Bachelor thesis

Martin Löf's J-Rule

by

Lennard Götz

July 16, 2018

supervised by

Dr. Iosif Petrakis

Faculty for mathematics, informatics and statistics of
Ludwig-Maximilian-Universität München

Statement in Lieu of an Oath

Ich versichere hiermit, dass ich die vorgelegte Bachelorarbeit eigenständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen verwendet und die den benutzten Quellen entnommenen Passagen als solche kenntlich gemacht habe. Diese Bachelorarbeit istt in dieser oder einer ähnlichen Form in keinem anderen Kurs und/oder Studiengang als Studien- oder Prüfungsleistung vorgelegt worden.
Hiermit stimme ich zu, dass die vorliegende Arbeit von dem Prüfer in elektronischer Form mit entsprechender Software über prüft wird.

Contents

1 Introduction **5**

2 Function types **5**
2.1 Dependent function types ($\Pi - types$) 6
2.2 Product type ($A \times B$) 6
2.3 Dependent pair types ($\sum -types$) 7
2.4 Coproduct type (A + B) 7

3 Identity types **8**
3.1 Martin Löf's J-Rule 9
3.2 J-Rule as function 9

4 The relation between the J-rule and the j-rule **10**
4.1 Transport 10
4.2 The M-judgement 12
4.3 Based versions 15
4.4 The j-Rule 15
4.5 Equivalence between the J-Rule and the j-Rule 16
4.6 Based Transport (transport) 17
4.7 Based LeastRefl (leastrefl) 18
4.8 The m-judgement 19

5 Applications of the J-rule **21**

6 C++ and the J-Rule **29**

7 Identity systems **33**

8 Appendix **40**

Abstract

After giving an overview of Martin-Löf Type Theory (MLTT), we focus on identity types. Based on the work of Coquand [1] and following [4], we present the relation between the fundamental judgements of MLTT with the J-rule, which is the main proof-tool of MLTT. Then we introduce a C++ program which is capable of using the J-rule in order to show the inhabitance of certain types. Finally, we present the fundamental theorem on identity systhems. Except from our construction of a proof checker in C++, this Thesis is based on the book-HoTT [5].

1 Introduction

Martin Löf's J-rule is the elimination rule for identity types in type theory. Before we treat this elimination rule closer, we want to give a short overview of type theory. In comparison to set theory, we have types instead of sets and terms instead of elements. By the Curry-Howard interpretation every type is understood as a proposition and every term is understood as a proof of the proposition represented by the type which is inhabited by that term. Type, space and proposition are synonyms in this theory. Term and proof are here synonyms as well. Every term belongs to some universe U. U is the type of a higher universe. As in [5, chapter 1.3] we give levels to the universes and require that all points of a universe and the universe it self are terms of the universe of the next level.
$U_i : U_{i+1}$ and if $a : U_i$ than $a : U_{i+1}$.

There are rules how to form a new type,
how to construct points of that type,
how to use these points (elimination rule),
how the construction and the use work together and
there can be something that expresses uniqueness.

We write $a : A$ to refer to a term a of type A. As we are not going to introduce a homotopical point of view, for us a function $f : A \to B$ is nothing but a proof that A implies B. we call terms of identity types paths, as it is done in the naive homotopical introduction of [5].

For this thesis the most important type is the identity type. It treats equality. For some $a, b : A$ the type $a =_A b$ is an identity type. The path $p : a =_A b$ is a proof that a and b are equal.
The J-rule is also called path induction.

If two terms are equal by definition we write $a \equiv b : A$ and we call a and b judgementally equal. If $a =_A b$ is inhabited we call a and b propositionally equal.

In the next chapter we present the basic types of MLTT.

2 Function types

Functions are a primitive concept of type theory. I explain the function type by prescribing how functions can be constructed and what we can do with them. The type $A \to B$ is a space of functions f. f can be defined or constructed using $\lambda - abstraction$.
Via Definition:
$f(x) :\equiv \phi$ where $x : A$ and for all $x : A$ $\phi : B$ is an expression of type B which can depend on x.
Via $\lambda - abstraction$:
Using an expression $\phi : B$ which can depend on $x : A$ we have the judgement $(\lambda(x : A).\phi) : A \to B$.
The function can be applied to an argument $a : A$ with the following judgemental equality:

$(\lambda x.\phi)(a) \equiv \phi^*$
where ϕ^* denotes the expression ϕ in which x was replaced by a.

2.1 Dependent function types ($\Pi - types$)

Given a type A and a family $P : A \to U$ one can construct the type

$$\prod_{x:A} P(x)$$

which is the type of dependant functions. We can construct dependent functions by definition or using $\lambda - abstraction$.
Via Definition:

$$f(x) :\equiv \phi$$

where $x : A$ and for all $x : A$ it holds that $\phi : P(x)$ is an expression which can depend on $x : A$.
Via $\lambda - abstraction$:

$$\lambda x.\phi : \prod_{x:A} P(x)$$

using an expression $\phi : P(x)$ which can depend on $x : A$.
Like an ordinary one the dependent function can be applied to an argument $a : A$ with the following judgemental equality:

$$f(a) \equiv (\lambda x.\phi)(a) \equiv \phi^*$$

where ϕ^* denotes the expression ϕ in which x was replaced by a.
If we choose B to be the family of constant functions, the dependent function becomes an ordinary function.

2.2 Product type ($A \times B$)

We call the type $A \times B$ the cartesian product of the types A and B. With $a : A$ and $b : B$ we can construct $(a, b) : A \times B$. For a family $C : A \times B \to U$ we can define a dependent function

$$f : \prod_{x:A\times B} C(x)$$

using a function

$$g : \prod_{x:A} \prod_{y:B} C((x, y))$$

by

$$f((x, y)) :\equiv g(x)(y).$$

in this way it is possible to construct a function

$$\text{uniqueness}_{A\times B} : \prod_{x:A\times B} ((\text{pr}_1(x), \text{pr}_2(x)) =_{A\times B} x)$$

with $\text{pr}_1((a, b)) :\equiv a$ and $\text{pr}_2((a, b)) :\equiv b$ by defining

$$\text{uniqueness}_{A\times B}((a, b)) :\equiv \text{refl}_{(a,b)}$$

which is a proof that all elements of the product type are pairs. Refl_x will be introduced later on.

2.3 Dependent pair types ($\sum -types$)

We want to allow the type of the second component of a pair to vary depending on the first component. Given a type $A : U$ and a family $P : A \to U$, we use the notation

$$\sum_{x:A} P(x) : U$$

for the dependent pair type.
With $a : A$ and $b : B(a)$ we get the pair

$$(a, b) : \sum_{x:A} P(x).$$

A function

$$f : (\sum_{x:A} P(x)) \to C$$

can be defined using a function

$$g : \prod_{x:A} P(x) \to C$$

by

$$f((a, b)) :\equiv g(a)(b)$$

therefore we have a family

$$C : (\sum_{x:A} P(x)) \to U.$$

We can construct a dependent function

$$f : \prod_{p:\sum_{x:A} P(x)} C(p)$$

using a function

$$g : \prod_{a:A} \prod_{b:P(a)} C((a, b))$$

by

$$f((a, b)) :\equiv g(a)(b).$$

This gives us the induction principle for $\sum -types$

$$\text{ind}_{\sum_{x:A} P(x)} : \prod_{(C:\sum_{x:A} P(x) \to U)} (\prod_{a:A} \prod_{b:P(a)} C((a, b))) \to \prod_{p:\sum_{x:A} P(x)} C(p).$$

2.4 Coproduct type (A + B)

With some types $A, B : U$ we use the notation $A+B : U$ for the coproduct type. For $a : A$ and $b : B$ we have the elements $\text{inl}(a), \text{inr}(b) : A + B$. To construct a function

$$f : A + B \to C$$

we use functions

$$g_0 : A \to C$$

$$g_1 : B \to C$$

and define

$$f(\mathrm{inl}(a)) :\equiv g_0(a),$$
$$f(\mathrm{inr}(b)) :\equiv g_1(b).$$

Now we have the family $P : (A + B) \to U$
For

$$g_0 : \prod_{a:A} C(\mathrm{inl}(a)),$$
$$g_1 : \prod_{b:B} C(\mathrm{inr}(b))$$

we define

$$f(\mathrm{inl}(a)) :\equiv g_0(a),$$
$$f(\mathrm{inr}(b)) :\equiv g_1(b).$$

This gives us the induction principle for (A + B)

$$\mathrm{ind}_{(A+B)} : \prod_{c:(A+B)\to U} (\prod_{a:A} C(\mathrm{inl}(a))) \to (\prod_{b:B} C(\mathrm{inr}(b)) \to \prod_{x:(A+B)} c(x).$$

3 Identity types

The proposition, that two points $a, b : A$ of a type A are equal, corresponds to some type. This type depend on a and b. Therefore the identity Types are a family which depend two times on A.
Write this family as

$$\mathrm{Id}_A : A \to A \to U$$

which is not the identity function.
The Type $\mathrm{Id}_A(a, b)$ that corresponds to the proposition that $a : A$ and $b : A$ are equal, will also be denoted by $a =_A b$.
A Point of $a =_A b$ is called a path.
The formation rule states that for a type $A : U$ and two elements $a, b : A$ we can form the type

$$a =_A b : U.$$

The basic way to do so is to know that a and b are equal.

$$(x \neq_A y) :\equiv \neg(x =_A y)$$

The introduction rule

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

is called reflexivity and states that every element of A is equal to itself. One could say, that the identity Types are generated by elements of the form $\mathrm{refl}_x : x = x$.

In order to characterize the identity Type we introduce the elimination rule (i.e. the induction principle) of it, which considers maps from $x =_A y$ and falsities over it. The elimination rule for identity Types is called J-Rule.

3.1 Martin Löf's J-Rule

Given a family

$$C : \prod_{x,y:A} (x =_A y) \to U$$

and a function

$$c : \prod_{x:A} C(x, x, \mathrm{refl}_x),$$

there is a function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

such that

$$f(x, x, \mathrm{refl}_x) :\equiv c(x).$$

The J-rule allows us to define a function $f : C$ with a property for all elements $x, y : A$ and paths $p : x =_A y$ by only considering cases where the elements are x, x and the path is $\mathrm{refl}_x : x =_A x$.

[2] tries to justify the J-rule with the uniqueness principle for identity types and the substitution of equals. [2] in which the term token is used for points, states in chapter 6.2 that

> ...the token constructors give us every token of the type up to identity. ...every token is equal to the output of one of the constructors,...

which leads to the uniqueness principle (M). The substitution of equals (Transport) shall come from pre-mathematical understanding of identity.

3.2 J-Rule as function

Definition 1 *We define*

$$\mathrm{ind}_{=_A} : \prod_{(C:\prod_{x,y:A}(x=_A y)\to U)} (\prod_{x:A} C(x, x, \mathrm{refl}_x) \to \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

with the computation rule

$$\mathrm{ind}_{=_A} (C, c, x, x, \mathrm{refl}_x) :\equiv c(x)$$

As the induction principal for identity types is called Martin Löf's J-rule, this function is called "J.".
As a consequence of the elimination rule equals can be substituted by equals which is expressed by Indiscernibility of identicals (Transport):
For every family

$$C : A \to U$$

there is a function

$$f : \Pi_{x,y:A} \Pi_{p:x=_A y} C(x) \to C(y)$$

such that

$$f(x, x, \mathrm{refl}_x) :\equiv \mathrm{id}_{C(x)}.$$

4 The relation between the J-rule and the j-rule

Definition 2 *We define*

$$\text{LeastRefl} : \prod_{R:A\to A\to U} \prod_{r:\prod_{x:A} R(x,x)} \prod_{x,y:A} \prod_{p:x=_A y} R(x,y)$$

with the compuattion rule

$$\text{LeastRefl}(R,r,x,x,\text{refl}_x) \equiv r(x); x : A$$

The LeastRefl-judgement expresses that $=_A$ is the least reflexive relation on A [4].

Proposition 1 *The J-Rule and its computation rule imply the judgement and the computaion rule of* LeastRefl.

Proof:

$$C : \prod_{x,y:A} (x =_A y) \to U$$

define

$$C(x,y,p) :\equiv \prod_{R:A\to A\to U} \prod_{r:\prod_{x:A} R(x,x)} R(x,y)$$

and

$$c(R,r,x) :\equiv r(x).$$

Than

$$c : \prod_{x:A} \prod_{R:A\to A\to U} \prod_{r:\prod_{x:A} R(x,x)} R(x,x).$$

Path induction implies the existence of a function

$$\text{LeastRefl} : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p)$$

such that

$$\text{LeastRefl}(R,r,x,x,\text{refl}_x) :\equiv c(R,r,x)$$

Replace $C(x,y,p)$ and $c(R,r,x)$ by its definitions

$$\text{LeastRefl} : \prod_{x,y:A} \prod_{p:x=_A y} \prod_{R:A\to A\to U} \prod_{r:\prod_{x:A} R(x,x)} R(x,y)$$

$$\text{LeastRefl}(R,r,x,x,\text{refl}_x) \equiv r(x)$$

4.1 Transport

Definition 3 *We define*

$$\text{Transport} : \prod_{P:A\to U} \prod_{x,y:A} \prod_{p:x=_A y} (P(x) \to P(y))$$

with the computation rule

$$\text{Transport}(P,x,x,\text{refl}_x) \equiv \text{id}_{P(x)}$$

where $x : A$.

Let P be a Property of elements of A and let x and y be equal. Than Transport implies that $P(x)$ holds if and only if $P(y)$ holds.
The Transport-judgement expresses the indiscernibility of identicals [4].

Proposition 2 *The J-Rule and its computation rule imply* Transport *and its computation rule*

Proof:

$$C : \prod_{x,y:A} (x =_A y) \to U$$

define

$$C(x, y, p) :\equiv \prod_{P:A\to U} (P(x) \to P(y))$$

and

$$c(P, x) :\equiv \mathrm{id}_{P(x)}.$$

Than

$$c : \prod_{x:A} \prod_{P:A\to U} (P(x) \to P(x)).$$

Path induction implies the existence of a function

$$\mathrm{Transport} : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

such that

$$\mathrm{Transport}(P, x, x, \mathrm{refl}_x) :\equiv c(P, x)$$

Replace $C(x, y, p)$ and $c(P, x)$ by its definitions

$$\mathrm{Transport} : \prod_{x,y:A} \prod_{p:x=_A y} \prod_{P:A\to U} (P(x) \to P(y))$$

$$\mathrm{Transport}(P, x, x, \mathrm{refl}_x) \equiv \mathrm{id}_{P(x)}$$

We can even show that LeastRefl and Transport are equivalent.

Proposition 3 LeastRefl *and its computation rule imply* Transport *and its computation rule.*

Proof: For all

$$P : A \to U$$

we have that

$$R(x, y) :\equiv P(x) \to P(y) : A \to A \to U.$$

Further

$$r(x) :\equiv \mathrm{id}_{P(x)} : P(x) \to P(x)$$

$$r : \prod_{x:A} R(x, x)$$

LeastRefl implies a function

$$f : \prod_{P:A\to U} \prod_{\mathrm{id}_{P(\bullet)}:\prod_{x:A} P(x)\to P(x)} \prod_{x,y:A} \prod_{p:x=_A y} (P(x) \to P(y))$$

which is the same as

$$f : \prod_{P:A\to U} \prod_{x,y:A} \prod_{p:x=_A y} (P(x) \to P(y))$$

such that

$$f(P, x, x, \mathrm{refl}_x) \equiv r(x) :\equiv \mathrm{id}_{P(x)}$$

As this f is equal to Transport, LeastRefl implies Transport.

Proposition 4 Transport *and its computation rule imply* LeastRefl *and its computation rule.*

Proof: For all $R : A \to A \to U$ and $r : \prod_{x:A} R(x,x)$ we define a function $P : A \to U$ by

$$P(x) :\equiv R(x,x).$$

Therefore $r(x) : P(x)$.

$$\mathrm{Transport}(P, x, y, p) : (P(x) \to P(y))$$

Now we define a function f by

$$f :\equiv \lambda(R, r, x, y, p).\mathrm{Transport}(P, x, y, p)(r(x))$$

$$\begin{aligned} f(R, r, x, x, \mathrm{refl}_x) &\equiv \mathrm{Transport}(P, x, x, \mathrm{refl}_x)(r(x)) \\ &\equiv \mathrm{id}_{P(x)}(r(x)) \\ &\equiv r(x) \end{aligned}$$

As this f is equal to LeastRefl, Transport implies LeastRefl.

4.2 The M-judgement

We already know that the $J - Rule$ implies LeastRefl and Transport. But the converse is not true. Therefore we want to find a sufficient condition M which in connection with LeastRefl or Transport implies the $J - Rule$. This M shall be implied by the $J - Rule$ as well. If we have that M, the following holds true:

$$\mathrm{Transport} \wedge \mathrm{M} \Leftrightarrow J - Rule$$

$$\mathrm{LeastRefl} \wedge \mathrm{M} \Leftrightarrow J - Rule$$

The following M is the M we want.

Definition 4 *With*

$$E_a :\equiv \sum_{x:A} (a =_A x)$$

we define the term

$$\mathrm{M} : \prod_{a,x:A} \prod_{p:a=_A x} (a, \mathrm{refl}_x) =_{E_a} (x, p),$$

with the computation rule

$$\mathrm{M}(a, a, \mathrm{refl}_a) \equiv \mathrm{refl}_{a,\mathrm{refl}_a}$$

Generally, the uniqueness principal for an inductively defined type A expresses the fact that every point of the identity type is equal to an output of a constructor of that type. According to [5] and [3] M is the uniqueness principle for identity types.

Theorem 1 *The J-Rule and its computation rule imply the* M*-judgement and its computation rule.*

Proof:

$$C : \prod_{a,x:A} (a =_A x) \to U$$

Define

$$C(a, x, p) :\equiv (a, \mathrm{refl}_a) =_{E_a} (x, p),$$
$$C(a, a, \mathrm{refl}_a) \equiv (a, \mathrm{refl}_a) =_{E_a} (a, \mathrm{refl}_a)$$

and

$$c(a) :\equiv \mathrm{refl}_{(a,\mathrm{refl}_a)}$$

. Than

$$c : \prod_{a:A} C(a, a, \mathrm{refl}_a) \equiv \prod_{a:A} (a, \mathrm{refl}_a) =_{E_a} (a, \mathrm{refl}_a).$$

Path induction implies a function

$$\mathrm{M} : \prod_{a,x:A} \prod_{p:a=_A x} (a, \mathrm{refl}_a) =_{E_a} (x, p)$$

such that

$$\mathrm{M}(a, a, \mathrm{refl}_a) :\equiv c(a) \equiv \mathrm{refl}_{(a,\mathrm{refl}_a)}.$$

Theorem 2 *The* LeastRefl- *and the* M*-judgement and their computation rules imply the J-Rule and its computation rule.*

Proof: We require

$$C : \prod_{a,x:A} (a =_A x) \to U$$

and a function

$$c : \prod_{a:A} C(a, a, \mathrm{refl}_a).$$

Define the function $R : E_a \to E_a \to U$ by

$$R((a, p), (x, p)) :\equiv C(a, x, p).$$

Further define r by

$$r(a, \mathrm{refl}_a) :\equiv c(a).$$

Now we have a point $r : \prod_{(x,\mathrm{refl}_x):E_a} R((x, \mathrm{refl}_x), (x, \mathrm{refl}_x))$. We have

$$\mathrm{M}(a, x, p) : (a, \mathrm{refl}_a) =_{E_a} (x, p)$$

and

$$\mathrm{LeastRefl}(R, r, (a, \mathrm{refl}_a), (x, p), \mathrm{M}(a, x, p)) : R((a, \mathrm{refl}_a), (x, p))$$

in which we can use the definition of R

$$\mathrm{LeastRefl}(R, r, (a, \mathrm{refl}_a), (x, p), \mathrm{M}(a, x, p)) : C(a, x, p)$$

We know

$$\mathrm{LeastRefl}(R, r, y, y, \mathrm{refl}_y) \equiv r(y),$$

$$\mathrm{M}(y, y, \mathrm{refl}_y) \equiv \mathrm{refl}_{(y,\mathrm{refl}_y)}.$$

Now we can define

$$F :\equiv \lambda(a, x : A, p : a =_A x).\mathrm{LeastRefl}(R, r, (a, \mathrm{refl}_a), (x, p), \mathrm{M}(a, x, p))$$

$$\begin{aligned} F(a, a, \mathrm{refl}_a) &\equiv \mathrm{LeastRefl}(R, r, (a, \mathrm{refl}_a), (a, \mathrm{refl}_a), \mathrm{M}(a, a, p)) \\ &\equiv \mathrm{LeastRefl}(R, r, (a, \mathrm{refl}_a), (a, \mathrm{refl}_a), \mathrm{refl}_{(a,\mathrm{refl}_a)}) \\ &\equiv \mathrm{LeastRefl}(R, r, (a, \mathrm{refl}_a), (a, \mathrm{refl}_a), \mathrm{refl}_{(a,\mathrm{refl}_a)}) \\ &\equiv r((a, \mathrm{refl}_a)) \equiv c(a). \end{aligned}$$

As $F(a, x, p) : C(a, x, p)$ we have

$$F : \prod_{a,x:A} \prod_{p:a=_A x} C(a, x, p)$$

such that

$$F(a, a, \mathrm{refl}_a) \equiv c(a).$$

Corollary 1 *The* Transport- *and the* M*-judgement and their computation rules imply the J-Rule and its computation rule.*

Proof: It follows immediately form 2 together with 4 but we can show the implication directly.
We require

$$C : \prod_{a,x:A} (a =_A x) \to U$$

and a function

$$c : \prod_{a:A} C(a, a, \mathrm{refl}_a).$$

For all $a : A$ and $p : a =_A x$ the recursion principle allows us to define a dependent function

$$P_a : E_a \to U$$

by

$$P_a((x, p)) :\equiv C(a, x, p).$$

We have

$$\mathrm{M} : (a, x, p) : (a, \mathrm{refl}_a) =_{E_a} (x, p)$$

and

$$\mathrm{Transport}(P_a, (a, \mathrm{refl}_a), (x, p), \mathrm{M}(a, x, p)) : (P_a((a, \mathrm{refl}_a)) \to P_a((x, p)))$$

in which we can use the definition of P_a

$$\mathrm{Transport}(P_a, (a, \mathrm{refl}_a), (x, p), \mathrm{M}(a, x, p)) : C(a, a, \mathrm{refl}_a) \to C(a, x, p).$$

We know

$$\text{Transport}(P, y, y, \text{refl}_y) \equiv \text{id}_{P(y)},$$

$$\text{M}(y, y, \text{refl}_y) \equiv \text{refl}_{(y,\text{refl}_y)}.$$

Now we can define

$$F :\equiv \lambda(a, x : A, p : a =_A x).\text{Transport}(P_a, (a, \text{refl}_a), (x, p), \text{M}(a, x, p))(c(a))$$

$$F(a, a, \text{refl}_a) \equiv \text{Transport}(P_a, (a, \text{refl}_a), (a, \text{refl}_a), \text{M}(a, a, p))(c(a))$$

$$\begin{aligned} F(a, a, \text{refl}_a) &\equiv \text{Transport}(P_a, (a, \text{refl}_a), (a, \text{refl}_a), \text{refl}_{(a,\text{refl}_a)})(c(a)) \\ &\equiv \text{id}_{P_a(a,\text{refl}_a)}(c(a)) \\ &\equiv (c(a)). \end{aligned}$$

As $F(a, x, p) : C(a, x, p)$ we have

$$F : \prod_{a,x:A} \prod_{p:a=_A x} C(a, x, p)$$

such that

$$F(a, a, \text{refl}_a) \equiv c(a).$$

4.3 Based versions

There are Based versions [3] of $J - Rule$, Transport, LeastRefl and M which are all equivalent to the normal forms. The based versions can be helpful as they use less unknown variables.

4.4 The j-Rule

The j-Rule is a synonym for based path induction.
Given a proof $P : a =_A b$, the J-rule replaces a and b with the unknown x and y. In some cases it can be more simple to replace a or b and do the remainder of the proof for a specific element instead for unknown x and y.
The j-rule says that the types $a =_A x$ are generated by $\text{refl}_a : a = a$.
Fix a:A
Given a family

$$C : \prod_{x:A} (a =_A x) \to U)$$

and an element

$$c : C(a, \text{refl}_a),$$

there is a function

$$f : \prod_{x:A} \prod_{p:a=x} C(x, p)$$

such that

$$f(a, \text{refl}_a) :\equiv c.$$

The j-rule allows us to define a function $f : C$ with property for all elements x and paths $p : a =_A x$ by only considering cases where the element is a and the path is $\text{refl}_a : a =_A a$.

Definition 5 *The j-Rule as function*

$$\text{ind}^* =_A : \prod_{a:A} \prod_{(C:\prod_{x:A}(a=_A x)\to U)} C(a, refl_a) \to \prod_{x:A} \prod_{p:a=_A x} C(x,p)$$

with the computation rule

$$\text{ind}^* =_A (a, C, c, a, \text{refl}_a) :\equiv c$$

4.5 Equivalence between the J-Rule and the j-Rule

Both induction principles for the identity Type are equivalent.

Proposition 5 *The j-Rule and its computation rule imply the J-Rule and its computation rule.*

Proof: Given a family

$$C : \prod_{x,y:A} (x =_A y) \to U)$$

and a function

$$c : \prod_{x:A} C(x, x, refl_x),$$

for some $x : A$ we can construct

$$C^* : \prod_{y:A} (x =_A y) \to U)$$

$$c^* : C(x, \text{refl}_x),$$

such that

$$C^* :\equiv C(x),$$

$$c^* :\equiv c(x).$$

Based path induction implies the existence of a function

$$g : \Pi_{y:A}\Pi_{p:x=_A y} C^*(y,p)$$

such that

$$g(x, \text{refl}_x) :\equiv c^*.$$

By discharging the assumption $x : A$ we derive a function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p)$$

with

$$f(x) :\equiv g.$$

It follows:

$$f(x, x, \text{refl}_x) \equiv g(x, \text{refl}_x) :\equiv c^* :\equiv c(x)$$

[5] gives the following homotopical proof. Later in corollary 2 we proof the same result in a different way.

Theorem 3 *The J-Rule implies the j-Rule and their computation rule.*

Proof: Let us assume all possible kinds of given Families

$$C : \prod_{z:A}(x =_A z) \to U$$

with elements

$$c : C(x, \text{refl}_x).$$

Define

$$D : \prod_{x,y:A}(x =_A y) \to U$$

with

$$D(x, y, p) :\equiv \Pi_{C:\Pi_{z:A}(x=_A z)\to U} C(x, \text{refl}_x) \to C(y, p).$$

We can construct

$$d : \prod_{x:A} D(x, x, \text{refl}_x)$$

using $\lambda - abstraction$

$$d :\equiv \lambda x.\lambda C.\lambda(c : C(x, \text{refl}_x)).c.$$

Path induction implies the existence of a function

$$f : \prod_{x,y:A} \Pi_{p:x=_A y} D(x, y, p)$$

with

$$f(x, x, \text{refl}_x) :\equiv d(x).$$

Inserting the definition of D into the Type of f leads to

$$f : \prod_{x,y:A}\prod_{p:x=_A y}\prod_{C:\prod_{z:A}(x=_A z)\to U} C(x, \text{refl}_x) \to C(y, p)$$

Fix an element $x : A$. For a path $p : a =_A x$

$$f(a, x, p, C, c) : C(x, p).$$

4.6 Based Transport (transport)

Definition 6 *We define*

$$\text{transport} : \prod_{a:A}\prod_{P:A\to U}\prod_{x:A}\prod_{p:a=_A x}(P(a) \to P(x)),$$

with their computation rule

$$\text{transport}(a, P, a, \text{refl}_a) \equiv \text{id}_{P(a)}.$$

Proposition 6 transport *and its computation rule is implied by the $j - Rule$ and its computation rule.*

$$C : \prod_{x:A} (a =_A x) \to U$$

Define for all $a : A$ and all $P : A \to U$ the functions

$$C_{(a,P)}(x, p) :\equiv (P(a) \to P(x))$$

where $C_{(a,P)} \equiv C(a, P)$. We have

$$\mathrm{id}_{P(a)} : C_{(a,P)}(a, \mathrm{refl}_a).$$

The $j - Rule$ gives a function

$$f : \prod_{x:A} \prod_{p:a=_A x} C_{(a,P)}(x, p)$$

such that

$$f(a, refl_a) \equiv \mathrm{id}_{P(a)}.$$

Than

$$\mathrm{transport}(a, P) :\equiv f$$

is what we were looking for.

Proposition 7 *Transport and transport are equivalent.*

If we fix an arbitrary $a : A$ at Transport, we get a function $\mathrm{Transport}_a$.

$$\mathrm{transport} :\equiv \lambda a.\mathrm{Transport}_a$$

If we fix arbitrary a and P in transport and Transport, we get equal functions $\mathrm{transport}_{a,P}$ and $\mathrm{Transport}_{P,a}$. Therefore

$$\mathrm{Transport} :\equiv \lambda P.\lambda a.\mathrm{transport}_{a,P}.$$

4.7 Based LeastRefl (leastrefl)

Definition 7 *We define*

$$\mathrm{leastrefl} : \prod_{a:A} \prod_{R_a:A\to U} \prod_{r_a:R_a(a)} \prod_{x:A} \prod_{p:a=_A x} R_a(x),$$

with its computation rule

$$\mathrm{leastrefl}(a, R_a, r_a, a, refl_a) \equiv r_a.$$

As expected, leastrefl is directly derived from the $j - Rule$:

$$C : \prod_{x:A} (a =_A x) \to U$$

Define for all $a : A$ and all $R_a : A \to U$ and all $r_a : R_a(a)$ the functions

$$C_{(a,R_a,r_a)}(x, p) :\equiv R_a(x)$$

where $C_{(a,R_a,r_a)} \equiv C(a, R_a, r_a)$. We have

$$r_a : C_{(a,R_a,r_a)}(x, p).$$

The $j - Rule$ gives a function

$$f : \prod_{x:A} \prod_{p:a=_A x} C_{(a,R_a,r_a)}(x,p)$$

such that

$$f(a, \mathrm{refl}_a) \equiv r_a.$$

Than

$$\mathrm{leastrefl}(a, R_a, r_a) :\equiv f$$

is what we were looking for.

LeastRefl and leastrefl are equivalent. We get one from the other by defining either

$$\prod_{a:A}(R_a(x) :\equiv R(a,x)),$$
$$r_a :\equiv r(a)$$

or

$$\prod_{a:A}(R(a,x) :\equiv R_a(x)),$$
$$r(a) :\equiv r_a.$$

4.8 The m-judgement

Definition 8 *We define*

$$\mathrm{m} : \prod_{a:A} \prod_{u:E_a} (a, \mathrm{refl}_a) =_{E_a} u,$$

with its computation rule

$$\mathrm{m}(a, (a, \mathrm{refl}_a)) \equiv \mathrm{refl}_{(a,\mathrm{refl}_a)}.$$

As expected, m is directly derived from $j - Rule$:

$$C : \prod_{x:A}(a =_A x) \to U$$

Define the function

$$C(x,p) :\equiv ((a, \mathrm{refl}_a) =_{E_a} (x,p)).$$

We have

$$\mathrm{refl}_{(a,\mathrm{refl}_a)} : C(a, \mathrm{refl}_a).$$

The $j - Rule$ gives a function

$$f : \prod_{x:A} \prod_{p:a=_A x} C(x,p)$$

such that

$$f(a, \mathrm{refl}_a) \equiv \mathrm{refl}_{(a,\mathrm{refl}_a)}.$$

Than

$$m :\equiv f$$

is what we were looking for.

M and m are equivalent.

Proposition 8 *The* m*-Judgement and its computation rule imply the* M*-judgement and its computation rule.*

Proof: We define $\mathrm{M}(a, x, p) \equiv \mathrm{m}_a((x, p))$ and get the implication.

Proposition 9 *The* M*-Judgement and its computation rule imply the* m*-Judgement and its computation rule.*

Proof: We define the family $Q : E_a \to U$ for all $u : E_a$ by

$$Q(u) \equiv (a, \mathrm{refl}_a) =_{E_a} u.$$

Now

$$M(a) : \prod_{x:A} \prod_{p:a=_A x} (a, \mathrm{refl}_a) =_{E_a} (x, p))$$

and as $Q((x,p) \equiv (a, \mathrm{refl}_a) =_{E_a} (x,p)$ it exists a family $F : \prod_{u:E_a} Q(u)$ such that

$$F((a, \mathrm{refl}_a)) \equiv \mathrm{M}(a)(a, \mathrm{refl}_a) \equiv \mathrm{M}(a, a, \mathrm{refl}_a) \equiv \mathrm{refl}_{a,\mathrm{refl}_a}).$$

Theorem 4 m_a *and* transport *with their computation rules imply the j-rule and its computation rule.*

Proof: We have

$$C : \prod_{x:a} \prod_{p:a=_A c} U$$

with $c : C(a, \mathrm{refl}_a)$. We get

$$F : \prod_{x:A} \prod_{p:a=_A x} C(x,p)$$

with $F(a, \mathrm{refl}_a) \equiv c$. Because of the recursion principal for $\sum$-types we can define $P : E_a \to U$ by $P((x, p)) \equiv C(x, p)$, for every $x : A$ and $p : a =_A x$. We have

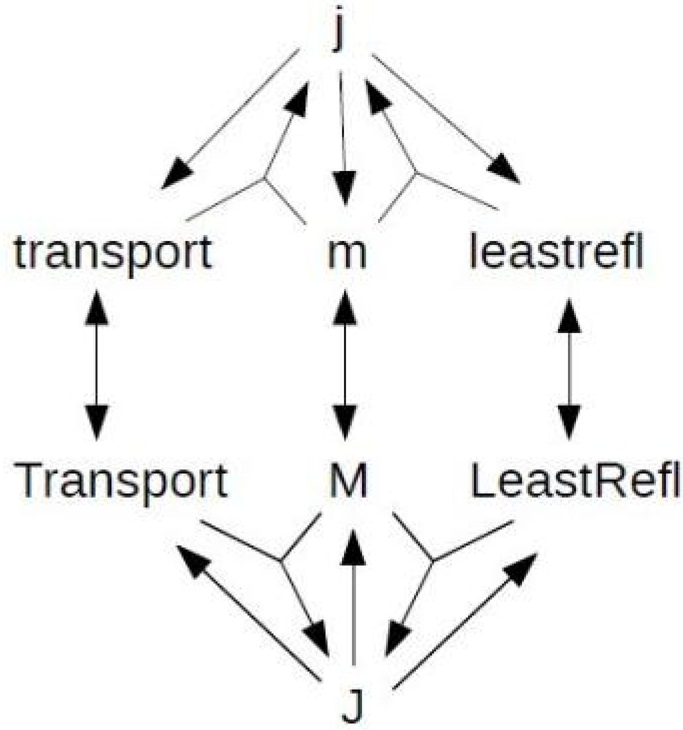
$$\mathrm{m}_a((x,p)) : (a, \mathrm{refl} : a) =_{E_a} (x,p).$$

It follows from transport that $[\mathrm{m}_a((x,p))]_*^P : P((a, \mathrm{refl}_a)) \to P((x,p))$.
By replacing P with C we get $[\mathrm{m}_a((x,p))]_*^P : C((a, \mathrm{refl}_a)) \to C((x,p)$.
Define $F :\equiv \lambda(x : A, p : a =_A x).[\mathrm{m}_a((x,p))]_*^P(c)$.
Now we get

$$\begin{aligned}
F(a, \mathrm{refl}_a) \equiv & \quad [\mathrm{m}_a((x,p))]_*^P(c) \\
\equiv & \quad [\mathrm{refl}_{(a,\mathrm{refl}_a)}]_*^P(c) \\
\equiv & \quad \mathrm{id}_P(a, \mathrm{refl}_a)(c) \\
\equiv & \quad \mathrm{id}_C(a, \mathrm{refl}_a)(c) \\
\equiv & \quad c
\end{aligned}$$

Corollary 2 *The J-rule and its computation rule imply the j-rule and its computation rule.*

Proof: The theorem 1 and the proposition 2 give us M, Transport and their computation rules. By proposition 9 M and its computation rule imply m and its computation rule. By proposition 7 Transport and its computation rule imply transport and its computation rule. By theorem 4 transport and m imply the j-rule.



5 Applications of the J-rule

As mentioned before we identify a proposition with a Type and a proven statement with an inhabited Type. We will proof the following propositions by translating them into Types and than use path induction to construct elements of these Types.

Lemma 1 *For every type A and every $x, y : A$ there is a function*

$$(x =_A y) \to (y =_A x)$$

denoted $p \to p^{-1}$, such that for every $x : A$

$$\mathrm{refl}_x^{-1} \equiv \mathrm{refl}_x$$

.

Proof: We have $A : U$ and $p : x =_A y$ and want to show that

$$\prod_{A:U} \prod_{x,y:A} (x =_A y) \to (y = x)$$

is inhabited. Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x, y, p) :\equiv (y =_A x)$$

Than $c :\equiv \lambda x.\mathrm{refl}_x$ is an element of $\prod_{x:A} C(x, x, \mathrm{refl}_x)$.
Path induction gives us a function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p),$$

such that $f(x, x, \mathrm{refl}_x) :\equiv c(x)$.
As $f(x, y, p) : (y =_A x)$ our type to poof is inhabited by

$$\lambda p.f(x, y, p).$$

$$\lambda p.f(x, x, \mathrm{refl}_x)(\mathrm{refl}_x) =_{x=_A x} \mathrm{refl}_x^{-1} \equiv \mathrm{refl}_x$$

as required. Call p^{-1} the inverse of p.

Lemma 2 *For every Type A and every $x, y, z : A$ there is a function*

$$(x =_A y) \to (y =_A z) \to (x =_A z)$$

denoted

$$p \to q \to p \bullet q,$$

such that

$$\mathrm{refl}_x \bullet \mathrm{refl}_x \equiv \mathrm{refl}_x$$

for any $x : A$.

Proof: We have $A : U$, $p : x =_A y$ and $q : y =_A z$ and want to show that

$$\prod_{A:U} \prod_{x,y,z:A} (x =_A y) \to (y =_A z) \to (x =_A z)$$

is inhabited. Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x, y, p) :\equiv \prod_{z:A} (y =_A z) \to (x =_A z)$$

$$C(x, x, \mathrm{refl}_x) \equiv \prod_{z:A} (x =_A z) \to (x =_A z).$$

We need an element of $\prod_{x:A} C(x, x, \mathrm{refl}_x)$.
Define

$$D : \prod_{x,z:A} (x =_A z) \to U$$

with

$$D(x, z, q) :\equiv (x =_A z).$$

Than d defined with $d(x) :\equiv \mathrm{refl}_x$ is an element of

$$\prod_{x:A} D(x, x, \mathrm{refl}_x).$$

Path induction gives us a function

$$c : \prod_{x,z:A} \prod_{q:x=_A z} D(x,z,q)$$

such that $c(x,x,\text{refl}_x) :\equiv d(x)$.

$$c : \prod_{x:A} C(x,x,\text{refl}_x)$$

holds true.
Path induction gives us a function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p)$$

such that $f(x,x,\text{refl}_x) :\equiv c(x)$.

$$f : \prod_{x,y,z:A} (x =_A y) \to (y =_A z) \to (x =_A z)$$

holds.

$$f(C,c,D,d,x,x,x,\text{refl}_x,\text{refl}_x) \equiv c(D,d,x,x,\text{refl}_x) \equiv d(x) \equiv \text{refl}_x$$

as required. Call $p \bullet q$ the composition of p and q.

Lemma 3 *Given $A : U$ and $x,y,z,w : A$ and $p : x = y$ and $q : y = z$ and $r : z = w$ the following hold.*

(i) $p = p \bullet \text{refl}_y$ *is inhabited.*

(ii) $p = \text{refl}_x \bullet p$ *is inhabited.*

(iii) $p^{-1} \bullet p = \text{refl}_y$ *is inhabited.*

(iv) $p \bullet p^{-1} = \text{refl}_x$ *is inhabited.*

(v) $(p^{-1})^{-1} = p$ *is inhabited.*

(vi) $p \bullet (q \bullet r) = (p \bullet q) \bullet r$ *is inhabited.*

Proof: (i) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x,y,p) :\equiv p = p \bullet \text{refl}_y,$$

$$C(x,x,\text{refl}_x) \equiv \text{refl}_x = \text{refl}_x \bullet \text{refl}_x$$

Since $\text{refl}_x \bullet \text{refl}_x \equiv \text{refl}_x$, $C(x,x,\text{refl}_x) \equiv \text{refl}_x = \text{refl}_x$.

$$c :\equiv \lambda x.\text{refl}_{\text{refl}_x} : \prod_{x:A} C(x,x,\text{refl}_x).$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p),$$

$$f(x, y, p) : p = p \bullet \mathrm{refl}_y$$

holds.

(ii) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x, y, p) :\equiv p = \mathrm{refl}_x \bullet p.$$

$$C(x, x, \mathrm{refl}_x) \equiv \mathrm{refl}_x = \mathrm{refl}_x \bullet \mathrm{refl}_x$$

Since $\mathrm{refl}_x \bullet \mathrm{refl}_x \equiv \mathrm{refl}_x$, $C(x, x, \mathrm{refl}_x) \equiv \mathrm{refl}_x = \mathrm{refl}_x$.

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_x} : \prod_{x:A} C(x, x, \mathrm{refl}_x).$$

Path induction gives us the required function

$$f : \Pi_{x,y:A} \Pi_{p:x=_A y} C(x, y, p)$$

$f(x, y, p) : p = \mathrm{refl}_x \bullet p$ holds.

(iii) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x, y, p) :\equiv p^{-1} \bullet p = \mathrm{refl}_y.$$

$$C(x, x, \mathrm{refl}_x) \equiv \mathrm{refl}_x^{-1} \bullet \mathrm{refl}_x = \mathrm{refl}_x$$

Since $\mathrm{refl}_x^{-1} \equiv \mathrm{refl}_x$, $\mathrm{refl}_x^{-1} \bullet \mathrm{refl}_x \equiv \mathrm{refl}_x \bullet \mathrm{refl}_x$. Further $\mathrm{refl}_x \bullet \mathrm{refl}_x \equiv \mathrm{refl}_x$. Therefore

$$C(x, x, \mathrm{refl}_x) \equiv \mathrm{refl}_x = \mathrm{refl}_x.$$

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_x} : \Pi_{x:A} C(x, x, \mathrm{refl}_x).$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

$$f(x, y, p) : p^{-1} \bullet p = \mathrm{refl}_y$$

holds.

(iv) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x,y,p) :\equiv p \bullet p^{-1} = \mathrm{refl}_x.$$

$$C(x,x,\mathrm{refl}_x) \equiv \mathrm{refl}_x \bullet \mathrm{refl}_x^{-1} = \mathrm{refl}_x$$

Since $\mathrm{refl}_x^{-1} \equiv \mathrm{refl}_x$, $\mathrm{refl}_x \bullet \mathrm{refl}_x^{-1} \equiv \mathrm{refl}_x \bullet \mathrm{refl}_x$. Further $\mathrm{refl}_x \bullet \mathrm{refl}_x \equiv \mathrm{refl}_x$. Therefore

$$C(x,x,\mathrm{refl}_x) \equiv \mathrm{refl}_x = \mathrm{refl}_x.$$

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_x} : \prod_{x:A} C(x,x,\mathrm{refl}_x).$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p),$$

$$f(x,y,p) : p \bullet p^{-1} = \mathrm{refl}_x$$

holds.

(v) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x,y,p) :\equiv (p^{-1})^{-1} = p,$$

$$C(x,x,\mathrm{refl}_x) \equiv (\mathrm{refl}_x^{-1})^{-1} = p.$$

Since $\mathrm{refl}_x^{-1} \equiv \mathrm{refl}_x$, $(\mathrm{refl}_x^{-1})^{-1} \equiv (\mathrm{refl}_x)^{-1} \equiv \mathrm{refl}_x$.
Therefore $C(x,x,\mathrm{refl}_x) \equiv \mathrm{refl}_x = \mathrm{refl}_x$.

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_x} : \prod_{x:A} C(x,x,\mathrm{refl}_x).$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p),$$

$$f(x,y,p) : (p^{-1})^{-1} = p$$

holds.

(vi) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x,y,p) :\equiv \prod_{z,w:A} \prod_{q:y=z} \prod_{r:z=w} p \bullet (q \bullet r) = (p \bullet q) \bullet r.$$

$$C(x,x,\mathrm{refl}_x) \equiv \prod_{z,w:A} \prod_{q:x=z} \prod_{r:z=w} \mathrm{refl}_x \bullet (q \bullet r) = (\mathrm{refl}_x \bullet q) \bullet r$$

We have to find an element of $C(x, x, \mathrm{refl}_x)$.
Define

$$D : \prod_{x,z:A} (x =_A z) \to U$$

with

$$D(x, z, q) :\equiv \prod_{w:A} \prod_{r:z=w} \mathrm{refl}_x \bullet (q \bullet r) = (\mathrm{refl}_x \bullet q) \bullet r.$$

$$D(x, x, \mathrm{refl}_x) \equiv \prod_{w:A} \Pi_{r:x=w} \mathrm{refl}_x \bullet (\mathrm{refl}_x \bullet r) = (\mathrm{refl}_x \bullet \mathrm{refl}_x) \bullet r$$

We have to find an element of $D(x, x, \mathrm{refl}_x)$.
Define

$$E : \prod_{x,w:A} (x =_A w) \to U$$

with

$$E(x, w, r) :\equiv \mathrm{refl}_x \bullet (\mathrm{refl}_x \bullet r) = (\mathrm{refl}_x \bullet \mathrm{refl}_x) \bullet r.$$

$$E(x, x, \mathrm{refl}_x) :\equiv \mathrm{refl}_x \bullet (\mathrm{refl}_x \bullet \mathrm{refl}_x) = (\mathrm{refl}_x \bullet \mathrm{refl}_x) \bullet \mathrm{refl}_x.$$

Since $\mathrm{refl}_x \bullet \mathrm{refl}_x \equiv \mathrm{refl}_x$,

$$E(x, x, \mathrm{refl}_x) \equiv \mathrm{refl}_x \bullet (\mathrm{refl}_x) = (\mathrm{refl}_x) \bullet \mathrm{refl}_x \equiv \mathrm{refl}_x = \mathrm{refl}_x,$$

$$e :\equiv \lambda.\mathrm{refl}_{\mathrm{refl}_x} : \prod_{x:A} E(x, x, \mathrm{refl}_x).$$

Path induction gives us the required function

$$d : \prod_{x,w:A} \prod_{r:x=_A w} E(x, w, r)$$

$$d(x, w, r) : \mathrm{refl}_x \bullet (\mathrm{refl}_x \bullet r) = (\mathrm{refl}_x \bullet \mathrm{refl}_x) \bullet r$$

$$d : \prod_{x:A} D(x, x, \mathrm{refl}_x)$$

hold.
Path induction gives us the required function

$$c : \prod_{x,z:A} \prod_{q:x=_A z} D(x, z, q)$$

$$c(x, z, q) : \prod_{w:A} \Pi_{r:z=w} \mathrm{refl}_x \bullet (q \bullet r) = (\mathrm{refl}_x \bullet q) \bullet r$$

$c : \prod_{x:A} C(x, x, \mathrm{refl}_x)$ hold. Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

$$f(x, y, p) : \prod_{z,w:A} \prod_{q:y=_A z} \prod_{r:z=_A w} p \bullet (q \bullet r) = (p \bullet q) \bullet r$$

$$f(x, y, z, w, p, q, r) : p \bullet (q \bullet r) = (p \bullet q) \bullet r$$

Lemma 4 *Let $f : A \to B$. For all $x, y : A$ there is a function*

$$\mathrm{ap}_f : (x =_A y) \to (f(x) =_B f(y))$$

such that, for all $x : A$ holds $\alpha_f(\mathrm{refl}_x) \equiv \mathrm{refl}_{f(x)}$ which means functions preserve paths.

Proof: Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$C(x, y, p) :\equiv (f(x) =_B f(y)).$$

$$C(x, x, \mathrm{refl}_x) \equiv f(x) =_B f(x)$$

$$c :\equiv \lambda x.\mathrm{refl}_{f(x)} : \prod_{x:A} C(x, x, \mathrm{refl}_x)$$

Path induction gives us the required function

$$\mathrm{ap}_f \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

with $\mathrm{ap}_f(x, x, \mathrm{refl}_x) \equiv c(x) \equiv \mathrm{refl}_{f(x)}$.

Note that $\mathrm{ap}_f(\mathrm{refl}_x) \equiv \mathrm{refl}_{f(x)}$.

Lemma 5 *Given $f : A \to B$ and $g : B \to C$ and $p : x =_A y$ and $q : y =_A z$ the following hold.*

(i) $\mathrm{ap}_f(p \bullet q) = \mathrm{ap}_f(p) \bullet \alpha_f(q)$ *is inhabited.*

(ii) $\mathrm{ap}_f(p^{-1}) = (\mathrm{ap}_f(p))^{-1}$ *is inhabited.*

(iii) $\mathrm{ap}_g(\mathrm{ap}_f(p)) = \mathrm{ap}_{g\bullet f}(p)$ *is inhabited.*

(iv) $\mathrm{ap}_{\mathrm{id}_A}(p) = p$ *is inhabited.*

Proof: (i) Define

$$C : \Pi_{x,y:A}(x =_A y) \to U$$

with

$$\begin{aligned} C(x, y, p) :\equiv && \mathrm{ap}_f(p \bullet q) = \mathrm{ap}_f(p) \bullet \mathrm{ap}_f(q) \\ C(x, x, \mathrm{refl}_x) \equiv && \mathrm{ap}_f(\mathrm{refl}_x \bullet \mathrm{refl}_x) = \mathrm{ap}_f(\mathrm{refl}_x) \bullet ap_f(\mathrm{refl}_x) \\ \equiv && \mathrm{ap}_f(\mathrm{refl}_x) = \mathrm{refl}_{f(x)} \bullet \mathrm{refl}_{f(x)} \\ \equiv && \mathrm{refl}_{f(x)} = \mathrm{refl}_{f(x)} \end{aligned}$$

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_x} : \prod_{x:A} C(x, x, \mathrm{refl}_x)$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

$$f(x,y,p) : \mathrm{ap}_f(p \bullet q) = \mathrm{ap}_f(p) \bullet \mathrm{ap}_f(q).$$

(ii) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$\begin{aligned}
C(x,y,p) :&\equiv & \mathrm{ap}_f(p^{-1}) = (\mathrm{ap}_f(p))^{-1} \\
C(x,x,\mathrm{refl}_x) &\equiv & ap_f(\mathrm{refl}_x^{-1}) = (\mathrm{ap}_f(\mathrm{refl}_x))^{-1} \\
&\equiv & \mathrm{ap}_f(\mathrm{refl}_x) = (\mathrm{refl}_{f(x)})^{-1} \\
&\equiv & \mathrm{refl}_{f(x)} = \mathrm{refl}_{f(x)}
\end{aligned}$$

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_x} : \prod_{x:A} C(x,x,\mathrm{refl}_x)$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p)$$

$$f(x,y,p) : \mathrm{ap}_f(p^{-1}) = (\mathrm{ap}_f(p))^{-1}$$

(iii) Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$\begin{aligned}
C(x,y,p) :&\equiv & \mathrm{ap}_g(\mathrm{ap}_f(p)) = \mathrm{ap}_{g\bullet f}(p) \\
C(x,x,\mathrm{refl}_x) &\equiv & \mathrm{ap}_g(\mathrm{ap}_f(\mathrm{refl}_x)) = \mathrm{ap}_{g\bullet f}(\mathrm{refl}_x) \\
&\equiv & \mathrm{ap}_g(\mathrm{refl}_{f(x)}) = \mathrm{refl}_{g\bullet f(x)} \\
&\equiv & \mathrm{refl}_{g(f(x)} = \mathrm{refl}_{g(f(x))}
\end{aligned}$$

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_{g(f(x))}} : \prod_{x:A} C(x,x,\mathrm{refl}_x)$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p)$$

$$f(x,y,p) : \mathrm{ap}_g(\mathrm{ap}_f(p)) = \mathrm{ap}_{g\bullet f}(p)$$

(iv) $\mathrm{id}_A(x) :\equiv x$
Define

$$C : \prod_{x,y:A} (x =_A y) \to U$$

with

$$\begin{aligned} C(x,y,p) :\equiv & \qquad \mathrm{ap}_{\mathrm{id}_A}(p) = p \\ C(x,x,\mathrm{refl}_x) \equiv & \qquad \mathrm{ap}_{\mathrm{id}_A}(\mathrm{refl}_x) = \mathrm{refl}_x \\ \equiv & \qquad \mathrm{refl}_{\mathrm{id}_A(x)} = \mathrm{refl}_x \\ \equiv & \qquad \mathrm{refl}_x = \mathrm{refl}_x \end{aligned} \tag{1}$$

$$c :\equiv \lambda x.\mathrm{refl}_{\mathrm{refl}_x} : \prod_{x:A} C(x,x,\mathrm{refl}_x)$$

Path induction gives us the required function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x,y,p)$$

$f(x,y,p) : \mathrm{ap}_{\mathrm{id}_A}(p) = p$

6 C++ and the J-Rule

From pre mathematical understanding we already know that the order of some A, B has nothing to do with the statement that A and B are equal. Which means Identity should be symmetric. Analogically Identity should be reflexive and transitive as well. Reflexivity of identity types holds by their construction. Symmetry and transitivity is implied by the J-Rule as shown in 1 on page 21 and 2 on page 22. Further the J-Rule implies LeastRefl.
Therefore identity in Martin Löf's type theory is similar to identity given by Lawvere's Law [6, Chapter 6].

As type theory is closer to computer science than set theory, it has a huge potential for programs.
It simplifies writing computer programs, which helps the mathematician writing proofs or which even construct full poofs on there own. The c++ code presented in 8 can generate proofs of many different types using the J-rule.
One only has to give the type in a certain form.
For some of the proofs above one can use the program below:

One can type in types similar to the following:

- $(x = y) > (y = x)$
- $Pi_(x, y : A)(x = y) > (y = z) > (x = z)$
- $p = p(\mathrm{dot})\mathrm{refl}_y$
- $p = \mathrm{refl}_x(\mathrm{dot})p$
- $(p)^-1(\mathrm{dot})p = \mathrm{refl}_y$
- $p(\mathrm{dot})(p)^-1 = \mathrm{refl}_x$
- $(p^-1)^-1 = p$
- $Pi_(ab, c)(ab = c) > p = \mathrm{refl}_c(\mathrm{dot})\mathrm{refl}_ab$
- $Pi_(ab, c : Z)(ab = c) > p = \mathrm{refl}_c(\mathrm{dot})\mathrm{refl}_ab$

- $Pi_(v,w)(v=w)=(w=v)$

The program uses string manipulation. When using strings C++ takes the input as a sequence of characters and saves every character on a byte in the same order, directly behind each other and safes an "end command" at the following byte.

The strings in this code are to represent types in a way a mathematician would write them down. So the relation between the program and type theory lays in the text representation.

In the program the input it saved at the string *type* and at the string *simpletype*. The first is just to remember the input, the second is used to construct an inhabitant. The program checks whether the input starts with $Pi_($, if yes the program assumes that it starts with $Pi_(...,... : ...)$ If this start is assumed, the characters between "(" and "," are saved as the string $x1$. The characters between "," and ":" are saved as the string $x2$ and the characters between ":" and ")" is saved at the string A. The start will be deleted from *simpletype*. Now the program checks whether *sympletype* starts with $(x1 = x2) >$ or not, if yes $(x1 = x2) >$ is deleted from the beginning of *simpletype*. Now *simpletype* represents the type to which U refers to in $C : \prod_{x,y:A}(x =_A y) \to U$.
If any of this fails the program tells that an error occured.
If the program does not assume this start. It safes $x1 = x$, $x2 = y$ and $A = A$. $x1$ and $x2$ are to represent the variables used in the type. It would be a bad idea to type in $Pi_(a,b,c : A)$ as b,c would become the name of the second variable. A is to represent the type inhabited by $x1$ and $x2$. The letter p refers to a possible proof that the types represented by $x1$ and $x2$ are equal.
The program gives as first output the type which inhabitans is to proof.
As it suffices to consider the case of reflexifity, the string part of *simpletype* which is equal to $x2$ is now replaced by the string $x1$. Than the letter p in *simpletype* is replaced by the string refl_$x1$. After that compositions of refl_$x1$ and its inverse are replaced by refl_$x1$. This is equal to the contraction of the path between equal points to a constant one. *simletype* is now simplified and represents only the case that the two variables are equal and the path is reflexifity.
Now the program tries to represent a function that maps the first variable so that the j-rule can be applied.
remember:
If there is

$$C : \prod_{x,y:A}(x =_A y) \to U$$

$$c : \prod_{x:A} C(x, x, \mathrm{refl}_x)$$

than, there exists

$$f : \prod_{x,y:A}\prod_{p:x=_A y} C(x, y, p)$$

$$f(x, x, \mathrm{refl}_x) :\equiv c(x).$$

To this point *simpletype* represents what $C(x, x, \mathrm{refl}_x)$ refers to in the formula above.
Now the program gains the information whether there is a character right in the middle of *simpletype* by counting the characters of *simpletype* and checking if the number is even or odd. If there is a character in the middle, the integer m will refer to its position within the string. Lets call this character [m]
If [m] is "=", than the program checks if the right side of [m] is identical to the left side. If yes, the program defines a function via text output, which maps a variable to a constant path between the types represented by the right and the left side of [m].
If [m] is ">", than the program checks if the right side of [m] is identical to the left side. If yes, the program defines a function via text output, which maps a variable to an identity function which maps the type represented by the right side of [m] to the type represented by the left side of [m].
If any of that fail, the program tells that an error occured. If no error occured The J-rule can be applied. The program, if it reaches this point, gives as last output:
"J-rule gives a function which existence is a proof for the type *type*"
type was the input made to the program at the very beginning.
One could say the program tries to write a proof. If the program was successful, the output of the program is a complete text based proof.

~/Dokumente : bash – Konsole

Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe

```
mdm53cu@cip69:~/Dokumente$ c++ type2.cpp
mdm53cu@cip69:~/Dokumente$ ./a.out
type:Pi_(u,v)(u=v)>(v=a)>(u=a)
C:Pi(u,v:A)(u=v) > U
C(u,v,p):= (v=a)>(u=a)
C(u,u,refl_u)=(u=a)>(u=a)
C(u,u,refl_u)=(u=a)>(u=a)
C(u,u,refl_u)=(u=a)>(u=a)
c=lamda u.id_{(u=a)} : Pi_(u:A) (u=a)>(u=a)
J-rule gives a function which existens is a proof for the type Pi_(u
,v)(u=v)>(v=a)>(u=a)
mdm53cu@cip69:~/Dokumente$
```

~/Dokumente : bash

~/Dokumente : bash – Konsole <2>

Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe

```
mdm53cu@cip69:~/Dokumente$ c++ type2.cpp
mdm53cu@cip69:~/Dokumente$ ./a.out
type:Pi_(x,y:A)(x=y)>p^-1(dot)refl_y=(refl_x)^-1(dot)p(dot)refl_y
C:Pi(x,y:A)(x=y) > U
C(x,y,p):= p^-1(dot)refl_y=(refl_x)^-1(dot)p(dot)refl_y
C(x,x,refl_x)=refl_x^-1(dot)refl_x=(refl_x)^-1(dot)refl_x(dot)refl_x
C(x,x,refl_x)=refl_x(dot)refl_x=refl_x(dot)refl_x(dot)refl_x
C(x,x,refl_x)=refl_x=refl_x
c=lamda x.refl_{refl_x} : Pi_(x:A) refl_x=refl_x
J-rule gives a function which existens is a proof for the type Pi_(x
,y:A)(x=y)>p^-1(dot)refl_y=(refl_x)^-1(dot)p(dot)refl_y
mdm53cu@cip69:~/Dokumente$
```

~/Dokumente : bash

7 Identity systems

At the very end of this essay I want to say something about the homotopical few of identity types.

Now we interpret a path $p : a =_A b$ as a continuous function between the to points a, b of the type A. Even though the constructor of identity types only enables us to construct the constant path refl_x for some type x, we can't say anymore that all path are equal to a constant one.

Definition 9 *A type A is contractible if it exists a point $a : A$ such that $a = x$ is inhabited for all $x : A$.*

$$\mathrm{isContr}(A) : \sum_{a:A} \prod_{x:A} a =_A x$$

We first describe the identity Type with the j-Rule, which is now called based path induction as we do homotopy. The identity Type is now regarded as a Family $P : A \to U$. We can deduce following results.

Definition 10 *For $a_0 : A$*

- *A pointed predicate over (A, a_0) is a family $R : A \to U$ with a point $r_0 : R(a_0)$.*

- *Let (R, r_0) and (S, s_0) be pointed predicates. A family*

$$g : \prod_{b:A} R(b) \to S(b)$$

is pointed if $g(a_0, r_0) = s_0$.

$$\mathrm{ppmap}(R, S) :\equiv \sum_{g:\prod_{b:A} R(b) \to S(b)} (g(a_0, r_0) = s_0)$$

- *Let (R, r_0) be a pointed predicate. If for all families*

$$D : \prod_{b:A} R(b) \to U$$

and $d : D(a_0, r_0)$, it exists a function

$$f : \prod_{b:A} \prod_{r:R(b)} D(b, r)$$

with $f(a_0, r_0) = d$ than (R, r_0) is an identity system.

Theorem 5 *For the pointed predicat (R, r_0), the following are equivalent:*

(i) (R, r_0) is an identity system at a_0.

(ii) For any pointed predicate (S, s_0), the type ppmap(R,S) is contractible.

(iii) For any $b : A$, the function

$$\mathit{transport}(-, r_0) : (a_0 =_A b) \to R(b)$$

is an equivalence.

(iv) The type $\sum_{b:A} R(b)$ is contractible.

(i) $\Rightarrow$ (ii):
Let (S, s_0) be a pointed predicat and define $D(b, r) :\equiv S(b)$ and $d :\equiv s_0 : S(a_0) \equiv D(a_0, r_0)$. Then we have

$$f : \prod_{b:A} R(b) \to S(b)$$

with $f(a_0, r_0) = s_0$. This means ppmap(R,S) is inhabited.
Let $(f, f_r), (g, g_r) : \text{ppmap}(R, S)$.
Define

$$D(b, r) :\equiv (f(b, r) = g(b, r))$$

and $d : f(a_0, r_0) = s_0 = g(a_0, r_0)$.
Now (i) gives a function

$$h : \prod_{b:A} \prod_{r:R(b)} D(b, r)$$

$$h : \prod_{b:A} \prod_{r:R(b)} (f(b, r) = g(b, r))$$

such that $h(a_0, r_0) = d$. $(f, f_r) = (g, g_r)$ holds and $\text{ppmap}(R, S)$ is contractible.

(ii) $\Rightarrow$ (iii):
Define $S(b) :\equiv (a_0 = b)$ with $s_0 :\equiv \text{refl}_{a_0} : S(a_0)$. Then (S, s_0) is a pointed predicat and

$$\lambda b.\lambda p.\text{transport}^R(p, r) : \prod_{b:A} S(b) \to R(b)$$

is a pointed family of maps from R to S. This means ppmap(S,R) is inhabited. ppmap(R,S) is inhabited by (ii). There fore we have a pointed family from R to S.
One composition is a pointed family from R to R which is equal to identity as ppmap(R,R) is contractible by (ii). Then (iii) holds.

(iii) $\Rightarrow$ (iv):
Define

$$f :\equiv \lambda b.\text{transport}(-, r_0) : \prod_{b:A} (a_0 =_A b) \to R(b)$$

which is an equivalence. Further define

$$\mathit{total}(f) :\equiv \lambda w.(\text{pr}_1 w, f(\text{pr}_1 w, \text{pr}_2 w)) : \sum_{b:A} (a_0 =_A b) \to \sum_{b:A} R(b)$$

which again is an equivalence.

$$\begin{aligned}
\mathrm{fib}_{\mathrm{total}(f)}((x,v)) &\equiv \sum_{w:\sum x:A(a_0=Ax)} (\mathrm{pr}_1 w, f(\mathrm{pr}_1 w, \mathrm{pr}_2 w)) = (x,v) \\
&\simeq \sum_{a:A} \sum_{u:(a_0=_A a)} (a, f(a,u)) = (x,v) \\
&\simeq \sum_{a:A} \sum_{u:(a_0=_A a)} \sum_{p:a=x} p_*(f(a,u)) = v \\
&\simeq \sum_{a:A} \sum_{p:a=x} \sum_{u:(a_0=_A a)} p_*(f(a,u)) = v \\
&\simeq \sum_{u:(a_0=_A a)} f(x,u) = v \\
&\equiv \mathrm{fib}_{f(x)}(v)
\end{aligned}$$

As $\mathrm{fib}_{\mathrm{total}(f)}((x,v))$ and $\mathrm{fib}_{f(x)}(v)$ are equal, $\mathrm{fib}_{f(x)}$ is an equivalence.
As $\sum_{b:A}(a_0 =_A b)$ is contractible by M for all $b : A$, we have that $\mathrm{fib}_{f(x)}$ is contractable.
Again because of the equivalence between $\mathrm{fib}_{\mathrm{total}(f)}(x,v)$ and $\mathrm{fib}_f(x)(v)$ we have that $\mathrm{fib}_{\mathrm{total}(f)}(w)$ is contractible. Therefore $\sum_{b:A} R(b)$ is contractable.

(iv) $\Rightarrow$ (i):
When we define $D' :\equiv \lambda w.D(\mathrm{pr}_1 w, \mathrm{pr}_2 w)$ we can express any

$$D : \prod_{b:A} R(b) \to U$$

as a family

$$D' : (\sum_{b:A} R(b)) \to U.$$

As $\sum_{b:A} R(b)$ is contractible, we have a

$$p : \prod_{u:\sum_{b:A} R(b)} (a_0, r_0) = u.$$

Define

$$f(u) :\equiv \mathrm{transport}^{D'}(p(u), d).$$

We have

$$f : \prod_{u:\sum_{b:A} R(b)} D'(u)$$

which is equivalent to

$$f : \prod_{b:A} \prod_{r:R(b)} D(b,r)$$

and

$$f(a_0, r_0) = \mathrm{transport}^{D'}(p((a_0, r_0), d) = d.$$

So (I) holds.

Now we describe the identity Type with the J-Rule, which is now called path induction. The identity Type is now regarded as a Family $P : A \to A \to U$. We can deduce following results.

Definition 11 *An identity system over a type A is a family*

$$R : A \to A \to U$$

with a function $r_0 : \prod_{a:A} R(a,a)$ such that for any type family

$$D : \prod_{a,b:A} R(a,b) \to U$$

and $d : \prod_{a:A} D(a,a,r_0(a))$, there exists a function

$$f : \prod_{a,b:A} \prod_{r:R(b)} D(a,b,r)$$

such that $f(a,a,r_0(a)) = d(a)$ for all $a : A$.

Theorem 6 *For $R : A \to A \to U$ with $r_0 : \prod_{a:A} R(a,a)$, the following are equivalent:*

(i) (R, r_0) is an identity system over A.

(ii) For all $a_0 : A$, the pointed predicate $(R(a_0), r_0(a_0))$ is an identity system at a_0.

(iii) For any $S : A \to A \to U$ and $s_0 : \prod_{a:A} S(a,a)$, type

$$\sum_{g:\prod_{a,b:A} R(a,b) \to S(a,b)} \prod_{a:A} g(a,a,r_0(a)) = s_0(a)$$

is contractible.

(iv) For any $a,b : A$, the function

$$transport(-, r_0(a)) : (a =_A b) \to R(a,b)$$

is an equivalence.

(v) For any $a : A$, the type $\sum_{b:A} R(a,b)$ is contractible.

(i) $\Rightarrow$ (ii) Fix any $a_0 : A$ at $R : A \to A \to U$ to get a family of functions

$$R_{a_0} : A \to U$$

and at

$$r_0 : \prod_{a:A} R(a,a) \equiv \prod_{a:A} R_a(a).$$

to get a function

$$r_{0(a_0)} : R_{a_0}(a_0).$$

Than $(R_{a_0}, r_{0(a_0)})$ is a pointed predicate.

For all $a : A$ take any family

$$D_a : \prod_{b:A} R_a(b) \to U$$

and $d_a : D_a(a_0, r_{0(a_0)})$.

Define $R :\equiv \lambda a.R_a$ and $r_0 :\equiv \lambda a.r_{0(a_0)}$.

Define $D :\equiv \lambda a.D_a$ and $r_0 :\equiv \lambda a.r_{0(a_0)}$.

Than (i) gives an

$$f : \prod_{a,b:A} \prod_{r:R(b)} D(a,b,r)$$

such that $f(a, a, r_0(a)) = d(a)$ for all $a : A$.
Fix any $a_0 : A$ at f to get

$$f_{a_0} : \prod_{b:A} \prod_{r:R(b)} D(a_0, b, r) \equiv \prod_{b:A} \prod_{r:R(b)} D_{a_0}(b, r)$$

such that $f_{a_0}(a_0, r_0(a_0)) = d(a)$ which is equal to
$f_{a_0}(a_0, r_{0(a_0)}) = d_a$.

(ii) $\Rightarrow$ (iii) We already know that for all $a_0 : A$ and any pointed predicate $(S_{a_0}, s_{0(a_0)})$ and a family of functions $g_{a_0} : \prod_{b:A} R_{a_0}(b) \to S_{a_0}(b)$ the type $\mathrm{ppmap}(R_{a_0}, S_{a_0})$ which can be written as

$$\sum_{g_{a_0}:\prod_{b:A} R_{a_0}(b) \to S_{a_0}(b)} (g_{a_0}(a_0, r_{o(a_0)}) = s_{0(a_0)})$$

is contractable.

Let $p_{a_0} : \mathrm{ppmap}(R_{a_0}, S_{a_0})$. Define $p :\equiv \lambda a.p_a$ and $g :\equiv \lambda a.g_a$.
Than

$$p : \prod_{a:A} \sum_{g_{a_0}:\prod_{b:A} R_{a_0}(b) \to S_{a_0}(b)} (g_{a_0}(a_0, r_{o(a_0)}) = s_{0(a_0)}),$$

which is equal to

$$\sum_{g:\prod_{a,b:A} R(a,b) \to S(a,b)} \prod_{a:A} (g(a)(a, r_{o(a)}) = s_{0(a)}),$$

which is equal to

$$\sum_{g:\prod_{a,b:A} R(a,b) \to S(a,b)} \prod_{a:A} (g(a, a, r_{o(a)}) = s_{0(a)}).$$

As every p_a is unique up to equality, p is unique up to equality and (III) holds.

(iii) $\Rightarrow$ (iv) Define $S(a, b) :\equiv (a =_A b)$ and $s_0 :\equiv \prod_{a:A} \mathrm{refl}_a : \prod_{a:A} S(a, a)$ and

$$h(r_0) :\equiv \lambda a.\lambda b.\lambda p.\mathrm{Transport}(p, r_0) : \prod_{a,b:A} S(a, b) \to R(a, b).$$

It holds that

$$h(r_0)(a_0, a_0, s_{0(a_0)}) = r_{0(a_0)}.$$

Now we have

$$\sum_{h(r_0):\prod_{a,b:A} S_(a,b)\to R_(a,b)} \prod_{a:A}(h(r_0)(a,a,s_{0(a)}) = r_{0(a)})$$

is inhabited. We also know that there exists a

$$k : \sum_{g:\prod_{a,b:A} R_(a,b)\to S_(a,b)} \prod_{a:A}(g(a,a,r_{o(a)}) = s_{0(a)}).$$

The composition $h(r_0) \bullet k$ is a function of the type

$$\sum_{i:\prod_{a,b:A} R_(a,b)\to R_(a,b)} \prod_{a:A}(i(a,a,r_{o(a)}) = r_{0(a)})$$

which we know to be contractable and inhabited by the identity function. Therefore $h(r_0)$ is equal to identity. Than for all $a, b : A$ the function $h(r_0)(a, b, p)$ is equal to identity or with other words

$$\text{Transport}(-, r_{0(a)}) : S(a, b) \to R(a, b)$$

is an equivalence.

(iv) $\Rightarrow$ (v) Fix $a : A$, than for any $b : A$ the type

$$\text{Transport}(-, r_{0(a)}) : S_a(b) \to R_a(b)$$

is an equivalence and we get from 5 that the type $\sum_{b:A} R_a(b)$ is contractable. So for any $a : A$ the type $\sum_{b:A} R(a, b)$ is contractable.

(v) $\Rightarrow$ (i) For any family

$$D : \prod_{a,b:A} R(a, b) \to U$$

and $d : \prod_{a:A} D(a, a, r_{0(a)})$
we have for any $a : A$ the family

$$D_a :\equiv D(a) : \prod_{b:A} R_a(b) \to U$$

and $d_a :\equiv d(a) : D(a, a, r_{0(a)})$.
As $\sum_{b:A} R_a(b)$ is contractable for any $a : A$, it follows from 5 that $(R_a, r_{0(a)})$ is an identity system at $a : A$ for any $a : A$.
So for any $a : A$ there exists a function

$$f_a : \prod_{b:A} \prod_{r:R_a(b)} D_a(b, r)$$

such that $f_a(a, r_{0(a)}) = d_a$.
Define

$$f :\equiv \lambda a.f_a : \prod_{a,b:A} \prod_{r:R(a,b)} D(a, b, r).$$

For all $a : A$

$$f(a, a, r_{0(a)}) = d(a)$$

So (R, r_0) is an identity system over A.

After [5] Chapter 5; p.174 the univalence axiom says that a type

$$(- \simeq -) : U \to U \to U$$

with $\mathrm{id} : \prod_{A:U}(A \simeq A)$ satisfies Theorem 6(IV).
Therefore it is equivalent to the corresponding version of 6(I). Which is:
For any

$$D : \prod_{A,B:U} (a \simeq B) \to U$$

and $d : \prod_{A:U} D(A, A, \mathrm{id}_A)$, there exists a function

$$f : \prod_{A,B:U} \prod_{e:A \simeq B} D(A, B, e)$$

such that $f(A, A, id_A) = d(A)$ for all $A : U$.

Further [5] Chapter 5; p.174 states that for any $B : A \to U$, the type family

$$(- \sim -) : (\prod_{a:A} B(a)) \to (\prod_{a:A} B(a)) \to U$$

with $\lambda f.\lambda a.\mathrm{refl}_f(a)$ satisfies 6(IV).
So it is equivalent to the corresponding version of 6(I). Which is:
For any

$$D : \prod_{f,g:\prod_{a:A} B(a)} (f \sim g) \to U$$

and $d : \prod_{f:\prod_{a:A} B(a)} D(f, f, \lambda x.\mathrm{refl}_{f(x)})$, there exists a function

$$k : \prod_{f,g:\prod_{a:A} B(a)} \prod_{h:f \sim g} D(f, g, h)$$

such that $k(f, f, \lambda x.\mathrm{refl}_{f(x)}) = d(f)$ for all f.

8 Appendix

Save the code in a file type with the ending ”*.cpp*”. Open the file with an editor ("cd.file address/" than "c++ file name.cpp") and start the program ("./a.out").

```
using namespace std;

#include <iostream>
#include <stdlib.h>
#include <string>

int main(){

 int fail=0;
 int l, lx1, lx2;
 std::string A;
 std::string type;

 std::string simpletype;
 std::string x1, x2, help, intro;
 std::string c, f;

 std::cout << "type:";

 std::cin >> type;       // The input is saved as string

 simpletype = type;
 l=type.length();
 x1 = type;
                            // in the following  "Pi_(...,...:...)"
 if(simpletype[0]=='P' && simpletype[1]=='i'
                       && simpletype[2]=='_'&& simpletype[3]=='('){
   std::size_t found = simpletype.find(')');
   if(found == std::string::npos) fail=5;                        //error 5
   else{
    simpletype.erase(0,found+1);
    x1.erase(found,l-found);
    x1.erase(0,4);
    found = x1.find(',');
    if(found!=std::string::npos){
      x2=x1;
      l=x1.length();
      x1.erase(found,l-found);   //x1 is the string of the name
                                 //of the first variable.
      x2.erase(0,found+1);
      l=x2.length();
      found= x2.find(':');
      if(found!=std::string::npos) {
A=x2;
```

```
A.erase(0,found+1);       //A is a string for the type of the
                          //two variables.
x2.erase(found,l-found);  //x2 is the string of the name
                          //of the second variable.
      }

    }
   }
   if(x2.length()!=0) {
     intro=")>";
     intro.insert(0,x2);
     intro.insert(0,"=");
     intro.insert(0,x1);
     intro.insert(0,"(");
     found = simpletype.find(intro);
     if(found==0){
       simpletype.erase(found,intro.length());
     }
   }
 }
 else {
   A="A";
   x1="x";
   x2="y";
   intro="(x,y)>";
 }
 if(A.length()==0) A="A";

 if(simpletype[0]=='('){
   std::size_t found=simpletype.find(")");
   std::size_t lsimpletype=simpletype.length();
   if(found == lsimpletype-1){
     simpletype.erase(lsimpletype-1,1);
     simpletype.erase(0,1);
   }
 }
// the string simpletype correspond to a type.
// We want to show that this type is inhabited.

 std::size_t lintro=intro.length();
 help=intro;
 help.erase(lintro-1,1);
 std::cout << "C:Pi("<< x1 <<","<< x2 <<":"<<A<<")"<< help <<" > U\n";
 std::cout << "C("<<x1<<","<<x2<<",p):= " << simpletype <<"\n";
 // this output represents a family of functions.
 // In the following changes are made to simpletype such that
 // it represents the output of the family of functions
 // depending two times on the first variable and reflexivity
 // as proof that the variable is equal to itself.
```

```
l = simpletype.length();
lx1= x1.length();
lx2= x2.length();

  // The second variable is replaced by the first
for(int i=0; i!=l; ++i) {
  std::size_t found=simpletype.find(x2);
  if(found!=std::string::npos){
    simpletype.erase(found,lx2);
    simpletype.insert(found,x1);   // The second variable is
                                   //replaced by the first.
    l=simpletype.length();
  }
}

help=x1;
help.insert(0,"refl_");

for(int i=0; i!=l; ++i) {
  if(simpletype[i]=='p') {   // the character p is the
                             // presetting for a proof of
                             // first variable equals the second.
    simpletype.erase (i,1),
    simpletype.insert(i,help), // p is replaced by reflexivity.
    l = simpletype.length();
  }
}
std::string refl=help;
std::cout << "C(" << x1 << "," << x1 << "," << refl << ")=" << simpletype <<"\n";
 // This output represents the family of functions which depend
 //  two times on the first variable and reflexivity.

       // As refl_x1^-1 equals refl_x1 it can be replaced.
std::string toreplace2="^-1";
toreplace2.insert(0,refl);
std::size_t lrefl=refl.length();
for(int i=0; i<l; ++i) {
 std::size_t found = simpletype.find(toreplace2);
 if (found==std::string::npos) break;
 else {
       // replace refl_x1^-1 in simpletype by refl_x1.
   simpletype.erase (found,lrefl+3),
   simpletype.insert(found,refl),
   i+=lrefl, l=simpletype.length();
 }
}
      // same step as before for different notation
      // (refl_x1)^-1 = refl_x1
std::string toreplace=")^-1";
toreplace.insert(0,refl);
```

```
toreplace.insert(0,"(");

for(int i=0; i<l; ++i) {
 std::size_t found = simpletype.find(toreplace);
 if (found==std::string::npos) break;
 else {
         // lösche (refl_x1)^-1 und setze refl_x1 ein
   simpletype.erase (found,lrefl+5),
   simpletype.insert(found,refl),
   i+=lrefl-1, l=simpletype.length();
 }
}

std::cout << "C(" << x1 << "," << x1 << "," << refl <<")="<< simpletype << "\n";

   // As refl_x1(dot)refl_x1 equals refl_x1 it can be
   // replaced.
std::string toreplacerefl=refl;
toreplacerefl.insert(0,"(dot)");
toreplacerefl.insert(0,refl);

for(int i=0; i<l; ++i) {
 std::size_t found = simpletype.find(toreplacerefl);
 if (found==std::string::npos) break;
 else {
   // replace refl_x1(dot)refl_x1 by refl_x1 in simpletype.
   simpletype.erase (found,lrefl+lrefl+5),
   simpletype.insert(found,refl),
   l=simpletype.length();
 }
}

std::cout << "C(" << x1 << "," << x1 << "," << refl <<")="<< simpletype << "\n";

 // To apply the J-rule one needs a function which maps
 // the first variable to the type represented by simpletype.
 // The code tries to gain a representation of that function.
 // First by representing a function which maps to reflexivity
 // and if it fails by representing an identity function.

 int m, z, found;
 std::string start=simpletype;
 std::string final;
 z=start.length();
 if (z%2!=0){  //z is even
  m=(z-1)/2;
  if (start[m]=='='){
    start.erase(m,z+1-m);    //string till =
    final=start;
    final.insert(0,"=");
```

```
      final.insert(0,start);
      if (final==simpletype)
       std::cout << "c=lamda "<< x1 << ".refl_{" << start << "} :
               Pi_("<< x1 << ":"<< A << ") "<< simpletype <<"\n";
      else fail=2;
    }
    if (start[m]=='>'){
      start.erase(m,z+1-m);   //string bis >
      final=start;
      final.insert(0,">");
      final.insert(0,start);
      if (final==simpletype)
       std::cout << "c=lamda " << x1 << ".id_{" << start << "} :
             Pi_(" << x1 << ":" << A<< ") "<< simpletype <<"\n";
      else fail=2;
    }
  }
  else fail=1;
  if (fail!=0) std::cout << "error="<< fail <<"\n" ;
  else std::cout << "J-rule gives a function which existence is a proof for
                                                the type " << type <<"\n";
return 0;}
```

As this program gives the proof with steps, it is easy to check if the program did well.

for "$a \to b$" type "a>b"; for "$p \bullet p$" use "p(dot)p";

References

[1] T. Coquand: "A remark on singleton types", manuscript, 2014.

[2] J. Ladyman and Stuart Presnell: "Identity in Homotopy Type Theory, Part I: The Justification of Path Induction", Philosophia Mathematica (III) Vol. 23, No.3, 2015, 386-406.

[3] C. Paulin-Mohring: "Inductive Definitions in the System Coq - Rules and Properties", in M. Bezem, J. F. Groote (Eds.) Proceedings of TLCA, LNM 664, Springer, 1993.

[4] I. Petrakis: "A Yoneda lemma-formulation of the univalence axiom", preprint, 2018.

[5] The Univalent Foundations Program: *Homotopy Type Theory: Univalent Foundation of Mathematics*, Institute for Advanced Study, Princeton, 2013.

[6] P. Walsh: *Justifying Path Induction*, Master's thesis, CMU, 2015.