isomorphism,  $\Pi$ -types represent implication and universal quantification.

### 4.3.5 $\Sigma$ -Types

$\Sigma$ -types, or dependent pair types, are types where the latter component of a pair depends on the former. It is analogous to an indexed sum over a given type. Given a type  $A : \mathcal{U}$  and a family  $B : A \rightarrow \mathcal{U}$ , we can construct the type of dependent pair functions:

$$\sum_{(x:A)} B(x) : \mathcal{U}$$

For a constant family  $B$ , this is judgementally equal to the cartesian product type:  $(A \times B)$ . Invoking the Curry-Howard isomorphism,  $\Sigma$ -types represent conjunction and existential quantification.

### 4.3.6 Identity Types

Identity or equality types are of the form  $a =_A b$  such that  $a : A$  and  $b : A$ . Given a type  $A : \mathcal{U}$  and elements  $a, b : A$  we can form type  $a =_A b : \mathcal{U}$  in the same universe. There is only one inhabitant of this type — the proof of reflexivity:

$$\mathbf{refl} : \prod_{(a:A)} (a =_A a)$$

$\mathbf{refl}$  states that all elements in  $A$  are equal to themselves. This means that if  $a$  and  $b$  are judgementally equal, we also have a witness of  $\mathbf{refl}$ ,  $\mathbf{refl}_a : (a =_A b)$ .

### 4.3.7 Inductive Types

Inductive types are those that includes a constructor that encodes how to create new elements of the type. Normally, there will be a base case and an inductive case. A good example of this is an inductive definition of the natural numbers. Elements of type  $\mathbb{N} : \mathcal{U}$  are constructed with  $0 : \mathbb{N}$  and  $successor : \mathbb{N} \rightarrow \mathbb{N}$ .

### 4.3.8 Universes

A universe,  $\mathcal{U}$ , is a type whose elements are types. To avoid Russell's paradox<sup>2</sup>, universes are ordered in a heirarchy. That is,  $\mathcal{U}_0 : \mathcal{U}_1, \mathcal{U}_1 : \mathcal{U}_2 \dots$ . Universes are *cumulative*, meaning all elements of the  $i^{th}$  universe are also elements of the  $(i+1)^{th}$  universe. When we declare a type, we say implicitly that it inhabits some universe  $\mathcal{U}_i$ .

### 4.3.9 Propositions as Types

Using the types above, it is possible to construct types that represent logical propositions, that is, logical sentences with a truth value, as such:

Logic	Type Theory
True	<b>1</b>
False	<b>0</b>
$A$ and $B$	$A \times B$
$A$ or $B$	$A + B$
If $A$ then $B$	$A \rightarrow B$
$A$ iff $B$	$(A \rightarrow B) \times (B \rightarrow A)$
Not $A$	$A \rightarrow \mathbf{0}$
For all $x : A, P(x)$ is true	$\prod_{x:A} P(x)$
There exists $x : A$ such that $P(x)$ is true	$\sum_{x:A} P(x)$

## 4.4 Equality proofs form a groupoid?

<sup>2</sup> In 1901, Bertrand Russell used this paradox to derive a contradiction in Cantor's naive set theory. The paradox is as follows: "Let R be the set of all sets which are not members of themselves. Then R is neither a member of itself nor not a member of itself." [9] Type theory was introduced by Russell as a solution to this problem.

## Chapter 5

# Homotopy Type Theory

### 5.1 Introduction

Homotopy type theory is a new branch of mathematics that came about after a special year on *Univalent Foundations of Mathematics* during 2012 and 2013 at The Institute for Advanced Study, Princeton. Organised by Vladimir Voevodsky, Thierry Coqand and Steve Awodey, the *Univalent Foundations* program sought to explore implications of Voevodsky’s *Univalence Axiom*,  $(A = B) \simeq (A \simeq B)$ , that is “identity is equivalent to equivalence. In particular, one may say that ‘equivalent types are identical’.”[3].

Homotopy type theory, an amalgam of type theory; homotopy theory and higher dimensional category theory, is *intentional*, *dependent* type theory with the *Univalence Axiom* forming the basis of its *structural identity principle*. Formally, this means that

$$(A \cong_C B) \rightarrow Id_C(A, B)$$

where  $C$  is the type of structured objects with arbitrary signature,  $(A \cong_C B)$  is the type of isomorphisms between  $A$  and  $B$  and  $Id_C(A, B)$  is the type of objects that witness the identification of  $A$  with  $B$ . In other words, ***Isomorphic structures can be identified***.

The interpretation of types as structured objects also means that it is possible to apply type-theoretic logic to other structured objects — namely topological objects, leading to a homotopical interpretation and the ability to reason about homotopy theory using type theory. This homotopical interpretation of types coupled with the *Univalence Axiom* is the basis of homotopy type theory.

## 5.2 Key Concepts in Homotopy Type Theory

The fundamental idea behind homotopy type theory is the marriage between Martin-Löf type theory and homotopical notions. The following uses the languages of type theory, category theory and homotopy theory to present concepts in Homotopy type theory.

### 5.2.1 Types and their Inhabitants

#### Types as spaces

In a homotopical interpretation, types are regarded as topological spaces and terms of a type are represented by points in a space.

#### Types as $\infty$ -Groupoids

In algebraic topology, a (pointed<sup>1</sup>) space has an associated group that stores information about the ‘basic shape’ of the space. More formally, it stores information on when two paths, beginning and ending at a basepoint can be continuously deformed onto one another. Naturally, it follows that for homeomorphic spaces, this group will be the same. This group is known as the *fundamental group*.

In homotopy theory, every space has a *fundamental  $\infty$ -groupoid* whose  $k$ -morphisms (morphisms between morphisms at level  $k$ ) are the  $k$ -dimensional paths in the space. Homotopy type theory views types as having the structure of this *fundamental  $\infty$ -groupoid*. This can be seen through iterating the identity type in the following way: given an identity type  $a =_A b$  we can consider the type of identified elements :  $c =_{(a=_A b)} d$ , that is, identifications between identifications and so on.

This structure is analogous to continuous paths of a space and the homotopies between them, in other words, an  $\infty$ -groupoid. This groupoid model of Martin-Löf type theory has far reaching implications in higher category theory as groupoids can be modelled as small categories in which all morphisms are isomorphic.

### 5.2.2 Equality

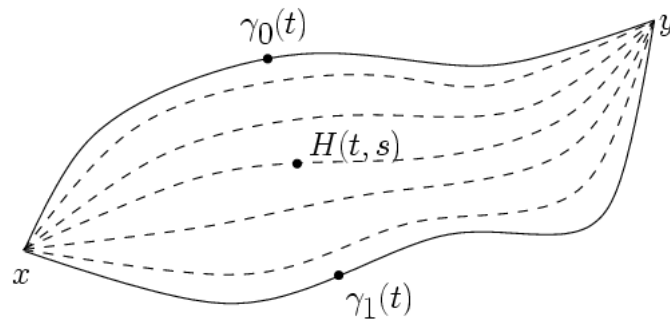
Given the above, it is consistent to interpret identity types as paths. Thus, for an element  $p : a =_A b$ , a path  $p : a \rightsquigarrow b$  is said to exist between  $a$  and  $b$  in the

---

<sup>1</sup>

A pointed topological space is a space  $X$  such that it has a distinct basepoint  $x_0 \in X$ . This basepoint will often dictate the *fundamental group*.

space of  $A$ . For two elements of the same identity type i.e. for  $p, q : a =_A b$ , paths are said to be parallel as they have the same start and end points. Thus, it follows that an element of the type  $c =_{(a=_A b)} d$  represents a homotopy — or a morphism between morphisms. This can also be thought of as a 2-dimensional path in  $A$ . Similarly, the next iteration, the type of identifications between identifications between identifications can be thought of as a 3-dimensional path or a morphism between morphisms between morphisms.



*Homotopy between two curves: the curves  $\gamma_0$  and  $\gamma_1$  are homotopic by the homotopy  $H$*

In the image above,  $\gamma_0, \gamma_1$  could be considered of the type  $x = y$  for some space.

### 5.2.3 Functions

Functions are represented by continuous mappings between spaces. For example, a function  $f : A \rightarrow B$  represents a continuous mapping from space  $A$  to space  $B$ . Functions can be identified if they are homotopic.

Functions also behave functorially on paths, that is, all functions are continuous and preserve paths.

### 5.2.4 Dependent types

### 5.2.5 Higher Inductive Types

Multisets as a HIT

## 5.3 Summary

The following table summarises the different interpretations of analogous concepts in the languages of type theory, logic, Zermelo-Fraenkel set theory and homotopy type theory:

Type Theory	Logic	ZFC	Homotopy Theory
$\mathbf{0}$	$\perp$	$\emptyset$	$\emptyset$
$\mathbf{1}$	$\top$	$\{\emptyset\}$	$*$
$A$	proposition	set	topological space
$a : A$	proof	element	point
$B(x)$	predicate	family of sets	fibration
$b(x) : B(x)$	conditional proof	family of elements	section
$Id_A$	equality	$\{(x, x)   x \in A\}$	path space $A^I$
$A + B$	$A \vee B$	disjoint union	coproduct
$A \times B$	$A \wedge B$	set of pairs	product space
$A \rightarrow B$	$A \Rightarrow B$	set of functions	function space
$\sum_{(x:A)} B(x)$	$\exists x : A, B(x)$	disjoint sum	total space
$\prod_{(x:A)} B(x)$	$\forall x : A, B(x)$	product	space of sections



# Chapter 6

## Containers

## Chapter 7

### $\delta$ for Data

# Chapter 8

## Agda

based on mltt

## Part III

# Quotient Containers in Homotopy Type Theory

## Chapter 9

# Implementation

## Part IV

# Conclusions

## Chapter 10

## Evaluation

## Chapter 11

## Conclusion



## Chapter 12

### Further Work

# Bibliography

- [1] Per Martin-Löf, *Intuitionistic Type Theory*, 1980
- [2] Bengt Nordström, Kent Petersson, Jan M. Smith, *Programming in Martin-Löf's Type Theory, An Introduction*, 1990
- [3] The Univalent Foundations Program, Institute for Advanced Study, *Homotopy type theory: Univalent foundations of mathematics*. 2013.
- [4] Michael Abbott, Thorsten Altenkirch, Neil Ghani, *Constructing Strictly Positive Types*. 2004.
- [5] Michael Abbott, Thorsten Altenkirch, Neil Ghani, Connor McBride,  *$\delta$  for Data, Differentiating Data Structures*. 2005.
- [6] Hakon Robbestad Gylterud, *Symmetric Containers (MSc Thesis)*. 2005.
- [7] Thorsten Altenkirch, Paul Levy, Sam Staton, *Higher Order Containers* 2010.
- [8] <http://homotopytypetheory.org/>
- [9] <http://mathworld.wolfram.com/RussellsAntinomy.html>

## Chapter 13

## Appendix