D.D. Otten - daniel@otten.co

Type Theory: $\mathbb{M}$-types and Bisimulation

Bachelor thesis

30 June 2020

Thesis supervisors: Dr. H. Basold
Dr. P.J. Bruin

Leiden University
Mathematical Institute
Leiden Institute of Advanced Computer Science

Contents

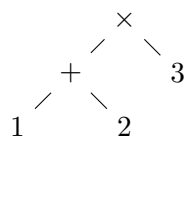
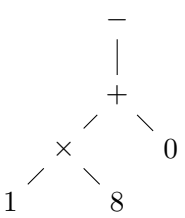
Chapter 0

Introduction and Related Work

Many structures in mathematics and computer science can be represented as trees. Natural numbers for example can be seen as nonbranching trees where the number corresponds to the length of the tree. Similarly strings can be seen as nonbranching trees with the nodes labelled by characters.

0	$*$	`"let"`	`'l'` — `'e'` — `'t'`
2	$* — * — *$	`"type"`	`'t'` — `'y'` — `'p'` — `'e'`
5	$* — * — * — * — * — *$	`"lambda"`	`'l'` — `'a'` — `'m'` — `'b'` — `'d'` — `'a'`

Expressions are another example, they can be represented as labelled binary trees.

2	$(1+2)\times 3$	$-(1\times 8+0)$
2		

In type theory we can use these tree representations to define structures in the form of W-types. We define a W-type by specifying the allowed labels and specifying the number of branches for every label. This gives the type of all finite-depth trees made using those specifications.

W-types are a special case of initial algebras, which is a concept from category theory. We can dualize W-types and initial algebras to get M-types and final coalgebras respectively. M-types can also be seen as a types consisting of trees. Here the trees have the same labels and branching behaviour. The difference is that an M-type also includes infinite-depth trees instead of only finite-depth trees like the corresponding W-type.

As an example, consider lists of natural numbers. We can define the type of potentially infinite lists as an M-type. Here we let nodes be labelled with a number or with END. A node labelled with a number should have one subtree while a node labelled with END should have no subtrees.

END

$19 — 12 — 8 -$ END

$92 — 0 — 56 — 1 — 22 — 9 — 892 — 2 — 71 — 7 — \cdots$

W-types and M-types allow us to create new types using existing ones. Another way to create new types is using $\sum$-types and $\prod$-types; these can be seen as the disjoint union and Cartesian product respectively. We will see that $\sum$-types are initial algebras while $\prod$-types are final coalgebras. To summarise, we now have:

Initial Algebras	Seen as	Final Coalgebras	Seen as
$\sum$-types	disjoint union	$\prod$-types	Cartesian product
W-types	type of finite-depth trees	M-types	type of all trees

In type theory two axioms we can include are function extensionality and univalence. The axiom of function extensionality is commonly assumed and states that two functions are equal if they are equal on every argument. The axiom of univalence is the basis of univalent type theory which is an area of inspired by homotopy theory. It is a well known result that univalence implies function extensionality. We will consider this result from the perspective of final coalgebras.

For coalgebras we can define special relations called bisimulations. Two members of a coalgebra are called bisimilar if they are related by a bisimulation. Function extensionality now shows that bisimilarity implies equality for $\prod$-types. This motivates our research question:

Assuming univalence: does bisimilarity imply equality for M-types?

We approached this question by examining different proofs that univalence implies function extensionality. We tried to translate these proofs from the case of $\prod$-types to the case of M-types. Here the specific definition of M-types became quite important. We will consider five different definitions of M-types and show the implications between these definitions. We will see for each of these definitions that bisimilarity implies equality. For this we will only need the function extensionality axiom; we do not require the univalence axiom.

In this thesis we start in chapter 1 by giving an explanation of Martin-Löf type theory, among other things we explain $\sum$-types and $\prod$-types and the different axioms we can assume. In chapter 2 we cover the concepts of algebras and coalgebras in a general category and more specifically W-types and M-types in type theory. In chapter 3 we show how $\sum$-types can be seen as initial algebras and how $\prod$-types can be seen as final coalgebras. We define bisimilarity for $\prod$-types and give two proofs that univalence implies function extensionality. After this in chapter 4 we will give two ways to define relations in type theory and two corresponding ways to define bisimulation for M-types. This allows us to give five different definitions of M-types. We show the implications between these definitions. We will see for each of these definitions that bisimilarity implies equivalence.

The results of this thesis have been formalised in the dependently typed programming language and proof-assistant `Agda`, for this see appendix A.

For related work see [AGS12] where the univalence axiom is used to prove that W-types are equivalent to initial algebras for polynomial functors. This result is used in [Uni13, section 5.5] to state three equivalent definitions of W-types. It is proven that M-types are derivable from W-types in [AAG05] and with a different methodology in [BM07]. See also [Sta11] for different ways to define bisimulations for coalgebras and the implications between these definitions.

Chapter 1

Type Theory

1.0 Summary

In this chapter we give a brief explanation of Martin-Löf's intuitionistic type theory [ML84] based on [Uni13]. In section 1.1 we explain the basic ideas of type theory and how it can be used as a formal foundation for mathematics. After this we start with the concept of a type universe in section 1.2. We use section 1.3 to explain $\sum$-types and $\prod$-types and in section 1.4 we explain $+$-types, $\times$-types, and $\to$-types. These are all ways to define new types using already existing types. We will give the basic initial types $\mathbb{0}$, $\mathbb{1}$, and $\mathbb{N}$ in section 1.5, with these all other types can be made. In section 1.7 we describe equality and in section 1.8 we will define some useful constructs that we will need in later chapters. We conclude this chapter by stating possible axioms for type theory in section 1.9.

1.1 Formal Foundation

Types are similar to sets and can likewise be used as a foundation of mathematics. First we give an explanation of the way set theory is defined to show how this differs for type theory. A set theory is typically built on top of a logic. The most notable example is Zermelo-Freankel set theory with the axiom of choice (ZFC) [Jec03], it can be built by stating axioms in first-order logic. A logic can be defined by giving a number of inference rules. The inference rule with name 'Rule' which says that from the premises $p_1, \ldots, p_m$ we can deduce the conclusions $c_1, \ldots, c_n$ is written as

$$\frac{p_1 \quad \ldots \quad p_m}{c_1 \quad \ldots \quad c_n}(\text{Rule}).$$

Modus ponens is an example of such an inference rule in propositional logic, it is given by

$$\frac{A \implies B \quad A}{B}(\text{MP}),$$

and says that if A implies B and we have A then we can deduce B. Another example on an inference rule is the law of the excluded middle, given by

$$\frac{}{A \vee \neg A}(\text{LEM})$$

which states that without any premises we can conclude that A is true or A is not true.

After we define our logic by stating inference rules we define our set theory by taking a number of axioms, these are statements we assume without proof. A formal proof is now an ordered list

of statements such that each statement can be deduced from earlier statements or axioms by some inference rule.

Type theory in contrast is not built on top of logic but is instead constructed from the ground up using inference rules. In this chapter we will introduce intensional Martin-Löf type theory in an informal way inspired by [Uni13, chapter 1]. For a fully formal approach using inference rules see for example [Uni13, appendix A.1 or A.2].

A type is inhabited by terms. We write $a : A$ if a is a term of the type A. Unlike in set theory where elements can by part of multiple sets a term generally inhabits only one type. Note that this is the way that types work in programming languages. In the language `C` [Cpp20] we have for instance the type `float` with terms like `0.5f` and `-6.28f` while the type `char` has terms like `'d'` and `'Y'`. In type theory we have two different kinds of equality. We use the symbol $\equiv$ for definitional equality, if we have $A \equiv B$ then every time we encounter A it can be replaced by B and likewise every time we encounter B it can be replaced by A. The symbol $=$ is reserved for propositional equality which works differently in type theory, we will see this in section 1.7.

In this thesis we will use the word 'postulate' for statements that are formally built with inference rules although we will not state these inference rules explicitly. We will reserve the word 'axiom' for assumptions that only introduce a new term of an already existing type.

1.2 Universes

Including a universe in type theory has the same concerns about size that we have in set theory. In set theory, it would be nice to include the axiom scheme of comprehension which says that for every predicate ϕ we can make the set $\{x : \phi(x)\}$ of all sets for which ϕ holds. Russell [ID16] however showed that this axiom scheme is unsound, with it we can create the set $P = \{x : x \notin x\}$ of all sets that do not contain themselves. By the law of the excluded middle we would have $P \in P$ or $P \notin P$. But $P \in P$ implies $P \notin P$ while $P \notin P$ implies $P \in P$ which leads to a contradiction. Therefore we have to restrict this axiom to the axiom scheme of specification which only allows creating subsets of existing sets. It says that for every set X and every predicate ϕ we can create the subset $\{x \in X : \phi(x)\}$. But with this axiom we can no longer have a set V that contains all sets. This is because with this set we would again be able to create $P = \{x \in V : x \notin x\}$.

In type theory it is likewise not possible to have a type of all types as it would allow us to encode Russell's paradox [Coq92]. Nevertheless it would be very useful to have such a universe Ty, this would for example allow us to give an infinite number of types by giving a function $f : \mathbb{N} \to \mathrm{Ty}$. Therefore we copy the approach of [Uni13] and introduce a hierarchy of universes

$$\mathrm{Ty}_0 : \mathrm{Ty}_1 : \mathrm{Ty}_2 : \ldots$$

For every ℓ equal to 0, 1, 2, ... we have that Ty_ℓ is itself a type and has other types as terms. We assume that these universes are cumulative, the terms of Ty_ℓ are also terms of $\mathrm{Ty}_{\ell+1}$. This does not run into problems because Ty_ℓ is not a term of itself only a term of higher universes, we do not have a biggest universe that contains everything. The downside of this approach is that terms no longer have an unique type and that a type now can itself be a term. This complicates the theory and gives a lot of new edge cases.

The other option is to have the terms of Ty_ℓ not be types themselves but to include an operator $\mathrm{term\_to\_type}_\ell$ such that for the term $X : \mathrm{Ty}_\ell$ we have a type $\mathrm{term\_to\_type}_\ell\, X$. Instead of having cumulative universes we then take a coercion function $\mathrm{coerce}_\ell : \mathrm{Ty}_\ell \to \mathrm{Ty}_{\ell+1}$. Therefore terms still have unique types and cannot be types themselves. Both $\mathrm{term\_to\_type}_\ell$ and coerce_ℓ can be left implicit when working informally.

To reduce clutter during this thesis we will write Ty instead of Ty_ℓ, leaving the ℓ implicit. Every theorem or proposition holds for every ℓ. For example we are now able to define the type $\mathbb{N} \to \mathrm{Ty}$ which we understand as being equal to $\mathbb{N} \to \mathrm{Ty}_\ell$ for some ℓ. We have that $\mathbb{N} \to \mathrm{Ty}_\ell$ is a term of $\mathrm{Ty}_{\ell+1}$. In our formalisation in `Agda` we do use these universe levels explicitly, see appendix A.

1.3 $\sum$-types, $\prod$-types

Now we will discuss ways to define new types using existing ones. We do this first because we need $\sum$-types and $\prod$-types when postulating the initial types $\mathbb{0}$, $\mathbb{1}$, and $\mathbb{N}$. If we have a type A and for every $a\colon A$ a type B_a then we can form $\sum_{(a\colon A)} B_a$ as a new type.[0] We can see this as the type of dependent pairs, its terms are of the form $(a,\ b)$ where $a\colon A$ and $b\colon B_a$. Another useful way to view $\sum_{(a\colon A)} B_a$ is as the disjoint union of the types B_a for $a\colon A$.

Postulate. ($\sum$-types) Suppose that $A\colon \mathrm{Ty}$ and for every $a\colon A$ that $B_a\colon \mathrm{Ty}$. We get a type

$$\sum_{(a\colon A)} B_a\colon \mathrm{Ty}; \qquad \text{(formation)}$$

and if we have $a\colon A$ and $b\colon B_a$ then we get a term

$$(a,\ b)\colon \sum_{(a\colon A)} B_a. \qquad \text{(introduction)}$$

If $w\colon \sum_{(a\colon A)} B_a$ then we get terms

$$\begin{aligned} &\mathrm{pr}_0\ w\colon A, \\ &\mathrm{pr}_1\ w\colon B_{\mathrm{pr}_0\, w}, \end{aligned} \qquad \text{(elimination)}$$

such that for every $a\colon A$ and $b\colon B_a$ we get

$$\begin{aligned} &\mathrm{pr}_0\ (a,\ b) \equiv a, \\ &\mathrm{pr}_1\ (a,\ b) \equiv b. \end{aligned} \qquad \text{(computation}^1\text{)}$$

For every $w\colon \sum_{(a\colon A)} B_a$ we have

$$(\mathrm{pr}_0\ w,\ \mathrm{pr}_1\ w) \equiv w. \qquad \text{(uniqueness}^2\text{)}$$

We will allow ourselves to use pattern matching on $\sum$-types, when doing this instead of declaring a new variable w by writing $w\colon \sum_{(a\colon A)} B_a$ we may declare two new variables a and b by writing $(a,\ b)\colon \sum_{(a\colon A)} B_a$. This does not increase expressiveness and is best understood by seeing a few examples. If we have for every $a\colon A$ and $b\colon B_a$ a type $C_{a,b}$ and a term $c_{a,b}\colon C_{a,b}$, then we may write

$$\begin{aligned} \textstyle\sum_{((a,b)\colon \sum_{(a\colon A)} B_a)} C_{a,b} \quad &\text{instead of} \quad \textstyle\sum_{(w\colon \sum_{(a\colon A)} B\, a)} C_{\mathrm{pr}_0\, w,\, \mathrm{pr}_1\, w}, \\ \textstyle\prod_{((a,b)\colon \sum_{(a\colon A)} B_a)} C_{a,b} \quad &\text{instead of} \quad \textstyle\prod_{(w\colon \sum_{(a\colon A)} B\, a)} C_{\mathrm{pr}_0\, w,\, \mathrm{pr}_1\, w}, \\ \lambda(a,\ b).\, c_{a,b} \quad &\text{instead of} \quad \lambda w.\, c_{\mathrm{pr}_0\, w,\, \mathrm{pr}_1\, w}; \end{aligned}$$

which already shows the next way to define new types.

[0] We use the notation B_a to indicate a free variable $a\colon A$. In a context with $a\colon A$ we have a type B_a.
[1] Also known as β-reduction.
[2] Also known as η-reduction.

If we have a type A and for every $a\colon A$ a type B_a then we can form the new type $\prod_{(a\colon A)} B_a$. We can see $\prod_{(a\colon A)} B_a$ as the type of dependent functions, its terms are of the form $\lambda a.\, b_a$ (also written as $a \mapsto b_a$) where for every $a\colon A$ we have that $b_a\colon B_a$. Another way to view $\prod_{(a\colon A)} B_a$ is as the Cartesian product of the types B_a for $a\colon A$.

Postulate. ($\prod$-types) Suppose that $A\colon \mathrm{Ty}$ and for every $a\colon A$ that $B_a\colon \mathrm{Ty}$. We get a type

$$\prod_{(a\colon A)} B_a\colon \mathrm{Ty}; \qquad \text{(formation)}$$

and if we have for every $a\colon A$ that $b_a\colon B_a$ then we get a term

$$\lambda a.\, b_a\colon \prod_{(a\colon A)} B_a. \qquad \text{(introduction)}$$

If $f\colon \prod_{(a\colon A)} B_a$ and $a\colon A$ then we get a term

$$f\ a\colon B_a, \qquad \text{(elimination)}$$

such that if we have for every $a\colon A$ that $b_a\colon B_a$ then we get

$$(\lambda a.\, b_a)\ a \equiv b_a. \qquad \text{(computation}^0\text{)}$$

For every $f\colon \prod_{(a\colon A)} B_a$ we have

$$\lambda a.\, f\ a \equiv f. \qquad \text{(uniqueness}^1\text{)}$$

Suppose that we have for every $a\colon A$ a term $b_a\colon B_a$. Then instead of using λ-notation to define $f\colon \prod_{(a\colon A)} B_a$ as $f \equiv \lambda a.\, b_a$ we may define this dependent function by writing

$$\begin{aligned} &f\colon \textstyle\prod_{(a\colon A)} B_a,\\ &f\ a \equiv b_a. \end{aligned}$$

Note that this is merely a different notation.

In type theory it is standard to let $\sum$, $\prod$, and λ automatically scope as far as possible, for example $\sum_{(a\colon A)} \prod_{(b\colon B_a)} B_a \to \mathrm{Ty}$ means $\sum_{(a\colon A)} (\prod_{(b\colon B_a)} (B_a \to \mathrm{Ty}))$ and $\lambda a.\, \lambda b.\, \lambda c.\, a$ means $\lambda a.\, (\lambda b.\, (\lambda c.\, a))$. Another convention is that for a type A and for every a_0, $a_1\colon A$ a type B_{a_0, a_1} we may write

$$\begin{aligned} \textstyle\sum_{(a_0, a_1\colon A)} B_{a_0, a_1} \quad &\text{instead of} \quad \textstyle\sum_{(a_0\colon A)} \sum_{(a_1\colon A)} B_{a_0, a_1},\\ \textstyle\prod_{(a_0, a_1\colon A)} B_{a_0, a_1} \quad &\text{instead of} \quad \textstyle\prod_{(a_0\colon A)} \prod_{(a_1\colon A)} B_{a_0, a_1}. \end{aligned}$$

We denote pairing right associative and function application left associative, therefore we may write

$$\begin{aligned} (a,\, b,\, c) \quad &\text{instead of} \quad (a,\, (b,\, c)),\\ f\, a\, b \quad &\text{instead of} \quad (f\, a)\, b. \end{aligned}$$

$\sum$-types and $\prod$-types are an important part of type theory. They are used very often which is why we define these alternative notations. Such notations are known as syntactic sugar, they make things easier to read and more concise without increasing the expressiveness.

[0] Also known as β-reduction.
[1] Also known as η-reduction.

1.4 $+$-types, $\times$-types, $\to$-types

If we have a type A and a type B then we can form a new type $A + B$. We can see $A + B$ as the disjoint union of A and B, its terms are of the form $\mathrm{in}_0\, a$ for $a\colon A$ or $\mathrm{in}_1\, b$ for $b\colon B$.

Postulate. ($+$-types) Suppose that $A\colon \mathrm{Ty}$ and $B\colon \mathrm{Ty}$. We get a type

$$A + B\colon \mathrm{Ty}; \qquad \text{(formation)}$$

and we can construct terms using

$$\begin{aligned} &\mathrm{in}_0\colon \textstyle\prod_{(a\colon A)} A + B, \\ &\mathrm{in}_1\colon \textstyle\prod_{(b\colon B)} A + B. \end{aligned} \qquad \text{(introduction)}$$

We have a dependent function

$$\mathrm{ind}_{A+B}\colon \textstyle\prod_{(P\colon \prod_{(s\colon A+B)}\mathrm{Ty})} \prod_{(l\colon \prod_{(a\colon A)} P\,(\mathrm{in}_0\, a))} \prod_{(r\colon \prod_{(b\colon B)} P\,(\mathrm{in}_1\, b))} \prod_{(s\colon A+B)} P\ s, \qquad \text{(elimination)}$$

such that we have

$$\begin{aligned} &\mathrm{ind}_{A+B}\ P\ l\ r\ (\mathrm{in}_0\ a) \equiv l\ a, \\ &\mathrm{ind}_{A+B}\ P\ l\ r\ (\mathrm{in}_1\ b) \equiv r\ b. \end{aligned} \qquad \text{(computation)}$$

For $P\colon \prod_{(s\colon A+B)}\mathrm{Ty}$ we will use ind_{A+B} implicitly. We will allow ourselves to define a term of $\prod_{(s\colon A+B)} P\, s$ by taking $l\colon \prod_{(a\colon A)} P\,(\mathrm{in}_0\, a)$ and $r\colon \prod_{(b\colon B)} P\,(\mathrm{in}_1\, b)$ and writing

$$\begin{aligned} &f\colon \textstyle\prod_{(s\colon A+B)} P\ s, \\ &f\ (\mathrm{in}_0\ a) \equiv l\ a, \\ &f\ (\mathrm{in}_1\ b) \equiv r\ b; \end{aligned}$$

which is the same as taking $f \equiv \mathrm{ind}_{A+B}\, P\, l\, r$. We call this defining f by induction on $A + B$.

Suppose that we have a type A and for every $a\colon A$ a type B_a. Then we have already noted that $\sum_{(a\colon A)} B_a$ can be seen as the type of dependent pairs. If we have a constant type B then $\sum_{(a\colon A)} B$ is the type of pairs over A and B. Therefore we will take the following definition.

Definition. ($\times$-types) For $A\colon \mathrm{Ty}$ and $B\colon \mathrm{Ty}$ we define the type $A \times B\colon \mathrm{Ty}$ as

$$A \times B \equiv \textstyle\sum_{(a\colon A)} B.$$

Similarly $\prod_{(a\colon A)} B_a$ can be seen as the type of dependent functions. Now if we again have a constant type B then we get that $\prod_{(a\colon A)} B$ is the type of functions from A to B.

Definition. ($\to$-types) For $A\colon \mathrm{Ty}$ and $B\colon \mathrm{Ty}$ we define the type $A \to B\colon \mathrm{Ty}$ as

$$A \to B \equiv \textstyle\prod_{(a\colon A)} B.$$

We will denote $+$, $\times$, and $\to$ all right associative, therefore for A, B, C: Ty we may write

$$\begin{array}{lll} A+B+C & \text{instead of} & A+(B+C), \\ A\times B\times C & \text{instead of} & A\times(B\times C), \\ A\to B\to C & \text{instead of} & A\to(B\to C). \end{array}$$

1.5 Basic Types

Now that we have seen multiple ways to construct new types using existing ones we need a few initial types to start this process. First we consider the empty type $\mathbb{0}$. There are no ways to construct a term of $\mathbb{0}$, at least if the type theory presented here is consistent. This is of course always difficult because of Gödels second incompleteness theorem [Raa20] which shows that a consistent system which contains elementary arithmetic cannot show its own consistency. However in [KL12] it is shown that the type theory given here, including the univalence axiom of section 1.9 is consistent if ZFC with two inaccessible cardinals is consistent.

Postulate. ($\mathbb{0}$-type) We have a type

$$\mathbb{0}\colon \mathrm{Ty}. \tag{formation}$$

and no way to construct terms. We have a dependent function

$$\mathrm{ind}_{\mathbb{0}}\colon \textstyle\prod_{(P\colon \mathbb{0}\to \mathrm{Ty})}\prod_{(i\colon \mathbb{0})} P\ i. \tag{elimation}$$

Similarly we postulate a type $\mathbb{1}$ with a single term $0_{\mathbb{1}}$.

Postulate. ($\mathbb{1}$-type) We have a type

$$\mathbb{1}\colon \mathrm{Ty}; \tag{formation}$$

and we have a term

$$0_{\mathbb{1}}\colon \mathbb{1}. \tag{introduction}$$

We have a dependent function

$$\mathrm{ind}_{\mathbb{1}}\colon \textstyle\prod_{(P\colon \mathbb{1}\to \mathrm{Ty})} P\ 0_{\mathbb{1}} \to \prod_{(i\colon \mathbb{1})} P\ i, \tag{elimination}$$

such that

$$\mathrm{ind}_{\mathbb{1}}\ P\ p_0\ 0_{\mathbb{1}} \equiv p_0. \tag{computation}$$

One option would be to generalise these types and also postulate $\mathbb{2}$, $\mathbb{3}$, $\mathbb{4}$, ... as shown below.

Postulate.[0] ($\mathbb{n}$-type) For n equal to 0, 1, 2, ... we have a type

$$\mathbb{n}\colon \mathrm{Ty}; \qquad \text{(formation)}$$

and we have terms

$$\begin{aligned} 0_{\mathbb{n}} &\colon \mathbb{n}, \\ 1_{\mathbb{n}} &\colon \mathbb{n}, \\ 2_{\mathbb{n}} &\colon \mathbb{n}, \\ &\vdots \quad \vdots \\ (n-1)_{\mathbb{n}} &\colon \mathbb{n}. \end{aligned} \qquad \text{(introduction)}$$

We can have a dependent function

$$\mathrm{ind}_{\mathbb{n}}\colon \textstyle\prod_{(P\colon \mathbb{n}\to \mathrm{Ty})} P\ 0_{\mathbb{n}} \to P\ 1_{\mathbb{n}} \to P\ 2_{\mathbb{n}} \to \cdots \to P\ (n-1)_{\mathbb{n}} \to \prod_{(i\colon \mathbb{n})} P\ i, \qquad \text{(elimination)}$$

such that

$$\begin{aligned} &\mathrm{ind}_{\mathbb{n}}\ P\ p_0\ p_1\ p_2\ \ldots\ p_{n-1}\ 0_{\mathbb{n}} \equiv p_0, \\ &\mathrm{ind}_{\mathbb{n}}\ P\ p_0\ p_1\ p_2\ \ldots\ p_{n-1}\ 1_{\mathbb{n}} \equiv p_1, \\ &\mathrm{ind}_{\mathbb{n}}\ P\ p_0\ p_1\ p_2\ \ldots\ p_{n-1}\ 2_{\mathbb{n}} \equiv p_2, \\ &\quad\vdots \\ &\mathrm{ind}_{\mathbb{n}}\ P\ p_0\ p_1\ p_2\ \ldots\ p_{n-1}\ (n-1)_{\mathbb{n}} \equiv p_{n-1}. \end{aligned} \qquad \text{(computation)}$$

However this is not necessary because we have already postulated $\mathbb{0}$, $\mathbb{1}$, and $+$-types. As we see on the next page we will be able to take

$$\begin{aligned} \mathbb{2} &\equiv \mathbb{1} + \mathbb{1}, \\ \mathbb{3} &\equiv \mathbb{1} + \mathbb{1} + \mathbb{1}, \\ \mathbb{4} &\equiv \mathbb{1} + \mathbb{1} + \mathbb{1} + \mathbb{1}, \\ &\vdots \quad \vdots \end{aligned}$$

Alternatively we could have postulated $\mathbb{0}$, $\mathbb{1}$, and $\mathbb{2}$ and defined $+$-types. To do this for A and B we would define $A + B \equiv \sum_{(i\colon \mathbb{2})} C_i$ where $C_{0_{\mathbb{2}}} \equiv A$ and $C_{1_{\mathbb{2}}} \equiv B$. Then we would take

$$\begin{aligned} &\mathrm{in}_0\colon A \to A + B, && \mathrm{in}_1\colon B \to A + B, \\ &\mathrm{in}_0\ a \equiv (0_{\mathbb{2}},\ a); && \mathrm{in}_1\ b \equiv (1_{\mathbb{2}},\ b). \end{aligned}$$

Similarly we could have defined $\times$-types by defining for A and B that $A \times B \equiv \prod_{(i\colon \mathbb{2})} C_i$ where $C_{0_{\mathbb{2}}} \equiv A$ and $C_{1_{\mathbb{2}}} \equiv B$. Then for $a\colon A$ and $b\colon B$ we would define $(a,\ b) \equiv \lambda i.\, c_i$ where $c_{0_{\mathbb{2}}} \equiv a$ and $c_{1_{\mathbb{2}}} \equiv b$. Lastly we would take

$$\begin{aligned} &\mathrm{pr}_0\colon A \times B \to A, && \mathrm{pr}_1\colon A \times B \to B, \\ &\mathrm{pr}_0\ p \equiv p\ 0_{\mathbb{2}}; && \mathrm{pr}_1\ p \equiv p\ 1_{\mathbb{2}}. \end{aligned}$$

[0] Note that we do not include this postulate in our type theory!

Instead we use $\mathbb{0}$, $\mathbb{1}$, and $+$-types to take the following definition.

Definition. ($\mathbb{n}$-types) For n equal to 2, 3, ... we define the type

$$\mathbb{n} \equiv \underbrace{\mathbb{1} + \mathbb{1} + \mathbb{1} + \cdots + \mathbb{1}}_{n}$$

and the terms

$$\begin{aligned}
0_{\mathbb{n}} &\equiv \mathrm{in}_0\, 0_{\mathbb{1}}, \\
1_{\mathbb{n}} &\equiv \mathrm{in}_1\, (\mathrm{in}_0\, 0_{\mathbb{1}}), \\
2_{\mathbb{n}} &\equiv \mathrm{in}_1\, (\mathrm{in}_1\, (\mathrm{in}_0\, 0_{\mathbb{1}})), \\
&\vdots \qquad \vdots \\
(n-1)_{\mathbb{n}} &\equiv \underbrace{\mathrm{in}_1\, (\mathrm{in}_1\, (\mathrm{in}_1\, (\ldots\, (\mathrm{in}_1}_{n-1}\, 0_{\mathbb{1}}) \ldots))).
\end{aligned}$$

We have a dependent function

$$\begin{aligned}
&\mathrm{ind}_{\mathbb{n}} \colon \textstyle\prod_{(P \colon \mathbb{n} \to \mathrm{Ty})} P\, 0_{\mathbb{n}} \to P\, 1_{\mathbb{n}} \to P\, 2_{\mathbb{n}} \to \cdots \to P\, (n-1)_{\mathbb{n}} \to \prod_{(i \colon \mathbb{n})} P\, i, \\
&\mathrm{ind}_{\mathbb{n}} \equiv \lambda P.\, \lambda p_0.\, \lambda p_1.\, \lambda p_2.\, \ldots\, \lambda p_{n-1}.\, (\\
&\quad \mathrm{ind}_{\underbrace{\mathbb{1}+\mathbb{1}+\mathbb{1}+\cdots+\mathbb{1}}_{n}}\, P\, (\mathrm{ind}_{\mathbb{1}}\, P\, p_0)\, (\\
&\qquad \mathrm{ind}_{\underbrace{\mathbb{1}+\mathbb{1}+\mathbb{1}+\cdots+\mathbb{1}}_{n-1}}\, P\, (\mathrm{ind}_{\mathbb{1}}\, P\, p_1)\, (\\
&\qquad\quad \mathrm{ind}_{\underbrace{\mathbb{1}+\mathbb{1}+\mathbb{1}+\cdots+\mathbb{1}}_{n-2}}\, P\, (\mathrm{ind}_{\mathbb{1}}\, P\, p_2)\, (\\
&\qquad\qquad \ddots \\
&\qquad\qquad\quad \mathrm{ind}_{\mathbb{1}+\mathbb{1}}\, P\, (\mathrm{ind}_{\mathbb{1}}\, P\, p_{n-2})\, (\\
&\qquad\qquad\qquad \mathrm{ind}_{\mathbb{1}}\, P\, p_{n-1} \\
&\qquad\qquad\quad) \\
&\qquad\qquad \iddots \\
&\qquad\quad) \\
&\qquad) \\
&\quad) \\
&).
\end{aligned}$$

For n equal to 0, 1, 2, ... and $P : \mathbb{n} \to \mathrm{Ty}$ we will use $\mathrm{ind}_{\mathbb{n}}$ implicitly. We allow ourselves to define a term of $\prod_{(i:\mathbb{n})} P\, i$ by taking $p_0 : P\, 0_{\mathbb{n}}$, $p_1 : P\, 1_{\mathbb{n}}$, $p_2 : P\, 1_{\mathbb{n}}$, ..., $p_{n-1} : P\, (n-1)_{\mathbb{n}}$ and writing

$$\begin{aligned} &f : \textstyle\prod_{(i:\mathbb{n})} P\, i, \\ &f\, 0_{\mathbb{n}} \equiv p_0, \\ &f\, 1_{\mathbb{n}} \equiv p_1, \\ &f\, 2_{\mathbb{n}} \equiv p_2, \\ &\vdots \\ &f\, (n-1)_{\mathbb{n}} \equiv p_{n-1}; \end{aligned}$$

which would be the same as taking $f \equiv \mathrm{ind}_{\mathbb{n}}\, P\, p_0\, p_1\, p_2\, \ldots\, p_{n-1}$. We call this induction on $\mathbb{n}$.

Notice that $\mathbb{0}$, $\mathbb{1}$, $\mathbb{2}$, ... are all types with a finite number of terms. If A and B are finite types then we have that $A + B$, $A \times B$, and $A \to B$ are also finite. And similarly if A is a finite type and for every $a : A$ we have a finite type B_a then $\sum_{(a:A)} B_a$ and $\prod_{(a:A)} B_a$ are also finite. To allow types with an infinite number of terms we will therefore need an initial infinite type. For this we take the type $\mathbb{N}$ of natural numbers. To avoid postulating an infinite number of terms we will postulate an initial term $0_{\mathbb{N}} : \mathbb{N}$ and a function $\mathrm{suc} : \mathbb{N} \to \mathbb{N}$. A natural number is of the form $0_{\mathbb{N}}$ or $\mathrm{suc}\, n$ for a natural number n. We define

$$\begin{aligned} &1_{\mathbb{N}} \equiv \mathrm{suc}\, 0_{\mathbb{N}}, \\ &2_{\mathbb{N}} \equiv \mathrm{suc}\, (\mathrm{suc}\, 0_{\mathbb{N}}), \\ &3_{\mathbb{N}} \equiv \mathrm{suc}\, (\mathrm{suc}\, (\mathrm{suc}\, 0_{\mathbb{N}})), \\ &\vdots \qquad\quad \vdots \end{aligned}$$

Postulate. ($\mathbb{N}$-type) We have a type

$$\mathbb{N} : \mathrm{Ty}; \tag{formation}$$

and we can construct terms using

$$\begin{aligned} 0_{\mathbb{N}} &: \mathbb{N}, \\ \mathrm{suc} &: \mathbb{N} \to \mathbb{N}. \end{aligned} \tag{introduction}$$

We have a dependent function

$$\mathrm{ind}_{\mathbb{N}} : \textstyle\prod_{(P:\mathbb{N}\to\mathrm{Ty})} P\, 0_{\mathbb{N}} \to (\prod_{(n:\mathbb{N})} P\, n \to P\, (\mathrm{suc}\, n)) \to \prod_{(n:\mathbb{N})} P\, n, \tag{elimination}$$

such that

$$\begin{aligned} &\mathrm{ind}_{\mathbb{N}}\, P\, p_0\, s\, 0_{\mathbb{N}} \equiv p_0, \\ &\mathrm{ind}_{\mathbb{N}}\, P\, p_0\, s\, (\mathrm{suc}\, n) \equiv s\, n\, (\mathrm{ind}_{\mathbb{N}}\, p_0\, s\, n). \end{aligned} \tag{computation}$$

For $P\colon \mathbb{N} \to \mathrm{Ty}$ we will use $\mathrm{ind}_{\mathbb{N}}$ implicitly. For $p_0\colon P\,0_{\mathbb{N}}$ and $s\colon \prod_{(n\colon \mathbb{N})} P\,n \to P\,(\mathrm{suc}\,n)$ we will allow ourselves to define a new dependent function as follows

$$\begin{aligned} &f\colon \textstyle\prod_{(n\colon \mathbb{N})} P\,n,\\ &f\,0 \equiv p_0,\\ &f\,(\mathrm{suc}\,n) \equiv s\,n\,(f\,n). \end{aligned}$$

This can be translated to taking $f \equiv \mathrm{ind}_{\mathbb{N}}\,p_0\,s$. We call this induction on $\mathbb{N}$.

1.6 Types as Propositions

In section 1.1 we have noted that type theory is not built on a logic. However after we have built type theory from the ground up we do get a logic as a part of this theory. For this we view types as propositions: the type A is seen as true if there exists a term $a\colon A$ and as false if such a term does not exist. Therefore we view

$$\begin{array}{lll} \mathbb{0} & \text{as} & \bot,\\ \mathbb{1} & \text{as} & \top; \end{array}$$

because $\mathbb{0}$ is the canonical type without terms and $\mathbb{1}$ is the canonical type with terms.

Suppose that A and B are types, then we view

$$\begin{array}{lll} A + B & \text{as} & A \;\vee\; B,\\ A \times B & \text{as} & A \;\wedge\; B,\\ A \to B & \text{as} & A \implies B. \end{array}$$

Indeed, we see that a term of $A+B$ is of the form $\mathrm{in}_0\,a\colon A+B$ where $a\colon A$ or of the form $\mathrm{in}_1\,b\colon A+B$ where $b\colon B$. Therefore $A + B$ has a term if and only if A or B has a term. Likewise, a term of $A \times B$ is of the form $(a,\, b)$ where $a\colon A$ and $b\colon B$. Therefore $A \times B$ has a term if and only if A and B have terms. Lastly we look at $A \to B$, this type has terms of the form $\lambda a.\, b_a$ where for $a\colon A$ we have $b_a\colon B$. To get a term of $A \to B$ we must send every term of A to a term of B. Therefore $A \to B$ has a term if and only if we have that a term of A implies a term of B. Note in particular that for every type A we have $\mathrm{ind}_{\mathbb{0}}\,(\lambda i.\, A)\colon \mathbb{0} \to A$. This shows the principle of *ex falso*, from a contradiction anything follows.

Now suppose that A is a type and for every $a\colon A$ we have a type B_a. Then we view

$$\begin{array}{lll} \sum_{(a\colon A)} B_a & \text{as} & \exists\, a \in A\colon B_a,\\ \prod_{(a\colon A)} B_a & \text{as} & \forall\, a \in A\colon B_a. \end{array}$$

Indeed a term of $\sum_{(a\colon A)} B_a$ is of the form $(a,\, b)$ where $a\colon A$ and $b\colon B_a$. Therefore $\sum_{(a\colon A)} B_a$ has a term if and only if there exists an $a\colon A$ such that B_a has a term. Likewise a term of $\prod_{(a\colon A)} B_a$ is of the form $\lambda a.\, b_a$ where for $a\colon A$ we have $b_a\colon B_a$. Therefore $\prod_{(a\colon A)} B_a$ has a term if and only if for every $a\colon A$ we have that B_a has a term.

The one remaining thing is to define is negation, if A is a type then we view

$$A \to \mathbb{0} \quad \text{as} \quad \neg A.$$

Note that in logic for a proposition A one often defines $\neg A$ as $A \implies \bot$.

Now we have that giving a proof for a proposition is the same as giving a term of a certain type. This is very useful and is the reason why type theory is the basis of many proof assistants, notably

of `Agda` [Nor07, Agd20] and `Coq` [Coq04]. The logic we get in this way is called constructive or intuitionistic. Constructive logic different from first-order logic, most notably it does not contain the law of the excluded middle which states

$$\prod_{(A\colon \mathrm{Ty})} A + (A \to \mathbb{0}) \quad \text{viewed as} \quad \forall A\colon A \vee \neg A.$$

The law of the excluded middle is logically equivalent to double negation elimination which states

$$\prod_{(A\colon \mathrm{Ty})} ((A \to \mathbb{0}) \to \mathbb{0}) \to A \quad \text{viewed as} \quad \forall A\colon \neg\neg A \implies A.$$

Because we do not have double negation elimination we cannot give a proof by contradiction. This is because if we want to prove A then a proof by contradiction would show $\neg\neg A$. It is possible to add a version of the law of the excluded middle as an axiom, see section 1.9. In this thesis however we will not do this, we will not need it and this keeps our results more general.

1.7 Equality

Equality in type theory works differently from equality in first-order logic and set theory. We have already seen definitional equality, written with the symbol $\equiv$. This equality works as we expect, if we have $A \equiv B$ then we can replace every occurrence of A with B and every occurrence of B with A. However, because logic is a part of our type theory it is not possible to prove statements for definitional equality. This is because $A \equiv B$ is not a type; we cannot prove that it is true by giving a term. Therefore we introduce propositional equality, written with the symbol $=$. If we have a type A and we have $a, b\colon A$ then we get a type $a = b$. For $a\colon A$ we get a term $\mathrm{refl}\, a\colon (a = a)$. It is important that $a = b$ is a type because this allows us to use it in propositions. For example we can express the statement that there exists a natural number that is equal to $0_{\mathbb{N}}$ as $\sum_{(n\colon \mathbb{N})}(n = 0_{\mathbb{N}})$. This statement is true because we have $(0_{\mathbb{N}}, \mathrm{refl}\, 0_{\mathbb{N}})\colon \sum_{(n\colon \mathbb{N})}(n = 0_{\mathbb{N}})$.

Postulate. (equality) If $A\colon \mathrm{Ty}$ and $a, b\colon A$ then we have a type

$$(a = b)\colon \mathrm{Ty}; \qquad \text{(formation)}$$

and we can construct terms using

$$\mathrm{refl}\colon \prod_{(a\colon A)}(a = a). \qquad \text{(introduction)}$$

We have a dependent function

$$\mathrm{ind}^{=}_{A}\colon \prod_{(P\colon \prod_{(a,b\colon A)}(a=b))}\left(\prod_{(a\colon A)} P\ a\ a\ (\mathrm{refl}\ a)\right) \to \prod_{(a,b\colon A)}\prod_{(p\colon a=b)} P\ a\ b\ p, \qquad \text{(elimination)}$$

such that

$$\mathrm{ind}^{=}_{A}\ P\ d\ a\ a\ (\mathrm{refl}\ a) \equiv d\ a. \qquad \text{(computation)}$$

For $P\colon \prod_{(a,b\colon A)} a = b$ we will allow ourselves to use $\mathrm{ind}^{=}_{A}$ implicitly. For $d\colon \prod_{(a\colon A)} P\, a\, a\, (\mathrm{refl}\, a)$ we can define a term of $\prod_{(a,b\colon A)}\prod_{(p\colon a=b)} P\, a\, b\, p$ by writing

$$f\colon \prod_{(a,b\colon A)}\prod_{(p\colon a=b)} P\ a\ b\ p,$$
$$f\ a\ a\ (\mathrm{refl}\ a) \equiv d\ a;$$

which would be the same as taking $f \equiv \mathrm{ind}^{=}_{A}\, d$. Defining such a function implicitly is called defining with path induction, this name comes from the intuition that a term of $a = b$ can be seen as a path from a to b. We will discuss this perspective below.

Proposition 1.7.0. For $A\colon \mathrm{Ty}$ we have that $=$ is an equivalence relation, that is, we have

$$\begin{aligned}&\mathrm{refl}\colon \textstyle\prod_{(a\colon A)}(a = a),\\ &\mathrm{sym}\colon \textstyle\prod_{(a,b\colon A)}(a = b) \to (b = a),\\ &\mathrm{trans}\colon \textstyle\prod_{(a,b,c\colon A)}(a = b) \to (b = c) \to (a = c).\end{aligned}$$

Proof. We have postulated refl directly. Now we use path induction to define

$$\begin{aligned}&\mathrm{sym}\colon \textstyle\prod_{(a,b\colon A)}(a = b) \to (b = a),\\ &\mathrm{sym}\ a\ a\ (\mathrm{refl}\ a) \equiv \mathrm{refl}\ a;\end{aligned}$$

and we use path induction twice to define

$$\begin{aligned}&\mathrm{trans}\colon \textstyle\prod_{(a,b,c\colon A)}(a = b) \to (b = c) \to (a = c),\\ &\mathrm{trans}\ a\ a\ a\ (\mathrm{refl}\ a)\ (\mathrm{refl}\ a) \equiv \mathrm{refl}\ a.\end{aligned}$$

Note that the dependent functions refl, sym, and trans show that $=$ is reflexive, symmetric, and transitive respectively. Therefore $=$ is, as we would expect, an equivalence relation. □

Suppose that we have a, b, $c\colon A$ and that we have $p\colon a = b$ and $q\colon b = c$. It is helpful to think of p as a path from a to b and q as a path from b to c. Then $\mathrm{refl}\, a$ is seen as the constant path at a, $\mathrm{sym}\, a\, b\, p$ is seen as the inverse path of p, and $\mathrm{trans}\, a\, b\, c\, p\, q$ is seen as the concatenation of the paths p and q.

Definition 1.7.1 If $A\colon \mathrm{Ty}$ then for a, b, $c\colon A$ with $p\colon a = b$ and $q\colon b = c$ we define the notation

$$\begin{aligned}\mathrm{refl}_a &\equiv \mathrm{refl}\ a,\\ p^{-1} &\equiv \mathrm{sym}\ a\ b\ p,\\ p \cdot q &\equiv \mathrm{trans}\ a\ b\ c\ p\ q.\end{aligned}$$

After seeing path induction it would be natural to expect for a, $b\colon A$ that every term of $a = b$ is of the form refl_a. We will develop some intuition for why this does not have to be the case.

First note that $a = b$ is once again a type and therefore for p_0, $p_1 : a = b$ we can make the type $p_0 = p_1$. More generally suppose that we have a_0, $b_0\colon A$ with $p_0\colon a_0 = b_0$ and suppose that we have a_1, $b_1\colon A$ with $p_1\colon a_1 = b_1$. Then we cannot make a type $p_0 = p_1$ directly because p_0 and p_1 are not necessarily terms of the same type. However we can make the type $(a_0, b_0, p_0) = (a_1, b_1, p_1)$ because (a_0, b_0, p_0) and (a_1, b_1, p_1) are both terms of the type $\sum_{(a,b\colon A)}(a = b)$. We can use path induction to define the dependent function

$$\begin{aligned}&f\colon \textstyle\prod_{(a,b\colon A)}\prod_{(p\colon a=b)}((a,\ b,\ p) = (a,\ a,\ \mathrm{refl}\ a)),\\ &f\ a\ a\ (\mathrm{refl}\ a) = \mathrm{refl}\ (a,\ a,\ \mathrm{refl}\ a);\end{aligned}$$

which proves for every a, $b\colon A$ and $p\colon a = b$ that $(a, b, p) = (a, a, \mathrm{refl}\, a)$.

We can also define

$$g\colon \prod_{(a_0, b_0 : A)} \prod_{(p_0 : a_0 = b_0)} \prod_{(a_1, b_1 : A)} \prod_{(p_1 : a_1 = b_1)} ((a_0,\ b_0,\ p_0) = (a_1,\ b_1,\ p_1)),$$
$$g\ a_0\ b_0\ p_0\ a_1\ b_1\ p_1 \equiv (f\ a_0\ b_0\ p_0) \cdot (f\ a_1\ b_1\ p_1)^{-1};$$

which proves for every $a_0, b_0 \colon A$ with $p_0 \colon a_0 = b_0$ and for every $a_1, b_1 \colon A$ with $p_1 \colon a_1 = b_1$ that $(a_0, b_0, p_0) = (a_1, b_1, p_1)$.

We cannot however prove for every $p \colon a = a$ that $p = \mathrm{refl}_a$ or for every $p_0, p_1 \colon a = b$ that $p_0 = p_1$. To make this intuitive it is again helpful to see p_0, p_1 as paths. We can see $\sum_{(a, b : A)} (a = b)$ as the type of paths in the space A. A path can be seen as a continuous transformation from one point to another point. A path between paths is a continuous transformation from one path to another path. If we have $a, b \colon A$ and $p \colon a = b$ then we can continuously transform the path p to the constant path refl_a by retracting the endpoint b. This can be done within $\sum_{(a, b : A)} (a = b)$ because it is the space of all paths regardless of endpoints. This makes it intuitive that $(a, b, p) = (a, a, \mathrm{refl}\, a)$.

In the same way for $a, b \colon A$ the type $a = b$ can be seen as the type of paths between a and b. However now for $p_0, p_1 \colon a = b$ a continuous transformation from p_0 to p_1 has to leave the endpoints intact. This is because such a transformation must happen within $a = b$ which only consists of paths with endpoints a and b. There are spaces and paths for which such a continuous transformation does not exist. An easy example is the circle with the constant path and the path that goes counterclockwise around the circle.

With the postulates we have stated there is no way to construct $p, q \colon a = b$ with $p \neq q$ however it is consistent to add such paths to our theory. The univalence axiom which we will consider in section 1.9 is one way to make such paths.

1.8 Useful Constructs

In this section we will define some types and functions that are useful when working in type theory. Most of these functions will be defined with path induction. They also have topological interpretations, for those we refer to [Uni13, chapter 2]. First we will prove the following proposition; for this we will assume the reader is familiar with the basics of category theory, that is, the definition of a category and the definition of a functor.

Proposition 1.8.0. Ty forms a category where objects are types $A \colon \mathrm{Ty}$ and for $A, B \colon \mathrm{Ty}$ the morphisms from A to B are functions $f \colon A \to B$.

Proof. For $A \colon \mathrm{Ty}$ we define the function

$$\mathrm{id}_A \colon A \to A,$$
$$\mathrm{id}_A \equiv \lambda a.\ a.$$

Suppose that we have $A, B, C \colon \mathrm{Ty}$ with $f \colon A \to B$ and $g \colon B \to C$, then we define

$$(g \circ f) \colon A \to C,$$
$$(g \circ f) \equiv \lambda a.\ g\ (f\ a).$$

We can also define this for dependent functions, suppose that we have $A \colon \mathrm{Ty}$, $B \colon A \to \mathrm{Ty}$, and $C \colon \prod_{(a : A)} B\, a \to \mathrm{Ty}$. Then for $f \colon \prod_{(a : A)} B\, a$ and $g \colon \prod_{(a : A)} \prod_{(b : B\, a)} C\, a\, b$ we define

$$(g \circ f) \colon \prod_{(a : A)} C\ a\ (f\ a),$$
$$(g \circ f) \equiv \lambda a.\ g\ a\ (f\ a).$$

Now to see that we get a category we note for A, $B\colon \mathrm{Ty}$ and $f\colon A \to B$ that

$$\begin{aligned} &\mathrm{id}_A \circ f \equiv \lambda a.\ \mathrm{id}_A\ (f\ a) \equiv \lambda a.\ f\ a \equiv f, \\ &f \circ \mathrm{id}_A \equiv \lambda a.\ f\ (\mathrm{id}_A\ a) \equiv \lambda a.\ f\ a \equiv f; \end{aligned}$$

and for A, B, C, $D\colon \mathrm{Ty}$ with $f\colon A \to B$, $g\colon B \to C$, $h\colon C \to D$ that

$$h \circ (g \circ f) \equiv h \circ (\lambda a.\ g\ (f\ a)) \equiv \lambda a.\ h\ (g\ (f\ a)) \equiv (\lambda b.\ h\ (g\ b)) \circ f \equiv (h \circ g) \circ f.$$

This proves that Ty is a category for definitional equality, denoted by $\equiv$. If $A \equiv B$ then we also get $A = B$. This is because $\mathrm{refl}_a\colon A = A$ and we can replace one occurrence of A with B to get $\mathrm{refl}_a\colon A = B$. Therefore Ty is also a category for propositional equality, denoted by $=$. □

The first dependent function we will consider is called tra which is short for 'transport'.

Definition. (tra) Suppose that $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$, then for a_0, $a_1\colon A$ we define

$$\begin{aligned} &\mathrm{tra}_B\colon \textstyle\prod_{(p\colon a_0 = a_1)} B\ a_0 \to B\ a_1, \\ &\mathrm{tra}_B\ \mathrm{refl}_a \equiv \mathrm{id}_{(B\,a)}\,. \end{aligned}$$

Next we show below that functions preserve equality. The names of these definitions come from 'application' and from 'dependent application' respectively.

Definition. (ap) Suppose that $f\colon A \to B$, then for a_0, $a_1\colon A$ we define

$$\begin{aligned} &\mathrm{ap}_f\colon (a_0 = a_1) \to (f\ a_0 = f\ a_1); \\ &\mathrm{ap}_f\ \mathrm{refl}_a \equiv \mathrm{refl}_{(f\,a)}. \end{aligned}$$

Definition. (apd) Suppose that $f\colon \prod_{(a\colon A)} B\,a$, then for a_0, $a_1\colon A$ we define

$$\begin{aligned} &\mathrm{apd}_f\colon \textstyle\prod_{(p\colon a_0 = a_1)} \mathrm{tra}_B\ p\ (f\ a_0) = f\ a_1; \\ &\mathrm{apd}_f\ \mathrm{refl}_a \equiv \mathrm{refl}_{(f\,a)}. \end{aligned}$$

We give a way to construct equalities of $\sum$-types and a way to destruct equalities of $\prod$-types. These are analogous to the functions

$$\begin{aligned} &\mathrm{pair}\colon \textstyle\prod_{(a\colon A)} B\ a \to (\sum_{(a\colon A)} B\ a), \\ &\mathrm{pair}\ a\ b \equiv (a,\ b); \end{aligned}$$

$$\begin{aligned} &\mathrm{apply}\colon (\textstyle\prod_{(a\colon A)} B\ a) \to \prod_{(a\colon A)} B\ a, \\ &\mathrm{apply}\ f\ a \equiv f\ a. \end{aligned}$$

Definition. (pair$^=$) Suppose that w_0, $w_1 : \sum_{(a:A)} B\,a$, then we define

$$\begin{aligned}&\mathrm{pair}^=_{w_0,w_1} : (\textstyle\sum_{(p:\mathrm{pr}_0\,w_0 = \mathrm{pr}_0\,w_1)}(\mathrm{tra}_B\;p\;(\mathrm{pr}_1\;w_0) = \mathrm{pr}_1\;w_1)) \to (w_0 = w_1),\\ &\mathrm{pair}^=_{w_0,w_1}\;(\mathrm{refl}_{(\mathrm{pr}_0\,w)},\;\mathrm{refl}_{(\mathrm{pr}_1\,w)}) \equiv \mathrm{refl}_w.\end{aligned}$$

Definition. (apply$^=$) Suppose that f_0, $f_1 : \prod_{(a:A)} B\,a$, then we define

$$\begin{aligned}&\mathrm{apply}^=_{f_0,f_1} : (f_0 = f_1) \to (\textstyle\prod_{(a:A)} f_0\;a = f_1\;a),\\ &\mathrm{apply}^=_{f_0,f_1}\;\mathrm{refl}_f\;a \equiv \mathrm{refl}_{(f\,a)}.\end{aligned}$$

We will sometimes leave subscripts implicit if they are clear from context. For example we may write apply$^=$ instead of $\mathrm{apply}^=_{f_0,f_1}$ if both f_0 and f_1 can be induced.

The next concept we define in this chapter is that of an equivalence, which corresponds to the concept of a bijection in set theory. We consider the following two definitions.

Definition. (quasi-inverse) For types A and B and a function $f : A \to B$ we define

$$\mathrm{Qinv}\;f \equiv \textstyle\sum_{(g:B\to A)}\;(\prod_{(a:A)} g\;(f\;a) \equiv a) \times (\prod_{(b:B)} f\;(g\;b) \equiv b).$$

Definition. (equivalence) For types A and B and a function $f : A \to B$ we define

$$\mathrm{IsEqui}\;f \equiv (\textstyle\sum_{(g:B\to A)}\prod_{(a:A)} g\;(f\;a) \equiv a) \times (\sum_{(g:B\to A)}\prod_{(b:B)} f\;(g\;b) \equiv b);$$

and we define

$$A \simeq B \equiv \textstyle\sum_{(f:A\to B)}\mathrm{IsEqui}\;f.$$

The advantage of taking $\mathrm{IsEqui}\,f$ over $\mathrm{Qinv}\,f$ is that for every e_0, $e_1 : \mathrm{IsEqui}\,f$ we have $e_0 = e_1$, the same is not true for $\mathrm{Qinv}\,f$ as shown in [Uni13] section 2.4. We do however have functions between $\mathrm{IsEqui}\,f$ and $\mathrm{Qinv}\,f$. We can define

$$\begin{aligned}&\mathrm{Qinv\_to\_IsEqui} : \mathrm{Qinv}\;f \to \mathrm{IsEqui}\;f,\\ &\mathrm{Qinv\_to\_IsEqui}\;(f,\;\mathrm{hom}_0,\;\mathrm{hom}_1) \equiv ((g,\;\mathrm{hom}_0),\;(g,\;\mathrm{hom}_1));\\ &\\ &\mathrm{IsEqui\_to\_Qinv} : \mathrm{IsEqui}\;f \to \mathrm{Qinv}\;f,\\ &\mathrm{IsEqui\_to\_Qinv}\;((g_0,\;\mathrm{hom}_0),\;(g_1,\;\mathrm{hom}_1)) \equiv (\\ &\qquad g_0,\\ &\qquad \mathrm{hom}_0,\\ &\qquad \lambda b.\;\mathrm{ap}_f\;(\mathrm{ap}_{g_0}\;(\mathrm{hom}_1\;b)^{-1} \cdot \mathrm{hom}_0\;(g_1\;b)) \cdot \mathrm{hom}_1\;b\\ &).\end{aligned}$$

Because of these functions we can prove that f is an equivalence by giving a quasi inverse of f.

Lastly we will show that equal types are equivalent.

Definition. For A_0, $A_1 \colon \mathrm{Ty}$ we define

$$\begin{aligned}&\mathrm{path\_to\_equi} \colon (A_0 = A_1) \to (A_0 \simeq A_1),\\ &\mathrm{path\_to\_equi}\ \mathrm{refl}_A \equiv ((\mathrm{id}_A,\ \lambda a.\ \mathrm{refl}_a),\ (\mathrm{id}_A,\ \lambda a.\ \mathrm{refl}_a)).\end{aligned}$$

1.9 Axioms

In this section we will discuss possible axioms to include in type theory. We start with the axiom of function extensionality, this is the only axiom that we will assume throughout this thesis.

Axiom. (function extensionality) Suppose that $A \colon \mathrm{Ty}$ and $B \colon A \to \mathrm{Ty}$, then for every f_0, $f_1 \colon \prod_{(a\colon A)} B\,a$ the function

$$\mathrm{apply}^{=}_{f_0,\,f_1} \colon (f_0 = f_1) \to (\textstyle\prod_{(a\colon A)} f_0\ a = f_1\ a)$$

is an equivalence.

Note that we do not need to include an axiom to prove for w_0, $w_1 \colon \sum_{(a\colon A)} B\,a$ that $\mathrm{pair}^{=}_{w_0,\,w_1}$ is an equivalence, this can be done without additional axioms, see [Uni13, theorem 2.7.2].

For f_0, $f_1 \colon \prod_{(a\colon A)} B\,a$ we will use the function extensionality axiom to define

$$\mathrm{funext}_{f_0,\,f_1} \colon (\textstyle\prod_{(a\colon A)} f_0\ a = f_1\ a) \to (f_0 = f_1)$$

as the quasi-inverse of $\mathrm{apply}^{=}_{f_0,\,f_1}$.

The next axiom we will consider is known as axiom K. The name comes from the fact that $\mathrm{ind}^{=}_{A}$ is historically known as J with K being the next letter in the alphabet. This axiom was first introduced in [Str93].

Axiom. (K) Suppose that $A \colon \mathrm{Ty}$ and $a \colon A$, then we have

$$\mathrm{ind}^{=}_{a} \colon \textstyle\prod_{(P\colon (a=a)\to \mathrm{Ty})} P\ \mathrm{refl}_a \to \prod_{(p\colon a=a)} P\ p.$$

We can add computable behaviour to this axiom by taking for $P \colon (a = a) \to \mathrm{Ty}$ and $p_0 \colon \mathrm{refl}_a$ that

$$\mathrm{ind}^{=}_{a}\ P\ p_0\ \mathrm{refl}_a \equiv p_0.$$

Adding this changes K from an axiom to a postulate.

With axiom K we can prove for every $p \colon a = a$ that $p = \mathrm{refl}\,a$. We namely see

$$\mathrm{ind}^{=}_{a}\ (\lambda p.\ p = \mathrm{refl}_a)\ \mathrm{refl}_{\mathrm{refl}_a} \colon \textstyle\prod_{(p\colon a=a)} (p = \mathrm{refl}_a).$$

Instead of adding axiom K we can add the following axiom to get univalent type theory, see [Uni13].

Axiom. (univalence) Suppose that A, $B\colon \mathrm{Ty}$, then the function

$$\mathrm{path\_to\_equi}\colon A = B \to A \simeq B$$

is an equivalence.

Adding both axiom K and the axiom of univalence gives a contradiction. To see this take

$$\begin{aligned} f &\colon 2 \to 2, \\ f\ 0_2 &\equiv 1_2, \\ f\ 1_2 &\equiv 0_2. \end{aligned}$$

We see that $f\,(f\,0_2) = 0_2$ and $f\,(f\,1_2) = 1_2$ therefore we have that f is its own quasi-inverse and that f is an equivalence. By univalence we have a corresponding path $p\colon 2 = 2$ such that $\mathrm{pr}_0\,(\mathrm{path\_to\_equi}\,p) = f$ but then by axiom K we have that $p = \mathrm{refl}_2$. We now have

$$f = \mathrm{pr}_0\,(\mathrm{path\_to\_equi}\,p) = \mathrm{pr}_0\,(\mathrm{path\_to\_equi}\,\mathrm{refl}_2) = \mathrm{id}_2$$

but then we get $0_2 = 1_2$. This is a contradiction because [Uni13, theorem 2.12.5] shows $0_2 \neq 1_2$.

The last axiom we consider is the law of the excluded middle.

Axiom. (excluded middle) For $A\colon \mathrm{Ty}$ such that $\prod_{(a,b\colon A)}(a = b)$ we have that $A + (A \to \mathbb{0})$.

Adding this axiom allows us to prove by contradiction.

Chapter 2

Inductive and Coinductive Types

2.0 Summary

In this chapter we will give a useful way to construct new types, called inductive types. These can be used to define for example natural numbers, finite lists, and finite trees. We will dualize this concept to get coinductive types. These are used to define for example streams, labelled transition systems, and infinite trees. In section 2.1 and section 2.2, we will first explain the concepts of algebras and coalgebras in a general category and for an arbitrary collection of endofunctors. These definitions are based on [Bas18, chapter 1]. We will give some examples in familiar categories and develop some intuition for them. In section 2.3 and section 2.4, we specialise to the category of types and to single functors. We consider polynomial functors based on [ACS15, definition 2] in section 2.5. Finally we explain how these concepts can be used as type constructors in section 2.6 and section 2.7.

2.1 General Algebras

The concept of an algebra is very broad; we start with the definition and some examples.

Definition. (algebra) Let C be a category, I a collection, and for every $i \in I$ let $F_i \colon C \to C$ be a functor.[0] An $(F_i)_{i \in I}$-algebra is a tuple $(X, (c_i)_{i \in I})$ consisting of an object X in C and for every $i \in I$ a morphism $c_i \colon F_i(X) \to X$ as shown below.

$$\begin{array}{c} X \\ {\scriptstyle c_i}\uparrow \\ F_i(X) \end{array}$$

[0] We define algebras and coalgebras for a collection of fuctors instead of for a single functor. This is because we need these more general definitions to show that $\sum$-types and $\prod$-types are algebras and coalgebras respectively.

Definition. (algebra morphism) A morphism between two $(F_i)_{i\in I}$-algebras $(X, (c_i)_{i\in I})$ and $(Y, (d_i)_{i\in I})$ is an $f\colon X \to Y$ such for every $i \in I$ the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{\quad f \quad} & Y \\ {\scriptstyle c_i}\uparrow & & \uparrow{\scriptstyle d_i} \\ F_i(X) & \xrightarrow[F_i(f)]{} & F_i(Y) \end{array}$$

Example. (magmas) We take the category of sets and the single functor F given by

$$\begin{aligned} F(X) &= X \times X, \\ F(f) &= f \times f. \end{aligned}$$

Then the F-algebras are pairs $(M, \circ)$ consisting of a set M and a binary operator $\circ\colon M \times M \to M$. We see that F-algebras are precisely magmas. An F-algebra morphism $f\colon (M, \circ) \to (N, *)$ is a function $f\colon M \to N$ such that the following diagram commutes.

$$\begin{array}{ccc} M & \xrightarrow{\quad f \quad} & N \\ {\scriptstyle \circ}\uparrow & & \uparrow{\scriptstyle *} \\ M \times M & \xrightarrow[f\times f]{} & N \times N \end{array}$$

This is equal to the requirement for a magma morphism that for every $m_1, m_2 \in M$ we have $f(m_1 \circ m_2) = f(m_1) * f(m_2)$. The name algebra comes in fact from this way to construct familiar algebraic structures such as magmas, groups, rings, and vector spaces. Note that for most of these it is necessary to place additional restrictions. For a group for example we require associativity, this is not immediately possible with our definition of an algebra.

Example. (natural numbers) Again in the category of sets take the functors F_0 and F_1 given by

$$\begin{aligned} F_0(X) &= \{\emptyset\}, & F_1(X) &= X, \\ F_0(f) &= \mathrm{id}_{\{\emptyset\}}; & F_1(f) &= f. \end{aligned}$$

Then $(\mathbb{N}, z, s)$ is an (F_0, F_1)-algebra where $\mathbb{N}$ is the set of natural numbers, $z\colon \{\emptyset\} \to \mathbb{N}$ is the constant function given by $z(\emptyset) = 0$, and $s\colon \mathbb{N} \to \mathbb{N}$ is the successor function given by $s(n) = n + 1$. The algebra $(\mathbb{N}, z, s)$ is special because it is the initial (F_0, F_1)-algebra, that is for every other (F_0, F_1)-algebra (X, c_0, c_1) we have an unique morphism $f\colon (\mathbb{N}, z, s) \to (X, c_0, c_1)$. To see this notice that a morphism $f\colon (\mathbb{N}, z, s) \to (X, c_0, c_1)$ is by definition a function $f\colon \mathbb{N} \to A$ such that the following two diagrams commute.

$$\begin{array}{ccc} \mathbb{N} & \xrightarrow{\quad f \quad} & X \\ {\scriptstyle z}\uparrow & & \uparrow{\scriptstyle c_0} \\ \{\emptyset\} & \xrightarrow[\mathrm{id}_{\{\emptyset\}}]{} & \{\emptyset\} \end{array} \qquad \begin{array}{ccc} \mathbb{N} & \xrightarrow{\quad f \quad} & X \\ {\scriptstyle s}\uparrow & & \uparrow{\scriptstyle c_1} \\ \mathbb{N} & \xrightarrow[f]{} & X \end{array}$$

This is precisely the requirement that for every $n \in \mathbb{N}$ we have

$$\begin{aligned} f(0) &= (f \circ z)(\emptyset) = (c_0 \circ \mathrm{id})(\emptyset) = c_0(\emptyset), \\ f(n+1) &= (f \circ s)(n) = (c_1 \circ f)(n) = c_1(f(n)). \end{aligned}$$

The fact that for every constant $c_0(\emptyset)\colon X$ and function $c_1\colon X \to X$ there is a unique $f\colon \mathbb{N} \to X$ satisfying $f(0) = c_0(\emptyset)$ and $f(n+1) = c_1(f(n))$ is the defining property of the natural numbers, known as the induction principle.

Definition. (initial algebra) An $(F_i)_{i\in I}$-algebra $(A, (e_i)_{i\in I})$ is called initial if we have for every $(F_i)_{i\in I}$-algebra $(X, (c_i)_{i\in I})$ an unique morphism $f\colon (A, (e_i)_{i\in I}) \to (X, (c_i)_{i\in I})$.

We justify our use of the phrase '<u>the</u> initial algebra' by the following proposition.

Proposition 2.1.0. An initial $(F_i)_{i\in I}$-algebra, if it exists, is unique up to unique isomorpism.

Proof. We see this by assuming that $(A, (e_i)_{i\in I})$ and $(A', (e'_i)_{i\in I})$ are both initial $(F_i)_{i\in I}$-algebras. Then we have two unique morphisms

$$f\colon (A,\ (e_i)_{i\in I}) \to (A',\ (e'_i)_{i\in I}),$$
$$g\colon (A',\ (e'_i)_{i\in I}) \to (A,\ (e_i)_{i\in I}).$$

This gives use the morphisms

$$g \circ f\colon (A,\ (e_i)_{i\in I}) \to (A,\ (e_i)_{i\in I}),$$
$$\mathrm{id}_A\colon (A,\ (e_i)_{i\in I}) \to (A,\ (e_i)_{i\in I}).$$

Because $(A, (e_i)_{i\in I})$ is initial we have that $g \circ f = \mathrm{id}_A$ and by symmetry we have that $f \circ g = \mathrm{id}_{A'}$. This shows that f is an isomorphism. □

2.2 General Coalgebras

The concept of a coalgebra is dual to that of an algebra and allows us to define coinductive types.

Definition. (coalgebra) Let C be a category, I a collection, and for every $i \in I$ let $F_i\colon C \to C$ be a functor. An $(F_i)_{i\in I}$-coalgebra is a tuple $(X, (o_i)_{i\in I})$ consisting of an object X in C and for every $i \in I$ a morphism $o_i\colon F_i(X) \to X$ as shown below.

$$\begin{array}{c} X \\ {\scriptstyle o_i}\big\downarrow \\ F_i(X) \end{array}$$

Definition. (coalgebra morphism) A morphism between two $(F_i)_{i\in I}$-coalgebras $(X, (o_i)_{i\in I})$ and $(Y, (p_i)_{i\in I})$ is an $f\colon X \to Y$ such for every $i \in I$ the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle o_i}\downarrow & & \downarrow{\scriptstyle p_i} \\ F_i(X) & \xrightarrow[F_i(f)]{} & F_i(Y) \end{array}$$

Example. Again take the category of sets and for a set S take the functors F_0 and F_1 given by

$$\begin{aligned} F_0(X) &= S, & F_1(X) &= X, \\ F_0(f) &= \mathrm{id}_S; & F_1(f) &= f. \end{aligned}$$

Then (S^ω, h, t) is a (F_0, F_1)-coalgebra where $S^\omega = \{(s_0, s_1, s_2, \dots)\colon s_0, s_1, s_2, \dots \in S\}$ is the set of infinite sequences on the set S, where h is the head function defined by

$$\begin{aligned} &h\colon S^\omega \to S, \\ &h(s_0,\ s_1,\ s_2,\ \dots) = s_0; \end{aligned}$$

and where t is the tail function defined by

$$\begin{aligned} &t\colon S^\omega \to S^\omega, \\ &t(s_0,\ s_1,\ s_2,\ \dots) = (s_1,\ s_2,\ s_3,\ \dots). \end{aligned}$$

In fact (S^ω, h, t) is special because it is the final F-coalgebra, that is for every F-coalgebra (X, o_0, o_1) we have an unique morphism $f\colon (X, o_0, o_1) \to (S^\omega, h, t)$. To see that this is true notice that a morphism $f\colon (X, o_0, o_1) \to (S^\omega, h, t)$ is by definition a function $f\colon X \to S^\omega$ such that the following two diagrams commute.

$$\begin{array}{ccc} X & \xrightarrow{f} & S^\omega \\ {\scriptstyle o_0}\downarrow & & \downarrow{\scriptstyle h} \\ S & \xrightarrow[\mathrm{id}_S]{} & S \end{array} \qquad\qquad \begin{array}{ccc} X & \xrightarrow{f} & S^\omega \\ {\scriptstyle o_1}\downarrow & & \downarrow{\scriptstyle t} \\ X & \xrightarrow[f]{} & S^\omega \end{array}$$

This is precisely the requirement that for every $x \in X$ we have

$$\begin{aligned} h(f(x)) &= (h \circ f)(x) = (\mathrm{id} \circ o_0)(x) = o_0(x), \\ t(f(x)) &= (t \circ f)(x) = (f \circ o_1)(x) = f(o_1(x)). \end{aligned}$$

The only f that satisfies this requirement is given by

$$f(x) = (o_0(x),\ o_0(o_1(x)),\ o_0(o_1(o_1(x))), \dots).$$

Definition. (final coalgebra) A $(F_i)_{i\in I}$-coalgebra $(C, (g_i)_{i\in I})$ is called final if we have for every $(F_i)_{i\in I}$-coalgebra $(X, (o_i)_{i\in I})$ an unique morphism $f\colon (X, (o_i)_{i\in I}) \to (C, (g_i)_{i\in I})$.

Again we justify using '<u>the</u> final coalgebra' by the following proposition.

Proposition 2.2.0. A final $(F_i)_{i\in I}$-coalgebra, if it exists, is unique up to unique isomorpism.

Proof. We see this by assuming that $(C,\ (g_i)_{i\in I})$ and $(C',\ (g'_i)_{i\in I})$ are both final $(F_i)_{i\in I}$-coalgebras. Then we have two unique morphisms

$$f\colon (C',\ (g'_i)_{i\in I}) \to (C,\ (g_i)_{i\in I}),$$
$$g\colon (C,\ (g_i)_{i\in I}) \to (C',\ (g'_i)_{i\in I}).$$

This gives use the morphisms

$$f\circ g\colon (C,\ (g_i)_{i\in I}) \to (C,\ (g_i)_{i\in I}),$$
$$\mathrm{id}_C\colon (C,\ (g_i)_{i\in I}) \to (C,\ (g_i)_{i\in I}).$$

Because $(C,\ (g_i)_{i\in I})$ is initial we have that $f\circ g = \mathrm{id}_C$ and by symmetry we have that $g\circ f = \mathrm{id}_{C'}$. This shows that f is an isomorphism. □

2.3 Inductive Types

Now we specialise our definitions to the the category of types. We will only consider algebras and coalgebras for a single functor F, in Martin-Löf type theory this does not limit expressiveness. To see this suppose I is a type and we have for every $i\colon I$ a functor $F_i\colon \mathrm{Ty}\to\mathrm{Ty}$. Then we consider the functors $\sum_{(i:I)}F_i\colon \mathrm{Ty}\to\mathrm{Ty}$ and $\prod_{(i:I)}F_i\colon \mathrm{Ty}\to\mathrm{Ty}$ which are given by

$$(\textstyle\sum_{(i:I)}F_i)\ X \equiv \sum_{(i:I)}F_i\ X, \qquad (\textstyle\prod_{(i:I)}F_i)\ X \equiv \prod_{(i:I)}F_i\ X,$$
$$(\textstyle\sum_{(i:I)}F_i)\ f \equiv \sum_{(i:I)}F_i\ f\,; \qquad (\textstyle\prod_{(i:I)}F_i)\ f \equiv \prod_{(i:I)}F_i\ f\,.$$

Notice that

$$((\textstyle\sum_{(i:I)}F_i\ X)\to X) \simeq \prod_{(i:I)}(F_i\ X\to X); \qquad (X\to(\prod_{(i:I)}F_i\ X)) \simeq \prod_{(i:I)}(X\to F_i\ X);$$

therefore giving an $(\sum_{(i:I)}F_i)$-algebra is equivalent to giving an $(F_i)_{(i:I)}$-algebra while giving a $(\prod_{(i:I)}F_i)$-coalgebra is equivalent to giving a $(F_i)_{(i:I)}$-coalgebra.

Definition. (algebra) For an endofunctor $F\colon \mathrm{Ty}\to\mathrm{Ty}$ we take the type of algebras as

$$\mathrm{Alg}_F \equiv \textstyle\sum_{(\mathrm{ty}:\mathrm{Ty})}F\ \mathrm{ty}\to\mathrm{ty}$$

where for $X\colon \mathrm{Alg}_F$ we write $\mathrm{ty}_X \equiv \mathrm{pr}_0\,X$ and $\mathrm{con}_X \equiv \mathrm{pr}_1\,X$, as shown below.

$$\begin{array}{c} \mathrm{ty}_X \\ {\scriptstyle \mathrm{con}_X}\uparrow \\ F(\mathrm{ty}_X) \end{array}$$

Definition. (algebra morphism) For X, $Y : \mathrm{Alg}_F$ we take the type of algebra morphisms as

$$\mathrm{AlgMor}_F\ X\ Y \equiv \textstyle\sum_{(\mathrm{fun} : \mathrm{ty}_X \to \mathrm{ty}_Y)} (\mathrm{fun} \circ \mathrm{con}_X = \mathrm{con}_Y \circ F\ \mathrm{fun})$$

where for $f : \mathrm{AlgMor}_F\ X\ Y$ we write $\mathrm{fun}_f \equiv \mathrm{pr}_0\ f$ and $\mathrm{com}_f \equiv \mathrm{pr}_1\ f$, as shown below.

$$\begin{array}{ccc} \mathrm{ty}_X & \xrightarrow{\mathrm{fun}_f} & \mathrm{ty}_Y \\ {\scriptstyle \mathrm{con}_X}\uparrow & \mathrm{com}_f & \uparrow{\scriptstyle \mathrm{con}_Y} \\ F(\mathrm{ty}_X) & \xrightarrow[F\,\mathrm{fun}_f]{} & F(\mathrm{ty}_Y) \end{array}$$

Example. (natural numbers) Instead of taking two functors like in section 2.1 we now define a single functor $F : \mathrm{Ty} \to \mathrm{Ty}$ given by

$$\begin{aligned} F\ X &\equiv \mathbb{1}\ + X, \\ F\ f &\equiv \mathrm{id}_{\mathbb{1}} + f\,. \end{aligned}$$

We take $A : \mathrm{Alg}_F$ by $\mathrm{ty}_A \equiv \mathbb{N}$ and

$$\begin{aligned} &\mathrm{con}_A : \mathbb{1} + \mathbb{N} \to \mathbb{N}, \\ &\mathrm{con}_A\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \equiv 0_{\mathbb{N}}, \\ &\mathrm{con}_A\ (\mathrm{in}_1\ n) \equiv \mathrm{suc}\ n. \end{aligned}$$

To show that A is the initial algebra we will define a dependent function $\mathrm{iter} : \prod_{(X : \mathrm{Alg})} \mathrm{AlgMor}\ A\ X$. We do this by taking for $X : \mathrm{Alg}$ that

$$\begin{aligned} &\mathrm{fun}_{(\mathrm{iter}\ X)} : \mathbb{N} \to \mathrm{ty}_X, \\ &\mathrm{fun}_{(\mathrm{iter}\ X)}\ 0_{\mathbb{N}} \equiv \mathrm{con}_X\ (\mathrm{in}_0\ 0_{\mathbb{1}}), \\ &\mathrm{fun}_{(\mathrm{iter}\ X)}\ (\mathrm{suc}\ n) \equiv \mathrm{con}_X\ (\mathrm{in}_1\ (f\ n)). \end{aligned}$$

We get $\mathrm{com}_{(\mathrm{iter}\ X)}$ as follows. We see

$$\begin{aligned} (\mathrm{fun}_{(\mathrm{iter}\ X)} \circ \mathrm{con}_A)\ (\mathrm{in}_0\ 0_{\mathbb{1}}) &\equiv \mathrm{fun}_{(\mathrm{iter}\ X)}\ (\mathrm{con}_A\ (\mathrm{in}_0\ 0_{\mathbb{1}})) \\ &\equiv \mathrm{fun}_{(\mathrm{iter}\ X)}\ 0_{\mathbb{N}} \\ &\equiv \mathrm{con}_X\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \\ &\equiv \mathrm{con}_X\ ((\mathrm{id}_{\mathbb{1}} + \mathrm{fun}_{(\mathrm{iter}\ X)})\ (\mathrm{in}_0\ 0_{\mathbb{1}})) \\ &\equiv (\mathrm{con}_X \circ (\mathrm{id}_{\mathbb{1}} + \mathrm{fun}_{(\mathrm{iter}\ X)}))\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \end{aligned}$$

and for $n : \mathbb{N}$ that

$$\begin{aligned} (\mathrm{fun}_{(\mathrm{iter}\ X)} \circ \mathrm{con}_A)\ (\mathrm{in}_1\ n) &\equiv \mathrm{fun}_{(\mathrm{iter}\ X)}\ (\mathrm{con}_A\ (\mathrm{in}_1\ n)) \\ &\equiv \mathrm{fun}_{(\mathrm{iter}\ X)}\ (\mathrm{suc}\ n) \\ &\equiv \mathrm{con}_X\ (\mathrm{in}_1\ (\mathrm{fun}_{(\mathrm{iter}\ X)}\ n)) \\ &\equiv \mathrm{con}_X\ ((\mathrm{id}_{\mathbb{1}} + \mathrm{fun}_{(\mathrm{iter}\ X)})\ (\mathrm{in}_1\ n)) \\ &\equiv (\mathrm{con}_X \circ (\mathrm{id}_{\mathbb{1}} + \mathrm{fun}_A))\ (\mathrm{in}_1\ n) \end{aligned}$$

therefore by induction on $\mathbb{1}+\mathbb{N}$ we get $\prod_{(n' : \mathbb{1}+\mathbb{N})}((\mathrm{fun}_{(\mathrm{iter}\ X)} \circ \mathrm{con}_A)\ n' = (\mathrm{con}_X \circ (\mathrm{id}_{\mathbb{1}} + \mathrm{fun}_{(\mathrm{iter}\ X)}))\ n')$. Now we apply function extensionality to take $\mathrm{com}_{(\mathrm{iter}\ X)} : \mathrm{fun}_{(\mathrm{iter}\ X)} \circ \mathrm{con}_A = \mathrm{con}_X \circ (\mathrm{id}_{\mathbb{1}} + \mathrm{fun}_{(\mathrm{iter}\ X)})$.

Definition. (initial algebra) Let $F\colon \mathrm{Ty} \to \mathrm{Ty}$ be an endofunctor. If we have $A\colon \mathrm{Alg}_F$ and a dependent function $\mathrm{iter}\colon \prod_{(X\colon \mathrm{Alg}_F)} \mathrm{AlgMor}_F\, A\, X$, then we define

$$\mathrm{IsIniAlg}_F\ A\ \mathrm{iter} \equiv \prod_{(X\colon \mathrm{Alg}_F)} \prod_{(f\colon \mathrm{AlgMor}_F\, A\, X)} (f = \mathrm{iter}\ X),$$

and we define

$$\mathrm{IniAlg}_F \equiv \sum_{(A\colon \mathrm{Alg}_F)} \sum_{(\mathrm{iter}\colon \prod_{(X\colon \mathrm{Alg}_F)} \mathrm{AlgMor}_F\, A\, X)} \mathrm{IsIniAlg}_F\ A\ \mathrm{iter}.$$

Example. (natural numbers) We continue the last example and prove that the coalgebra A given there is the initial algebra. Assume we have $X\colon \mathrm{Alg}_F$ and $f\colon \mathrm{AlgMor}_F\, A\, X$. We see

$$\begin{aligned}
\mathrm{fun}_f\ 0_{\mathbb{N}} &\equiv \mathrm{fun}_f\ (\mathrm{con}_A\ (\mathrm{in}_0\ 0_{\mathbb{1}})) \\
&\equiv (\mathrm{fun}_f \circ \mathrm{con}_A)\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \\
&= (\mathrm{con}_X \circ (\mathrm{id} + \mathrm{fun}_f))\ (\mathrm{in}_0\ 0_{\mathbb{N}}) && \text{by apply}^{=}\ \mathrm{com}_f\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \\
&\equiv \mathrm{con}_X\ ((\mathrm{id} + \mathrm{fun}_f)\ (\mathrm{in}_0\ 0_{\mathbb{1}})) \\
&\equiv \mathrm{con}_X\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \\
&\equiv \mathrm{con}_X\ ((\mathrm{id} + \mathrm{fun}_{(\mathrm{iter}\, A)})\ (\mathrm{in}_0\ 0_{\mathbb{1}})) \\
&\equiv (\mathrm{con}_X \circ (\mathrm{id} + \mathrm{fun}_{(\mathrm{iter}\, A)}))\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \\
&= (\mathrm{fun}_{(\mathrm{iter}\, X)} \circ \mathrm{con}_A)\ (\mathrm{in}_0\ 0_{\mathbb{1}}) && \text{by apply}^{=}\ \mathrm{com}_{(\mathrm{iter}\, X)}\ (\mathrm{in}_0\ 0_{\mathbb{1}}) \\
&\equiv \mathrm{fun}_{(\mathrm{iter}\, X)}\ (\mathrm{con}_A\ (\mathrm{in}_0\ 0_{\mathbb{1}})) \\
&\equiv \mathrm{fun}_{(\mathrm{iter}\, X)}\ 0_{\mathbb{N}},
\end{aligned}$$

and for $n\colon \mathbb{N}$ assuming an $h\colon \mathrm{fun}_f\, n = \mathrm{fun}_{(\mathrm{iter}\, X)}\, n$ that

$$\begin{aligned}
\mathrm{fun}_f\ (\mathrm{suc}\ n) &\equiv \mathrm{fun}_f\ (\mathrm{con}_A\ (\mathrm{in}_1\ n)) \\
&\equiv (\mathrm{fun}_f \circ \mathrm{con}_A)\ (\mathrm{in}_1\ n) \\
&= (\mathrm{con}_X \circ (\mathrm{id} + \mathrm{fun}_f))\ (\mathrm{in}_1\ n) && \text{by apply}^{=}\ \mathrm{com}_f\ (\mathrm{in}_1\ n) \\
&\equiv \mathrm{con}_X\ ((\mathrm{id} + \mathrm{fun}_f)\ (\mathrm{in}_1\ n)) \\
&\equiv \mathrm{con}_X\ (\mathrm{in}_1\ (\mathrm{fun}_f\ n)) \\
&= \mathrm{con}_X\ (\mathrm{in}_1\ (\mathrm{fun}_{(\mathrm{iter}\, X)}\ n)) && \text{by } \mathrm{ap}_{\lambda x.\, \mathrm{con}_X\, (\mathrm{in}_1\, x)}\ h \\
&\equiv \mathrm{con}_X\ ((\mathrm{id} + \mathrm{fun}_{(\mathrm{iter}\, X)})\ (\mathrm{in}_1\ n)) \\
&\equiv (\mathrm{con}_X \circ (\mathrm{id} + \mathrm{fun}_{(\mathrm{iter}\, X)}))\ (\mathrm{in}_1\ n) \\
&= (\mathrm{fun}_{(\mathrm{iter}\, X)} \circ \mathrm{con}_A)\ (\mathrm{in}_1\ n) && \text{by apply}^{=}\ \mathrm{com}_{(\mathrm{iter}\, X)}\ (\mathrm{in}_1\ n) \\
&\equiv \mathrm{fun}_{(\mathrm{iter}\, X)}\ (\mathrm{con}_A\ (\mathrm{in}_1\ n)) \\
&\equiv \mathrm{fun}_{(\mathrm{iter}\, X)}\ (\mathrm{suc}\ n).
\end{aligned}$$

Now using induction on $\mathbb{N}$ gives $\prod_{(n\colon \mathbb{N})} (\mathrm{fun}_f\, n = \mathrm{fun}_{(\mathrm{iter}\, X)}\, n)$ and we can apply function extensionality to get $p\colon \mathrm{fun}_f = \mathrm{fun}_{(\mathrm{iter}\, X)}$. By [Uni13, theorem 5.4.5] we also have $q\colon \mathrm{tra}\, p\, \mathrm{com}_f = \mathrm{com}_{(\mathrm{iter}\, X)}$. We can combine these to get $\mathrm{pair}^{=}\ (p,\, q)\colon f = \mathrm{iter}\, X$. We conclude that A is the initial F-algebra.

2.4 Coinductive Types

The definition of a coalgebra has a similar translation into type theory.

Definition. (coalgebra) For an endofunctor $F : \mathrm{Ty} \to \mathrm{Ty}$ we take the type of coalgebras as

$$\mathrm{Coalg}_F \equiv \textstyle\sum_{(\mathrm{ty} : \mathrm{Ty})} \mathrm{ty} \to F\ \mathrm{ty}$$

where for $X : \mathrm{Coalg}_F$ we denote $\mathrm{ty}_X \equiv \mathrm{pr}_0\, X$ and $\mathrm{obs}_X \equiv \mathrm{pr}_1\, X$, as shown below.

$$\begin{array}{c} \mathrm{ty}_X \\ {\scriptstyle \mathrm{obs}_X}\big\downarrow \\ F(\mathrm{ty}_X) \end{array}$$

Definition. (coalgebra morphism) For $X, Y : \mathrm{Alg}_F$ we take the type of algebra morphisms as

$$\mathrm{CoalgMor}_F\ X\ Y \equiv \textstyle\sum_{(\mathrm{fun} : \mathrm{ty}_X \to \mathrm{ty}_Y)} \mathrm{obs}_Y \circ \mathrm{fun} = F\ \mathrm{fun} \circ \mathrm{obs}_X$$

where for $f : \mathrm{AlgMor}_F\, X\, Y$ we denote $\mathrm{fun}_f \equiv \mathrm{pr}_0\, f$ and $\mathrm{com}_f \equiv \mathrm{pr}_1\, f$, as shown below.

$$\begin{array}{ccc} \mathrm{ty}_X & \xrightarrow{\mathrm{fun}_f} & \mathrm{ty}_Y \\ {\scriptstyle \mathrm{obs}_X}\big\downarrow & \mathrm{com}_f & \big\downarrow{\scriptstyle \mathrm{obs}_Y} \\ F(\mathrm{ty}_X) & \xrightarrow[F\,\mathrm{fun}_f]{} & F(\mathrm{ty}_Y) \end{array}$$

Example. Consider the identity functor $\mathrm{Id} : \mathrm{Ty} \to \mathrm{Ty}$ given by

$$\begin{aligned} \mathrm{Id}\ X &\equiv X, \\ \mathrm{Id}\ f &\equiv f\,. \end{aligned}$$

We take $C : \mathrm{Coalg}_{\mathrm{Id}}$ given by $\mathrm{ty}_C \equiv \mathbb{1}$ and $\mathrm{obs}_C \equiv \mathrm{id}_{\mathbb{1}}$. We can define a dependent function $\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_{\mathrm{Id}})} \mathrm{CoalgMor}_{\mathrm{Id}}\, X\, C$ in the following way. For $X : \mathrm{Coalg}_{\mathrm{Id}}$ take

$$\begin{aligned} &\mathrm{fun}_{(\mathrm{coiter}\, X)} : \mathrm{ty}_X \to \mathbb{1}, \\ &\mathrm{fun}_{(\mathrm{coiter}\, X)}\ x \equiv 0_{\mathbb{1}}. \end{aligned}$$

We see for every $x : \mathrm{ty}_X$ that

$$(\mathrm{obs}_C \circ \mathrm{fun}_{(\mathrm{coiter}\, X)})\ x \equiv 0_{\mathbb{1}} \equiv (\mathrm{fun}_{(\mathrm{coiter}\, X)} \circ \mathrm{obs}_X)\ x$$

which gives $\prod_{(x : \mathrm{ty}_X)}((\mathrm{obs}_C \circ \mathrm{fun}_{(\mathrm{coiter}\, X)})\, x = (\mathrm{fun}_{(\mathrm{coiter}\, X)} \circ \mathrm{obs}_X)\, x)$. Now by applying function extensionality we take $\mathrm{com}_{(\mathrm{coiter}\, X)} : \mathrm{obs}_C \circ \mathrm{fun}_{(\mathrm{coiter}\, X)} = \mathrm{fun}_{(\mathrm{coiter}\, X)} \circ \mathrm{obs}_X$.

For more examples and general exercises in coinduction see [Rut19].

Definition. (final coalgebra) Let $F : \mathrm{Ty} \to \mathrm{Ty}$ be an endofunctor. If we have $C : \mathrm{Coalg}_F$ and a dependent function coiter : $\prod_{(X : \mathrm{Coalg}_F)} \mathrm{CoalgMor}_F\, X\, C$, then we define

$$\mathrm{IsFinCoalg}_F\ C\ \mathrm{coiter} \equiv \prod_{(X : \mathrm{Coalg}_F)} \prod_{(f : \mathrm{CoalgMor}_F\, X\, C)} f = \mathrm{coiter}\ X,$$

and we take

$$\mathrm{FinCoalg}_F \equiv \sum_{(C : \mathrm{Coalg}_F)} \sum_{(\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_F)} \mathrm{CoalgMor}_F\, X\, C)} \mathrm{IsFinCoalg}_F\ C\ \mathrm{coiter}.$$

Example. We prove that $C : \mathrm{Coalg}_F$ given in the last example the final coalgebra. Let $X : \mathrm{Coalg}_F$ and $f : \mathrm{CoalgMor}_F\, X\, C$, we see for every $x : X$ that

$$\mathrm{fun}_f\ x = \mathrm{fun}_{(\mathrm{coiter}\, X)}\ x$$

because $\mathbb{1}$ is a proposition as defined in [Uni13, chapter 3]. By function extensionality we can take $p : \mathrm{fun}_f = \mathrm{fun}_{(\mathrm{coiter}\, X)}$. In the same way we have for every $x : X$ that

$$\mathrm{apply}^{=}\ (\mathrm{tra}\ p\ \mathrm{com}_f)\ x = \mathrm{apply}^{=}\ \mathrm{com}_{(\mathrm{coiter}\, X)}\ x$$

because $\mathbb{1}$ is a set as defined in [Uni13, chapter 3]. If we apply function extensionality we get $\mathrm{apply}^{=}\,(\mathrm{tra}\, p\, \mathrm{com}_f) = \mathrm{apply}^{=}\, \mathrm{com}_{(\mathrm{coiter}\, X)}$. Now because the function $\mathrm{apply}^{=}$ is an equivalence we can take $q : \mathrm{tra}\, p\, \mathrm{com}_f = \mathrm{com}_{(\mathrm{coiter}\, X)}$. We combine p and q to see $\mathrm{pair}^{=}\,(p,\, q) : f = \mathrm{coiter}\, X$. We conclude that C is indeed the final F-coalgebra.

2.5 Polynomial Functors

We now specialise to functors of a particular form. This definition is based on [GK13]. Polynomial functors are also known as container functors in type theoretic context, see [AAG03, AAG05].

Definition. (polynomial functor) A polynomial functor is a functor P of the form

$$P\ X \equiv \sum_{(a : A)} (B\ a \to X),$$
$$P\ f \equiv \lambda(a,\, d).\ (a,\ f \circ d);$$

for $A : \mathrm{Ty}$ and $B : A \to \mathrm{Ty}$. We can P the polynomial functor for A and B.

Notice that we do not have a requirement that A is finite or for $a : A$ that $B\, a$ is finite. Indeed we have not even defined what it means for a type to be finite. The name polynomial comes from the perspective of seeing types as numbers, we will show this below. There are three useful ways to view a type. We can view a type as a set which elements correspond to its terms. Alternatively we can view a type as a proposition which we regard as true if and only if the type has terms. Lastly we can view a type as a, potentially infinite, number corresponding to the number of terms. Note that we only use these perspectives for intuition. For A, $B : \mathrm{Ty}$ we get:

Type	Set	Proposition	Number
$A + B$	disjoint union	or	sum
$A \times B$	Cartesian product	and	product
$A \to B$	functions	implies	power

For $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ we get:

Type	Set	Proposition	Number
$\sum_{(a\colon A)} B\,a$	disjoint union	exists	sum
$\prod_{(a\colon A)} B\,a$	Cartesian product	forall	product

2.6 W-types

In section 1.3 we have postulated for types $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ that we can form new types $\sum_{(a\colon A)} B\,a$ and $\prod_{(a\colon A)} B\,a$. In this section we discuss W-types as a possible additional postulate. If P is the polynomial functor for A and B then we can postulate the existence of $\mathsf{W}_{(a\colon A)} B\,a$ which is the initial algebra for P. In the next section we will similarly see how $\mathsf{M}_{(a\colon A)} B\,a$ can be postulated as the final coalgebra for P.

This is a categorical definition of W-types similar to [MP00], for a more conventional and practical approach see for instance [Uni13, section 5.3].

Postulate. (W-types) For $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ take P as the polynomial functor given by

$$P\ X \equiv \textstyle\sum_{(a\colon A)} B\ a \to X,$$
$$P\ f \equiv \lambda(a,\ d).\ (a,\ f \circ d).$$

We postulate the existence of $\mathsf{W}_{(a\colon A)} B\,a\colon \mathrm{Alg}_P$ with

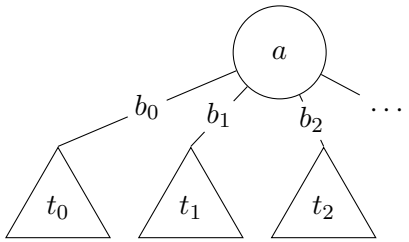
$$\mathrm{iter}\colon \textstyle\prod_{(X\colon \mathrm{Alg}_P)} \mathrm{AlgMor}_P\ (\mathsf{W}_{(a\colon A)} B\ a)\ X,$$
$$\mathrm{isIni}\colon \mathrm{IsIniAlg}_P\ (\mathsf{W}_{(a\colon A)} B\ a)\ \mathrm{iter};$$

such that for $X\colon \mathrm{Alg}$ we have

$$\mathrm{fun}_{(\mathrm{iter}\,X)} \circ \mathrm{con}_{\mathsf{W}_{(a\colon A)} B\,a} \equiv \mathrm{con}_X \circ F\ \mathrm{fun}_{(\mathrm{iter}\,X)}$$

and $\mathrm{com}_{(\mathrm{iter}\,X)} \equiv \mathrm{refl}$.

We will develop some intuition for the initial P-algebra. Consider trees consisting of a root node with a label $a\colon A$ and for every $b\colon B\,a$ a subtree made in the same way.

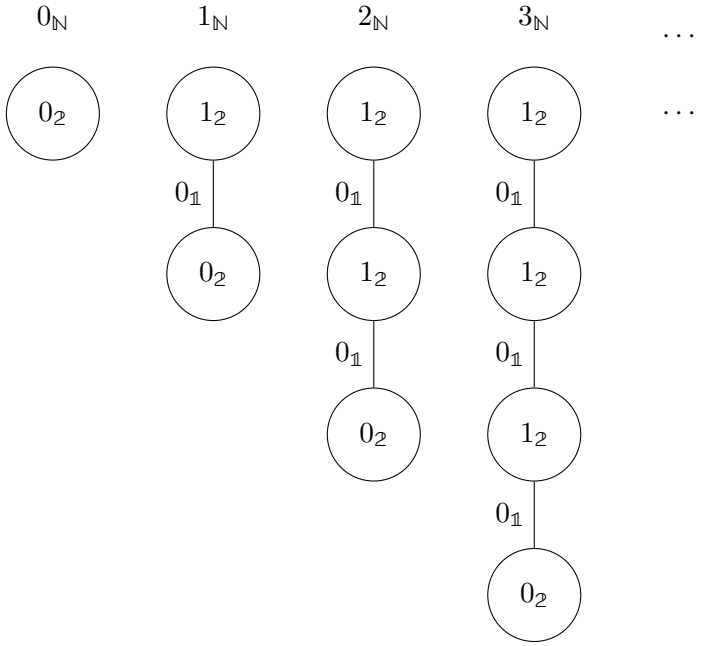


For the initial algebra $\mathsf{W}_{(a\colon A)} B\,a$ we can consider $\mathrm{ty}_{(\mathsf{W}_{(a\colon A)} B\,a)}$ as the type of finite-depth trees made in this way and $\mathrm{con}_{(\mathsf{W}_{(a\colon A)} B\,a)}$ as the function that constructs a new tree using a label $a\colon A$ for the root and for every $b\colon B\,a$ a finite-depth tree as subtree. The use of the letter W comes from 'well founded' which means finite-depth in this case.

Example. (natural numbers) Take $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ by

$$A \equiv \mathbb{2}; \qquad\qquad B\,0_{\mathbb{2}} \equiv \mathbb{0}, \quad B\,1_{\mathbb{2}} \equiv \mathbb{1}.$$

Then we can consider $\mathsf{W}_{(a\colon A)} B\,a$ as the natural numbers in the following way.

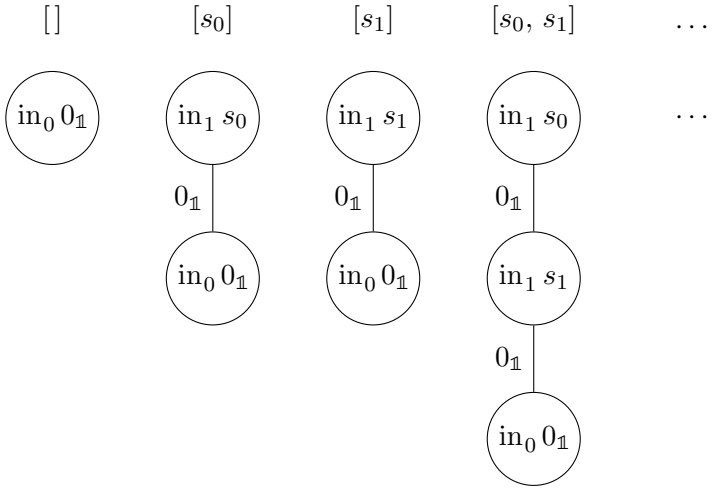


Therefore if we postulate W-types it is no longer necessary to postulate the natural numbers.

Example. (finite lists) For an $S\colon \mathrm{Ty}$ take $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ by

$$A \equiv \mathbb{1} + S; \qquad\qquad B\,(\mathrm{in}_0\,0_{\mathbb{1}}) \equiv \mathbb{0}, \quad B\,(\mathrm{in}_1\ s) \equiv \mathbb{1}.$$

Then we can consider $\mathsf{W}_{(a\colon A)} B\,a$ as the type of finite lists on S in the following way.



2.7 M-types

Let P be the polynomial functor for $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$. Then we can add an additional postulate that takes $\mathsf{M}_{(a\colon A)} B\, a$ as the final coalgebra for P.

Postulate. (M-types) For $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ take P as the polynomial functor given by

$$P\ X \equiv \textstyle\sum_{(a\colon A)} B\ a \to X,$$
$$P\ f \equiv \lambda(a,\ d).\ (a,\ f \circ d).$$

We postulate the existence of $\mathsf{M}_{(a\colon A)} B\, a\colon \mathrm{Coalg}_P$ with

$$\mathrm{coiter}\colon \textstyle\prod_{(X\colon \mathrm{Coalg})} \mathrm{CoalgMor}_P\ X\ (\mathsf{M}_{(a\colon A)} B\ a),$$
$$\mathrm{isFin}\colon \mathrm{IsFinCoalg}_P\ (\mathsf{M}_{(a\colon A)} B\ a)\ \mathrm{coiter};$$

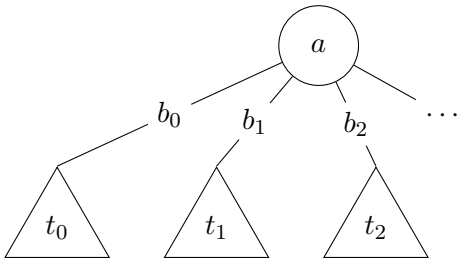
such that for $X\colon \mathrm{Alg}$ we have

$$\mathrm{obs}_X \circ \mathrm{fun}_{(\mathrm{coiter}\, X)} \equiv F\ \mathrm{fun}_{(\mathrm{coiter}\, X)} \circ \mathrm{obs}_{\mathsf{M}_{(a\colon A)} B\, a}$$

and $\mathrm{com}_{(\mathrm{coiter}\, X)} \equiv \mathrm{refl}$.

If we postulate natural numbers then we can prove that there exists a final coalgebra for every polynomial functor, see [AAG05,BM07,ACS15]. The advantage of postulating such a final coalgebra directly is that we can add definitional equalities instead of only propositional equalities. These make M-types easier to work with.

For $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ we again consider trees of the following form. The root of the tree has a label $a\colon A$ and for every $b\colon B\, a$ a subtree made of the same form.

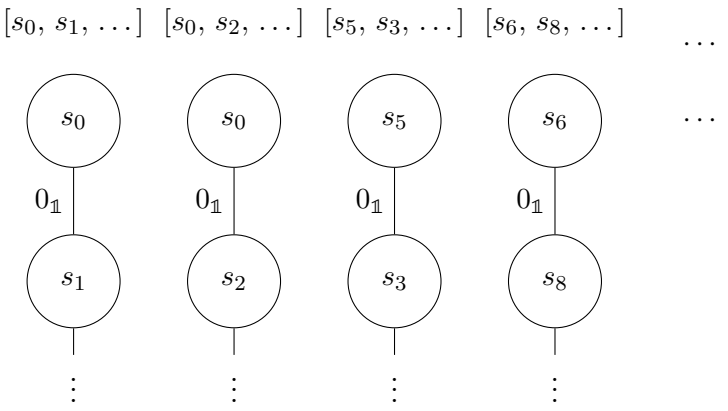


For the final coalgebra $\mathsf{M}_{(a\colon A)} B\, a$ we can consider $\mathrm{ty}_{(\mathsf{M}_{(a\colon A)} B\, a)}$ as the type consisting of all, possibly infinite, trees made in this way. Then $\mathrm{obs}_{(\mathsf{M}_{(a\colon A)} B\, a)}$ observes the tree returning the root label $a\colon A$ and for every $b\colon B\, a$ the, possibly infinite, subtree.

Suppose that we have for every $a\colon A$ that $B\, a$ has terms. Then there are no well-founded trees. This means that $\mathsf{W}_{(a\colon A)} B\, a$ does not have any term, the type $\mathsf{M}_{(a\colon A)} B\, a$ still does because it also contains all infinite trees.

Another edge case to consider is the case where A does not have terms. Then there are no trees and we have that both $\mathsf{W}_{(a\colon A)} B\, a$ and $\mathsf{M}_{(a\colon A)} B\, a$ have no terms.

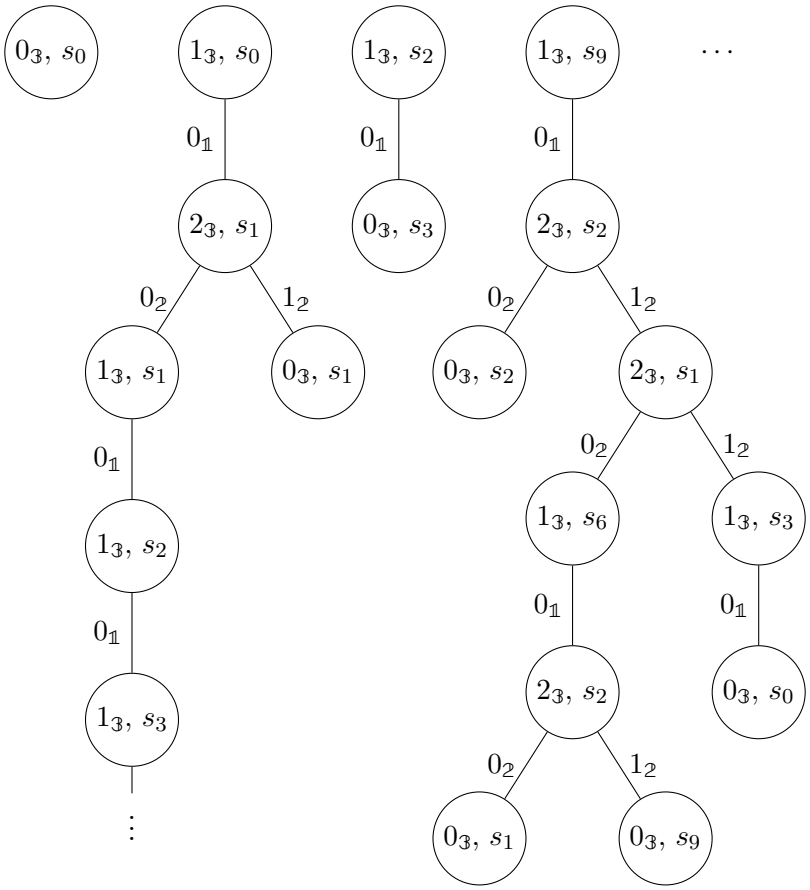
Example. (streams) For $S\colon \mathrm{Ty}$ we consider $\mathsf{M}_{(s\colon S)}\mathbb{1}$ as the type of streams on S as follows.



Example. (possibly infinite binary trees) For $S\colon \mathrm{Ty}$ we define $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$ by

$$A \equiv \mathbb{3} \times S; \qquad \begin{aligned} B\,(0_{\mathbb{3}},\, s) &\equiv \mathbb{0},\\ B\,(1_{\mathbb{3}},\, s) &\equiv \mathbb{1},\\ B\,(2_{\mathbb{3}},\, s) &\equiv \mathbb{2}. \end{aligned}$$

Then we can see $\mathsf{M}_{(a\colon A)} B\,a$ as the type of all, possibly infinite, binary trees.



Chapter 3

Univalence implies Function Extensionality

3.0 Summary

In this chapter we will show that $\sum$-types are initial algebras in section 3.1 and that $\prod$-types are final coalgebras in section 3.2. In section 3.2 we also define bisimulation for $\prod$-types. If we see $\prod$-types as final coalgebras then function extensionality shows that equality is equivalent to bisimilarity. We will give two different proofs for function extensionality using the axiom of univalence. In section 3.3 we show the proof from [Uni13, chapter 4.9] while in section 3.4 we show an alternative proof from [Lic14]. Both proofs make use of lemma 3.3.0 which is the only time univalence is used.

3.1 $\sum$-types are Initial Algebras

We will show that $\sum$-types are initial algebras using the general definition from section 2.1. Suppose that $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$, then we take for every $a\colon A$ the functor $F_a\colon \mathrm{Ty} \to \mathrm{Ty}$ given by

$$\begin{aligned} F_a\ X &\equiv\ B\ a, \\ F_a\ f &\equiv \mathrm{id}_{(B\,a)}\,. \end{aligned}$$

We define for $a\colon A$ the injection from $B\,a$ as

$$\begin{aligned} &\mathrm{in}_a\colon B\ a \to (\textstyle\sum_{(a\colon A)} B\ a), \\ &\mathrm{in}_a\ b \equiv (a,\ b); \end{aligned}$$

and see that $(\sum_{(a\colon A)} B\,a,\ (\mathrm{in}_a)_{(a\colon A)})$ is an $(F_a)_{(a\colon A)}$-algebra. We show that it is the initial algebra. Let $(X,\ (c_a)_{(a\colon A)})$ be an $(F_a)_{(a\colon A)}$-algebra. We define the function

$$\begin{aligned} &f\colon (\textstyle\sum_{(a\colon A)} B\ a) \to X, \\ &f\ (a,\ b) \equiv c_a\ b. \end{aligned}$$

For every $a\colon A$ and $b\colon B\,a$ we see that

$$f\ (\mathrm{in}_a\ b) \equiv f\ (a,\ b) \equiv c_a\ b \equiv c_a\ (\mathrm{id}_{(B\,a)}\ b)$$

and with function extensionality we get an $i_a\colon f \circ \mathrm{in}_a = c_a \circ \mathrm{id}_{(B\,a)}$ which shows that the following diagram commutes and that f is an algebra morphism.

$$\begin{array}{ccc} \sum_{(a:\,A)} B\,a & \xrightarrow{f} & X \\ {\scriptstyle \mathrm{in}_a}\uparrow & & \uparrow{\scriptstyle c_a} \\ B\,a & \xrightarrow[\mathrm{id}_{(B\,a)}]{} & B\,a \end{array}$$

Suppose that f' is also a morphism where for every $a\colon A$ we have a $j_a\colon f' \circ \mathrm{in}_a = c_a \circ \mathrm{id}_{(B\,a)}$. We see for every $(a,\, b)\colon \prod_{(a:\,A)} B\,a$ that

$$\begin{aligned} f'\,(a,\, b) &\equiv f'\,(\mathrm{in}_a\, b) \\ &= c_a\,(\mathrm{id}_{(B\,a)}\; b) && \text{by apply}^{=}\; j_a\; b \\ &\equiv f\,(\mathrm{in}_a\, b) \\ &\equiv f\,(a,\, b). \end{aligned}$$

Therefore by function extensionality we have that $f' = f$. We conclude that $(\sum_{(a:\,A)} B\,a,\, (\mathrm{in}_a)_{(a:\,A)})$ is the initial $(F_a)_{(a:\,A)}$-algebra.

3.2 $\prod$-types are Final Coalgebras

We will show that $\prod$-types are initial algebras using the general definition from section 2.2. Suppose that $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$, then we take for every $a\colon A$ the functor $F_a\colon \mathrm{Ty} \to \mathrm{Ty}$ given by

$$\begin{aligned} F_a\; X &\equiv\; B\; a, \\ F_a\;\; f\; &\equiv \mathrm{id}_{(B\,a)}\,. \end{aligned}$$

We define for $a\colon A$ the projection into $B\,a$ as

$$\begin{aligned} &\mathrm{pr}_a\colon (\textstyle\prod_{(a:\,A)} B\; a) \to B\; a, \\ &\mathrm{pr}_a\; p \equiv p\; a; \end{aligned}$$

and see that $(\prod_{(a:\,A)} B\,a,\, (\mathrm{pr}_a)_{(a:\,A)})$ is a $(F_a)_{(a:\,A)}$-coalgebra. We show that it is the final coalgebra. Let $(X,\, (o_a)_{(a:\,A)})$ be an $(F_a)_{(a:\,A)}$-algebra. We define the function

$$\begin{aligned} &f\colon X \to (\textstyle\prod_{(a:\,A)} B\; a), \\ &f\; x \equiv \lambda a.\; o_a\; x. \end{aligned}$$

For every $x\colon X$ we see that

$$\mathrm{pr}_a\,(f\; x) \equiv \mathrm{pr}_a\,(\lambda a.\; o_a\; x) \equiv o_a\; x \equiv \mathrm{id}_{(B\,a)}\,(o_a\; x)$$

and with function extensionality we get an $i_a\colon \mathrm{pr}_a \circ f = \mathrm{id}_{(B\,a)} \circ o_a$ which shows that the following diagram commutes and that f is a coalgebra morphism.

$$\begin{array}{ccc} X & \xrightarrow{f} & \prod_{(a:\,A)} B\,a \\ {\scriptstyle o_a}\downarrow & & \downarrow{\scriptstyle \mathrm{pr}_a} \\ B\,a & \xrightarrow[\mathrm{id}_{(B\,a)}]{} & B\,a \end{array}$$

Suppose that f' is also a morphism where for every $a\colon A$ we have a $j_a\colon \mathrm{pr}_a \circ f' = \mathrm{id}_{(B\,a)} \circ o_a$. We see for every $x\colon X$ and $a\colon A$ that

$$\begin{aligned} f'\ x\ a &\equiv \mathrm{pr}_a\ (f'\ x) \\ &= \mathrm{id}_{(B\,a)}\ (o_a\ x) && \text{by apply}^{=}\ j_a\ x \\ &\equiv \mathrm{pr}_a\ (f\ x) \\ &\equiv f\ x\ a \end{aligned}$$

Therefore by function extensionality we have that $f' = f$. We conclude that $(\prod_{(a\colon A)} B\,a,\ (\mathrm{pr}_a)_{(a\colon A)})$ is the final $(F_a)_{(a\colon A)}$-coalgebra.

Now that we have seen that $(\prod_{(a\colon A)} B\,a,\ (\mathrm{pr}_a)_{(a\colon A)})$ is a coalgebra we define the notion of bisimilarity for $\prod$-types. We call two dependent functions $p_0,\ p_1\colon \prod_{(a\colon A)} B_a$ bisimilar if we have

$$\prod_{(a\colon A)} p_0\ a = p_1\ a.$$

With this definition function extensionality means that equality is equivalent to bisimilarity for $\prod$-types. Therefore, as we will see in the next two chapters, univalence is a way to show that bisimilarity implies equality. In section 4.2 we define bisimilarity for M-types, in this case we will not need univalence to prove that bisimilarity implies equality.

3.3 Proof of Function Extensionality by Contractibility

We assume the axiom of univalence to prove the following lemma, it is not used after that.

Lemma 3.3.0. Let $A\colon \mathrm{Ty}$ and $B,\ B'\colon \mathrm{Ty}$. Then if $e\colon B \to B'$ is an equivalence we also have that $\lambda f.\, e \circ f\colon (A \to B) \to (A \to B')$ is an equivalence.

Proof. By univalance we have that $\mathrm{path\_to\_equi}\colon (B = B') \to (B \simeq B')$ is an equivalence. Suppose that $e\colon B \to B'$ and $\mathrm{isEqui}\colon \mathrm{IsEqui}\, e$. Then we can assume that $(e, \mathrm{isEqui}) \equiv \mathrm{path\_to\_equi}\, p$ for some $p\colon B = B'$. With path induction we assume that $B \equiv B'$ and $p \equiv \mathrm{refl}_B$. But then $e \equiv \mathrm{id}_B$ and

$$\lambda f.\, e \circ f \equiv \lambda f.\, \lambda a.\, e\ (f\ a) \equiv \lambda f.\, \lambda a.\, f\ a \equiv \lambda f.\, f \equiv \mathrm{id}_{A \to B}$$

which shows $\mathrm{IsEqui}\,(\lambda f.\, e \circ f)$. □

For the rest of this proof we first need the following definitions. We take

$$\begin{aligned} &\mathrm{IsContr}\colon \mathrm{Ty} \to \mathrm{Ty}, \\ &\mathrm{IsContr}\ X \equiv \textstyle\sum_{(x\colon X)} \prod_{(x'\colon X)} x' = x; \end{aligned}$$

and call a type $X\colon \mathrm{Ty}$ contractible if $\mathrm{IsContr}\, X$ has terms. For $X,\ Y\colon \mathrm{Ty}$ and $f\colon X \to Y$ we take

$$\begin{aligned} &\mathrm{fib}_f\colon Y \to \mathrm{Ty}, \\ &\mathrm{fib}_f\ y \equiv \textstyle\sum_{(x\colon X)} f\ x = y. \end{aligned}$$

Theorem. (weak function extensionality) Suppose that $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$. Then we have the implication $(\prod_{(a\colon A)} \mathrm{IsContr}\,(B\,a)) \to \mathrm{IsContr}\,(\prod_{(a\colon A)}(B\,a))$.

Proof. Suppose for every $a\colon A$ that $B\,a$ is contractible, then by [Uni13, lemma 4.8.1] we have that $\mathrm{pr}_1\colon (\sum_{(a\colon A)} B\,a) \to A$ is an equivalence. Take the function

$$\begin{aligned}&\alpha\colon (A \to \textstyle\sum_{(a\colon A)} B\ a) \to (A \to A),\\ &\alpha \equiv \lambda f.\ \mathrm{pr}_1 \circ f.\end{aligned}$$

By lemma 3.3.0 we see that α is an equivalence. We see that $\mathrm{fib}_\alpha\ \mathrm{id}_A$ is contractible because α is an equivalence, see [Uni13, section 4.4]. We will show that $(\prod_{(a\colon A)} B\,a) \simeq \mathrm{fib}_\alpha\ \mathrm{id}_A$ and therefore that $\prod_{(a\colon A)} B\,a$ is contractible. First note for $f\colon \prod_{(a\colon A)} B\,a$ that $\lambda a.\,(a,\ f\,a)\colon A \to \sum_{(a\colon A)} B\,a$ and

$$\alpha\ (\lambda a.\ (a,\ f\ a)) \equiv \lambda a.\ \mathrm{pr}_1\ (a, f\ a) \equiv \lambda a.\ a \equiv \mathrm{id}_A,$$

therefore we can define the function

$$\begin{aligned}&\phi\colon (\textstyle\prod_{(a\colon A)} B\ a) \to \mathrm{fib}_\alpha\ \mathrm{id}_A,\\ &\phi\ f \equiv (\lambda a.\ (a,\ f\ a),\ \mathrm{refl}_{\mathrm{id}_A}).\end{aligned}$$

For $(g, p)\colon \mathrm{fib}_\alpha\ \mathrm{id}_A$ we have $p\colon \alpha\,g = \mathrm{id}_A$ therefore we get for $a\colon A$ that $\mathrm{apply}^{=}\,p\,a\colon \mathrm{pr}_1\,(g\,a) = a$ and $\mathrm{tra}_B\,(\mathrm{apply}^{=}\,p\,a)\colon B\,(\mathrm{pr}_1\,(g\,a)) \to B\,a$, now we can define the function

$$\begin{aligned}&\psi\colon \mathrm{fib}_\alpha\ \mathrm{id}_A \to (\textstyle\prod_{(a\colon A)} B\ a),\\ &\psi\ (g,\ p) \equiv \lambda a.\ \mathrm{tra}_B\ (\mathrm{apply}^{=}\ p\ a)\ (\mathrm{pr}_2\ (g\ a)).\end{aligned}$$

We see for $f\colon \prod_{(a\colon A)} B\,a$ that

$$\begin{aligned}\psi\ (\phi\ f) &\equiv \psi\ (\lambda a.\ (a,\ f\ a),\ \mathrm{refl}_{\mathrm{id}_A})\\ &\equiv \lambda a.\ \mathrm{tra}_B\ (\mathrm{apply}^{=}\ \mathrm{refl}_{\mathrm{id}_A}\ a)\ (\mathrm{pr}_2\ (a, f\ a))\\ &\equiv \lambda a.\ \mathrm{tra}_B\ \mathrm{refl}_a\ (f\ a)\\ &\equiv \lambda a.\ \mathrm{id}_{B\,a}\ (f\ a)\\ &\equiv \lambda a.\ f\ a\\ &\equiv f\end{aligned}$$

and in the other direction for $(g,\ p)\colon \mathrm{fib}_\alpha\ \mathrm{id}_A$ that

$$\phi\ (\psi\ (g,\ p)) = (g,\ p) \qquad \text{because } \mathrm{fib}_\alpha\ \mathrm{id}_A \text{ is contractible}$$

which shows that $(\prod_{(a\colon A)} B\,a) \simeq \mathrm{fib}_\alpha\ \mathrm{id}_A$ and that $\prod_{(a\colon A)} B\,a$ is contractible. □

Theorem. (function extensionality) Suppose that $A\colon \mathrm{Ty}$ and $B\colon A \to \mathrm{Ty}$. Then we have for every $f_0,\, f_1\colon \prod_{(a\colon A)} B\,a$ that $\mathrm{apply}^{=}_{f_0,\,f_1}\colon (f_0 = f_1) \to (\prod_{(a\colon A)} f_0\,a = f_1\,a)$ is an equivalence.

Proof. We prove this considering for $f_0\colon \prod_{(a\colon A)} B\,a$ the function

$$\lambda f_1.\ \mathrm{apply}^{=}_{f_0,\,f_1}\colon \prod_{(f_1\colon \prod_{(a\colon A)} B\,a)} (f_0 = f_1) \to (\prod_{(a\colon A)} f_0\ a = f_1\ a).$$

By [Uni13, theorem 4.7.7] we have for $f_1\colon \prod_{(a\colon A)} B\,a$ that $\mathrm{apply}_{f_0,\,f_1}$ is an equivalence if and only if

$$\lambda(f_1,\ p).\ (f_1,\ \mathrm{apply}^{=}_{f_0,\,f_1}\ p)\colon (\textstyle\sum_{(f_1\colon \prod_{(a\colon A)} B\,a)} f_0 = f_1) \to (\sum_{(f_1\colon \prod_{(a\colon A)} B\,a)} \prod_{(a\colon A)} f_0\ a = f_1\ a)$$

is an equivalence. The left side $\sum_{(f_1\colon \prod_{(a\colon A)} B\,a)} (f_0 = f_1)$ is contractible by [Uni13, lemma 3.11.8], therefore it is sufficient to prove that the right side $\sum_{(f_1\colon \prod_{(a\colon A)} B\,a)} \prod_{(a\colon A)} (f_0\,a) = (f_1\,a)$ is also contractible. By [Uni13, theorem 2.15.7] this type is equivalent to $\prod_{(a\colon A)} \sum_{(b\colon B\,a)} (f_0\,a) = b$. For every $a\colon A$ we have that $\sum_{(b\colon B\,a)} (f_0\,a) = b$ is contractible by [Uni13, lemma 3.11.8], therefore by weak function extensionality $\prod_{(a\colon A)} \sum_{(b\colon B\,a)} (f_1\,a) = b$ is contractible. □

3.4 Proof of Function Extensionality by Total Space

For $A,\, B\colon \mathrm{Ty}$ we define

$$\mathrm{PathSpace}\ A \equiv \textstyle\sum_{(s\colon A^2)} (\mathrm{pr}_0\ s = \mathrm{pr}_1\ s);$$
$$\mathrm{BisimulationSpace}\ (A \to B) \equiv \textstyle\sum_{(s\colon (A\to B)^2)} \prod_{(a\colon A)} (\mathrm{pr}_0\ s\ a = \mathrm{pr}_1\ s\ a).$$

In this proof we use univalence in lemma 3.3.0, it is not used after that.

Lemma 3.4.0. For every $A\colon \mathrm{Ty}$ we have $\mathrm{PathSpace}\,A \simeq A$.

Proof. We define by path induction

$$\mathrm{path\_to\_point}\colon \mathrm{PathSpace}\ A \to A,$$
$$\mathrm{path\_to\_point}\ ((a,\ a),\ \mathrm{refl}_a) \equiv a;$$

and we take

$$\mathrm{point\_to\_path}\colon A \to \mathrm{PathSpace}\ A,$$
$$\mathrm{point\_to\_path}\ a \equiv ((a,\ a),\ \mathrm{refl}_a).$$

We see for $a\colon A$ that $\mathrm{path\_to\_point}\,(\mathrm{point\_to\_path}\,a) \equiv a$. In the other direction, for $p\colon \mathrm{PathSpace}\,A$ we assume with path induction that $p \equiv ((a, a), \mathrm{refl}_a)$ and we see $\mathrm{path\_to\_point}\,(\mathrm{point\_to\_path}\,p) \equiv p$. Hence we have $\mathrm{PathSpace}\,A \simeq A$. □

Lemma 3.4.1. Suppose that $A\colon \mathrm{Ty}$ and $B\colon \mathrm{Ty}$. Then we we have

$$(A \to \mathrm{PathSpace}\, B) \simeq \mathrm{BisimulationSpace}\, (A \to B).$$

Proof. Note that

$$A \to \mathrm{PathSpace}\, B \equiv \textstyle\prod_{(a\colon A)} \sum_{(s\colon B^2)} (\mathrm{pr}_0\, s = \mathrm{pr}_1\, s);$$
$$\mathrm{BisimulationSpace}\, (A \to B) \equiv \textstyle\sum_{(s\colon (A\to B)^2)} \prod_{(a\colon A)} (\mathrm{pr}_0\, s\, a = \mathrm{pr}_1\, s\, a).$$

We define

$$\mathrm{rewr}\colon (A \to \mathrm{PathSpace}\, B) \to \mathrm{BisimulationSpace}\, (A \to B),$$
$$\mathrm{rewr}\, g \equiv ((\lambda a.\, \mathrm{pr}_0\, (\mathrm{pr}_0\, (g\, a)),\ \lambda a.\, \mathrm{pr}_1\, (\mathrm{pr}_0\, (g\, a))),\ \lambda a.\, \mathrm{pr}_1\, (g\, a));$$

$$\mathrm{rewr}^{-1}\colon \mathrm{BisimulationSpace}\, (A \to B) \to (A \to \mathrm{PathSpace}\, B),$$
$$\mathrm{rewr}^{-1}\, h \equiv \lambda a.\, (((\mathrm{pr}_0\, (\mathrm{pr}_0\, h))\, a,\ (\mathrm{pr}_1\, (\mathrm{pr}_0\, h))\, a),\ (\mathrm{pr}_1\, h)\, a).$$

For $g\colon A \to \mathrm{Paths}\, B$ we see that

$$\begin{aligned}
\mathrm{rewr}^{-1}\, (\mathrm{rewr}\, g) &\equiv \mathrm{rewr}^{-1}\, ((\lambda a.\, \mathrm{pr}_0\, (\mathrm{pr}_0\, (g\, a)),\ \lambda a.\, \mathrm{pr}_1\, (\mathrm{pr}_0\, (g\, a))),\ \lambda a.\, \mathrm{pr}_1\, (g\, a)) \\
&\equiv \lambda a.\, (((\lambda a.\, \mathrm{pr}_0\, (\mathrm{pr}_0\, (g\, a)))\, a,\ (\lambda a.\, \mathrm{pr}_1\, (\mathrm{pr}_0\, (g\, a)))\, a),\ (\lambda a.\, \mathrm{pr}_1\, (g\, a))\, a) \\
&\equiv \lambda a.\, ((\mathrm{pr}_0\, (\mathrm{pr}_0\, (g\, a)),\ \mathrm{pr}_1\, (\mathrm{pr}_0\, (g\, a))),\ \mathrm{pr}_1\, (g\, a)) \\
&\equiv \lambda a.\, (\mathrm{pr}_0\, (g\, a), \mathrm{pr}_1\, (g\, a)) \\
&\equiv g
\end{aligned}$$

and for $h\colon \mathrm{Bisimilarities}\, (A \to B)$ that

$$\begin{aligned}
\mathrm{rewr}\, (\mathrm{rewr}^{-1}\, h) &\equiv \mathrm{rewr}\, (\lambda a.\, (((\mathrm{pr}_0\, (\mathrm{pr}_0\, h))\, a,\ (\mathrm{pr}_1\, (\mathrm{pr}_0\, h))\, a),\ (\mathrm{pr}_1\, h)\, a)) \\
&\equiv ((\lambda a.\, (\mathrm{pr}_0\, (\mathrm{pr}_0\, h))\, a,\ \lambda a.\, (\mathrm{pr}_1\, (\mathrm{pr}_0\, h))\, a),\ \lambda a.\, (\mathrm{pr}_1\, h)\, a) \\
&\equiv (((\mathrm{pr}_0\, (\mathrm{pr}_0\, h)), (\mathrm{pr}_1\, (\mathrm{pr}_0\, h))),\ (\mathrm{pr}_1\, h)\, a) \\
&\equiv ((\mathrm{pr}_0\, h),\ (\mathrm{pr}_1\, h)) \\
&\equiv h
\end{aligned}$$

therefore we get $(A \to \mathrm{PathSpace}\, B) \simeq \mathrm{BisimulationSpace}\, (A \to B)$. □

Theorem. (function extensionality) Suppose that $A\colon \mathrm{Ty}$ and $B\colon \mathrm{Ty}$. Then we have for every $f_0,\, f_1\colon A \to B$ that $\mathrm{apply}^{=}_{f_0, f_1}\colon (f_0 = f_1) \to (\prod_{(a\colon A)} f_0\, a = f_1\, a)$ is an equivalence.

Proof. We will define an equivalence

$$\mathrm{path\_to\_bisimilarity}\colon \mathrm{PathSpace}\, (A \to B) \to \mathrm{BisimulationSpace}\, (A \to B).$$

We get this equivalence because we have

$$\begin{aligned}
\mathrm{PathSpace}\, (A \to B) &\simeq A \to B && \text{by path_to_point} \\
&\simeq A \to (\mathrm{PathSpace}\, B) && \text{by } \lambda f.\ \mathrm{point_to_path} \circ f \\
&\simeq \mathrm{BisimulationSpace}\, (A \to B) && \text{by rewr}
\end{aligned}$$

by the previous lemmas and lemma 3.3.0. For $p\colon \mathrm{Paths}\,(A \to B)$ we can assume with path induction that $p \equiv ((f, f), \mathrm{refl}_f)$ in which case we see

$$\begin{aligned}
\text{path_to_bisimilarity}\ ((f,\ f),\ \mathrm{refl}_f) &\equiv \mathrm{rewr}\ ((\lambda f.\ \text{point_to_path} \circ f)\ (\text{path_to_point}\ ((f,\ f),\ \mathrm{refl}_f))) \\
&\equiv \mathrm{rewr}\ ((\lambda f.\ \text{point_to_path} \circ f)\ f) \\
&\equiv \mathrm{rewr}\ (\text{point_to_path} \circ f) \\
&\equiv \mathrm{rewr}\ (\lambda a.\ ((f\ a,\ f\ a), \mathrm{refl}_{(f\,a)})) \\
&\equiv ((\lambda a.\ f\ a, \lambda a.\ f\ a),\ \lambda a.\ \mathrm{refl}_{(f\,a)}) \\
&\equiv ((f,\ f),\ \lambda a.\ \mathrm{refl}_{(f\,a)}) \\
&\equiv ((f,\ f),\ \mathrm{apply}^{=}_{f,f}\ \mathrm{refl}_f).
\end{aligned}$$

Now we will use the terminology of [Uni13, section 4.7] and note that path_to_bisimilarity is a total map for the fibrewise map given by

$$\lambda s.\ \mathrm{apply}^{=}_{\mathrm{pr}_0\,s,\,\mathrm{pr}_1\,s}\colon \textstyle\prod_{(s\colon (A\to B)^2)}(\mathrm{pr}_0\ s = \mathrm{pr}_1\ s) \to (\prod_{(a\colon A)}\mathrm{pr}_0\ s\ a = \mathrm{pr}_1\ s\ a).$$

because path_to_bisimilarity is an equivalence we see that $\lambda s.\ \mathrm{apply}^{=}_{\mathrm{pr}_0\,s,\,\mathrm{pr}_1\,s}$ is a fibrewise equivalence which means for every $(f_0,\ f_1)\colon (A \to B)^2$ that $\mathrm{apply}_{f_0,\,f_1}\colon (f_0 = f_1) \to (\prod_{(a\colon A)} f_0\,a = f_1\,a)$ is an equivalence. □

Chapter 4

Five ways to define M-types

4.0 Summary

There are different ways to define W-types, [Uni13, section 5.5] gives three definitions and shows that they are equivalent. In this chapter we will do something similar for M-types; we give five definitions for M-types and we show the implications between these definitions. We start in section 4.1 by giving two ways to define relations in type theory. In section 4.2, we give two corresponding ways to define bisimulations for M-types. Using this in section 4.3 we give the five definitions for M-types and in section 4.4 we show the implications between these definitions.

4.1 Two Types of Relations

There are two main ways two define relations in type theory. We will call these span-relations and dependent-relations. The first definition is based on category theory. A span-relation on a type X consists of a type R and two functions ρ_0, $\rho_1 \colon R \to X$. We say that x_0, $x_1 \colon X$ are related if there exists a $r \colon R$ with $\rho_0\, r = x_0$ and $\rho_1\, r = x_1$.

Definition. (span-relation) For $X \colon \mathrm{Ty}$ we define

$$\mathrm{SpanRel}\, X \equiv \textstyle\sum_{(\mathrm{ty} \colon \mathrm{Ty})} (\mathrm{ty} \to X) \times (\mathrm{ty} \to X)$$

where for $\sim \colon \mathrm{SpanRel}\, X$ we write

$$\begin{aligned} \mathrm{ty}_\sim &\equiv \mathrm{pr}_0\, \sim, & \mathrm{ty}_\sim &\colon \mathrm{Ty}, \\ \rho_{0,\sim} &\equiv \mathrm{pr}_0\, (\mathrm{pr}_1\ \sim), & \rho_{0,\sim} &\colon \mathrm{ty}_\sim \to X, \\ \rho_{1,\sim} &\equiv \mathrm{pr}_1\, (\mathrm{pr}_1\ \sim); & \rho_{1,\sim} &\colon \mathrm{ty}_\sim \to X; \end{aligned}$$

as shown below.

$$X \xleftarrow{\ \rho_{0,\sim}\ } \mathrm{ty}_\sim \xrightarrow{\ \rho_{1,\sim}\ } X$$

For x_0, $x_1 \colon X$ we write

$$(x_0 \sim x_1) \equiv \textstyle\sum_{(s \colon \mathrm{ty}_\sim)} (\rho_{0,\sim}\, s = x_0) \times (\rho_{1,\sim}\, s = x_1).$$

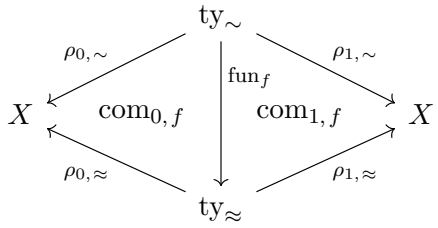
Definition. (span-relation morphism) For $X : \mathrm{Ty}$ and $\sim, \approx : \mathrm{SpanRel}\, X$ we define

$$\mathrm{SpanRelMor} \sim \approx \equiv \textstyle\sum_{(\mathrm{fun} : \mathrm{ty}_\sim \to \mathrm{ty}_\approx)} (\rho_{0,\approx} \circ \mathrm{fun} = \rho_{0,\sim}) \times (\rho_{1,\approx} \circ \mathrm{fun} = \rho_{1,\sim})$$

where for $f : \mathrm{SpanRelMor} \sim \approx$ we write

$$\begin{aligned} \mathrm{fun}_f &\equiv \mathrm{pr}_0\, f, & \mathrm{fun}_f &: \mathrm{ty}_\sim \to \mathrm{ty}\, \approx, \\ \mathrm{com}_{0,f} &\equiv \mathrm{pr}_0\, (\mathrm{pr}_1\, f), & \mathrm{com}_{0,f} &: \rho_{0,\approx} \circ \mathrm{fun}_f = \rho_{0,\sim}, \\ \mathrm{com}_{1,f} &\equiv \mathrm{pr}_1\, (\mathrm{pr}_1\, f); & \mathrm{com}_{1,f} &: \rho_{1,\approx} \circ \mathrm{fun}_f = \rho_{1,\sim}; \end{aligned}$$

as shown below.



Example. (equality) We define $=$ as a span-relation. For $X : \mathrm{Ty}$ take $=_{\mathrm{spanRel}} : \mathrm{SpanRel}\, X$ by

$$\begin{aligned} \mathrm{ty}_{=_{\mathrm{spanRel}}} &\equiv X, \\ \rho_{0,=_{\mathrm{spanRel}}} &\equiv \mathrm{id}_X, \\ \rho_{1,=_{\mathrm{spanRel}}} &\equiv \mathrm{id}_X\,. \end{aligned}$$

For $x_0, x_1 : X$ we can define

$$\begin{aligned} &\mathrm{to} : (x_0 = x_1) \to (x_0 =_{\mathrm{spanRel}} x_1), \\ &\mathrm{to}\, p \equiv (x_0,\ \mathrm{refl}_{x_0},\ p); \end{aligned}$$

$$\begin{aligned} &\mathrm{from} : (x_0 =_{\mathrm{spanRel}} x_1) \to (x_0 = x_1), \\ &\mathrm{from}\, (x,\ p_0,\ p_1) \equiv p_0^{-1} \cdot p_1; \end{aligned}$$

and we see that

$$\begin{aligned} \mathrm{from}\, (\mathrm{to}\, p) &\equiv \mathrm{from}\, (x_0,\ \mathrm{refl}_{x_0},\ p) \\ &\equiv \mathrm{refl}_{x_0}^{-1} \cdot p \\ &= p \end{aligned}$$

$$\begin{aligned} \mathrm{to}\, (\mathrm{from}\, (x,\ \mathrm{refl}_x,\ \mathrm{refl}_x)) &\equiv \mathrm{to}(\mathrm{refl}_x^{-1} \cdot \mathrm{refl}_x) \\ &\equiv \mathrm{to}(\mathrm{refl}_x) \\ &\equiv (x,\ \mathrm{refl}_x,\ \mathrm{refl}_x) \end{aligned}$$

therefore $(x_0 = x_1) \simeq (x_0 =_{\mathrm{spanRel}} x_1)$.

Example. (order) We take $\leq_{\mathrm{spanRel}} : \mathrm{SpanRel}\,\mathbb{N}$ by

$$\begin{aligned}
\mathrm{ty}_{\leq_{\mathrm{spanRel}}} &\equiv \textstyle\sum_{(n_0, n_1 : \mathbb{N})} \sum_{(m : \mathbb{N})} (n_0 + m = n_1), \\
\rho_{0, \leq_{\mathrm{spanRel}}} &\equiv \mathrm{pr}_0, \\
\rho_{1, \leq_{\mathrm{spanRel}}} &\equiv \mathrm{pr}_0 \circ \mathrm{pr}_1.
\end{aligned}$$

If $n_0,\, n_1 : \mathbb{N}$ then $(n_0 \leq_{\mathrm{spanRel}} n_1) \equiv \sum_{((n_0', n_1', m, p) : \mathrm{ty}_{\mathrm{spanRel}})} (n_0' = n_0) \times (n_1' = n_1)$.

Example. (inclusion) Continuing the last two examples we show for $\mathbb{N}$ that we have a morphism from $=_{\mathrm{spanRel}}$ to $\leq_{\mathrm{spanRel}}$. We can define $f : \mathrm{SpanRelMor}\,(=_{\mathrm{spanRel}})\,(\leq_{\mathrm{spanRel}})$ by

$$\begin{aligned}
\mathrm{fun}_f &\equiv \lambda n.\ (n,\ n,\ 0_{\mathbb{N}},\ \mathrm{refl}_n), \\
\mathrm{com}_{0,f} &\equiv \mathrm{funext}\ (\lambda n.\ \mathrm{refl}_n), \\
\mathrm{com}_{1,f} &\equiv \mathrm{funext}\ (\lambda n.\ \mathrm{refl}_n).
\end{aligned}$$

Proposition 4.1.0. For $X : \mathrm{Ty}$ the type $\mathrm{SpanRel}\,X$ forms a category. For $\sim, \approx : \mathrm{SpanRel}\,X$ the morphisms from $\sim$ to $\approx$ are given by $\mathrm{SpanRelMor}\,{\sim}\,{\approx}$. Equality is given by propositional equality, denoted by $=$.

Proof. Let $\sim, \approx, \approxeq : \mathrm{SpanRel}\,X$ with $f : \mathrm{SpanRelMor}\,{\sim}\,{\approx}$ and $g : \mathrm{SpanRelMor}\,{\approx}\,{\approxeq}$, we define

$$\begin{aligned}
&\mathrm{id}_\sim : \mathrm{SpanRelMor}\ {\sim}\ {\sim}, \\
&\mathrm{id}_\sim \equiv (\mathrm{id}_{\mathrm{ty}_\sim},\ \mathrm{refl}_{\rho_{0,\sim}},\ \mathrm{refl}_{\rho_{1,\sim}})
\end{aligned}$$

$$\begin{aligned}
&g \circ f : \mathrm{SpanRelMor}\ {\sim}\ {\approxeq}, \\
&g \circ f \equiv (\\
&\qquad \mathrm{fun}_g \circ \mathrm{fun}_f, \\
&\qquad \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\ \mathrm{com}_{0,g} \cdot \mathrm{com}_{0,f}, \\
&\qquad \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\ \mathrm{com}_{1,g} \cdot \mathrm{com}_{1,f} \\
&).
\end{aligned}$$

Now we see using [Uni13] lemma 2.2.2 that

$$\begin{aligned}
f \circ \mathrm{id}_\sim &\equiv (\mathrm{fun}_f \circ \mathrm{id}_{\mathrm{ty}_\sim},\ \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{id}_{\mathrm{ty}_\sim})}\ \mathrm{com}_{0,f} \cdot \mathrm{refl}_{\rho_{0,\sim}},\ \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{id}_{\mathrm{ty}_\sim})}\ \mathrm{com}_{1,f} \cdot \mathrm{refl}_{\rho_{1,\sim}}) \\
&= (\mathrm{fun}_f,\ \mathrm{com}_{0,f},\ \mathrm{com}_{1,f}) \\
&\equiv f,
\end{aligned}$$

$$\begin{aligned}
\mathrm{id}_\approx \circ f &\equiv (\mathrm{id}_{\mathrm{ty}_\approx} \circ \mathrm{fun}_f,\ \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\ \mathrm{refl}_{\rho_{0,\sim}} \cdot \mathrm{com}_{0,f},\ \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\ \mathrm{refl}_{\rho_{1,\sim}} \cdot \mathrm{com}_{1,f}) \\
&= (\mathrm{fun}_f,\ \mathrm{com}_{0,f},\ \mathrm{com}_{1,f}) \\
&\equiv f;
\end{aligned}$$

and that

$$\begin{aligned}
h \circ (g \circ f) \equiv (& \\
& \mathrm{fun}_h \circ (\mathrm{fun}_g \circ \mathrm{fun}_f), \\
& \mathrm{ap}_{(\lambda k.\, k \circ (\mathrm{fun}_g \circ \mathrm{fun}_f))}\, \mathrm{com}_{0,h} \cdot (\mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\, \mathrm{com}_{0,g} \cdot \mathrm{com}_{0,f}), \\
& \mathrm{ap}_{(\lambda k.\, k \circ (\mathrm{fun}_g \circ \mathrm{fun}_f))}\, \mathrm{com}_{1,h} \cdot (\mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\, \mathrm{com}_{1,g} \cdot \mathrm{com}_{1,f}) \\
)& \\
= (& \\
& (\mathrm{fun}_h \circ \mathrm{fun}_g) \circ \mathrm{fun}_f, \\
& \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\, (\mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_g)}\, \mathrm{com}_{0,h} \cdot \mathrm{com}_{0,g}) \cdot \mathrm{com}_{0,f}, \\
& \mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_f)}\, (\mathrm{ap}_{(\lambda k.\, k \circ \mathrm{fun}_g)}\, \mathrm{com}_{1,h} \cdot \mathrm{com}_{1,g}) \cdot \mathrm{com}_{1,f}, \\
)& \\
\equiv (h \circ g) \circ f.&
\end{aligned}$$

We conclude that $\mathrm{SpanRel}\, X$ forms a category. □

The second type of relation is specific to type theory. As we will see in the examples below it is generally easier to work with. The reason that we define both types of relation is that a bisimulation is easier to define for a span-relation. A dependent-relation on a type X consists of a dependent function $R \colon X \to X \to \mathrm{Ty}$. We say that $x_0, x_1 \colon X$ are related if we have $R\, x_0\, x_1$.

Definition. (dependent-relation) For $X \colon \mathrm{Ty}$ we define

$$\mathrm{DepRel}\, X \equiv X \to X \to \mathrm{Ty};$$

and for $\sim \colon \mathrm{DepRel}\, X$ and $x_0, x_1 \colon X$ we write

$$(x_0 \sim x_1) \equiv \sim x_0\, x_1.$$

Definition. (dependent-relation morphism) For $X \colon \mathrm{Ty}$ and $\sim, \approx \colon \mathrm{DepRel}$ we define

$$\mathrm{DepRelMor} \sim \approx \equiv \prod_{(x_0, x_1 \colon X)} (x_0 \sim x_1) \to (x_0 \approx x_1).$$

Example. (equality) We can also define equality as a dependent-relation. For $X \colon \mathrm{Ty}$ we take $=_{\mathrm{depRel}} \colon \mathrm{DepRel}\, X$ by $(=_{\mathrm{depRel}}) \equiv \lambda x_0.\, \lambda x_1.\, (x_0 = x_1)$. Now for $x_0, x_1 \colon X$ we see by definition that $(x_0 = x_1) \equiv (x_0 =_{\mathrm{depRel}} x_1)$.

Example. (order) We take $\leq_{\mathrm{depRel}} \colon \mathrm{DepRel}\, \mathbb{N}$ by $(\leq_{\mathrm{depRel}}) \equiv \lambda n_0.\, \lambda n_1.\, \sum_{(m \colon \mathbb{N})} (n_0 + m = n_1)$. Then we have for every $n_0, n_1 \colon \mathbb{N}$ that $(n_0 \leq_{\mathrm{depRel}} n_1) \equiv \sum_{(m \colon \mathbb{N})} (n_0 + m = n_1)$.

Example. (inclusion) We show for $\mathbb{N}$ that we have a morphism from $=_{\mathrm{depRel}}$ to $\leq_{\mathrm{depRel}}$. We define

$$\begin{aligned}
& f \colon \mathrm{DepRelMor}\, (=_{\mathrm{depRel}})\, (\leq_{\mathrm{depRel}}), \\
& f\, n_0\, n_1\, p \equiv (0_{\mathbb{N}},\, p).
\end{aligned}$$

Proposition 4.1.0. For $X : \mathrm{Ty}$ the type $\mathrm{DepRel}\, X$ forms a category. For $\sim, \approx : \mathrm{DepRel}\, X$ the morphisms from $\sim$ to $\approx$ are given by $\mathrm{DepRelMor} \sim \approx$. Equality is given by propositional equality, donoted by $=$.

Proof. Let $\sim, \approx, \approxeq : \mathrm{DepRel}\, X$ with $f : \mathrm{DepRelMor} \sim \approx$ and $g : \mathrm{DepRelMor} \approx \approxeq$ then, we define

$$\begin{aligned}
&\mathrm{id}_\sim : \mathrm{DepRelMor} \sim \sim, \\
&\mathrm{id}_\sim \equiv \lambda x_0.\ \lambda x_1.\ \mathrm{id}_{x_0 \sim x_1};
\end{aligned}$$

$$\begin{aligned}
&g \circ f : \mathrm{DepRelMor} \sim \approxeq, \\
&g \circ f \equiv \lambda x_0.\ \lambda x_1.\ g\ x_0\ x_1 \circ f\ x_0\ x_1.
\end{aligned}$$

Now we see that

$$\begin{aligned}
f \circ \mathrm{id}_\sim &\equiv \lambda x_0.\ \lambda x_1.\ f\ x_0\ x_1 \circ \mathrm{id}_{x_0 \sim x_1} \\
&\equiv \lambda x_0.\ \lambda x_1.\ f\ x_0\ x_1 \\
&\equiv f,
\end{aligned}$$

$$\begin{aligned}
\mathrm{id}_\approx \circ f &\equiv \lambda x_0.\ \lambda x_1.\ \mathrm{id}_{x_0 \approx x_1} \circ f\ x_0\ x_1; \\
&\equiv \lambda x_0.\ \lambda x_1.\ f\ x_0\ x_1 \\
&\equiv f;
\end{aligned}$$

and that

$$\begin{aligned}
h \circ (g \circ f) &\equiv \lambda x_0.\ \lambda x_1.\ h\ x_0\ x_1 \circ (g\ x_0\ x_1 \circ f\ x_0\ x_1) \\
&\equiv \lambda x_0.\ \lambda x_1.\ (h\ x_0\ x_1 \circ g\ x_0\ x_1) \circ f\ x_0\ x_1 \\
&\equiv (h \circ g) \circ f.
\end{aligned}$$

We conclude that $\mathrm{DepRel}\, X$ forms a category. □

We can define functions between the two types of relations. For $X : \mathrm{Ty}$ we define

$$\begin{aligned}
&\mathrm{SpanRel_to_DepRel} : \mathrm{SpanRel}\ X \to \mathrm{DepRel}\ X, \\
&\mathrm{SpanRel_to_DepRel} \sim \equiv \lambda x_0.\ \lambda x_1.\ x_0 \sim x_1;
\end{aligned}$$

and

$$\begin{aligned}
&\mathrm{DepRel_to_SpanRel} : \mathrm{DepRel}\ X \to \mathrm{SpanRel}\ X, \\
&\mathrm{DepRel_to_SpanRel} \sim \equiv (\\
&\quad \textstyle\sum_{(x_0.\, x_1 : X)} x_0 \sim x_1, \\
&\quad \mathrm{pr}_0, \\
&\quad \mathrm{pr}_0 \circ \mathrm{pr}_1 \\
&).
\end{aligned}$$

We also define functions between the two types of morphisms. For $\sim, \approx : \mathrm{SpanRel}\, X$ we define

$$\begin{aligned}&\mathrm{SpanRelMor\_to\_DepRelMor}:\\ &\quad \mathrm{SpanRelMor} \sim \approx \to \mathrm{DepRelMor}\ (\mathrm{SpanRel\_to\_DepRel} \sim)\ (\mathrm{SpanRel\_to\_DepRel} \approx),\\ &\mathrm{SpanRelMor\_to\_DepRelMor}\ (\mathrm{fun},\ \mathrm{refl}_{\rho_{0,\sim}},\ \mathrm{refl}_{\rho_{1,\sim}})\ x_0\ x_1\ (s,\ \mathrm{refl}_{x_0},\ \mathrm{refl}_{x_1}) \equiv (\mathrm{fun}\ s,\ \mathrm{refl}_{x_0},\ \mathrm{refl}_{x_1});\end{aligned}$$

and for $\sim, \approx : \mathrm{DepRel}\, X$ we define

$$\begin{aligned}&\mathrm{DepRelMor\_to\_SpanRelMor}:\\ &\quad \mathrm{DepRelMor} \sim \approx \to \mathrm{SpanRelMor}\ (\mathrm{DepRel\_to\_SpanRel} \sim)\ (\mathrm{DepRel\_to\_SpanRel} \approx),\\ &\mathrm{DepRelMor\_to\_SpanRelMor}\ f \equiv (\lambda(x_0,\ x_1,\ s).\ (x_0,\ x_1,\ f\ x_0\ x_1\ s)),\ \mathrm{refl}_{\mathrm{pr}_0},\ \mathrm{refl}_{\mathrm{pr}_0 \circ \mathrm{pr}_1}).\end{aligned}$$

4.2 Two Types of Bisimulations

Now that we have defined relations and have seen that they form categories we look again at polynomial functors. Let P be the polynomial functor for $A : \mathrm{Ty}$ and $B : A \to \mathrm{Ty}$. Now for $X : \mathrm{Ty}$ we get the induced functor $P_{\mathrm{spanRel}} : \mathrm{SpanRel}\, X \to \mathrm{SpanRel}\, (P\, X)$ given by

$$\begin{aligned}P_{\mathrm{spanRel}}\ \sim &\equiv (P\ \mathrm{ty}_\sim,\ P\ \rho_{0,\sim},\ P\ \rho_{1,\sim}),\\ P_{\mathrm{spanRel}}\ f &\equiv (P\ \mathrm{fun}_f,\ \mathrm{ap}_P\ \mathrm{com}_{0,\sim},\ \mathrm{ap}_P\ \mathrm{com}_{0,\sim}).\end{aligned}$$

This functor is called the canonical relation lifting of P, see [BPPR14]. Below we define span-bisimulations, these are also known as AM-bisimulations, see [Sta11].

Definition. (span-bisimulation) For $X : \mathrm{Coalg}_P$ we define

$$\mathrm{SpanBisim}_P\ X \equiv \textstyle\sum_{(\mathrm{coalg} :\, \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\ \mathrm{coalg}\ X \times \mathrm{CoalgMor}_P\ \mathrm{coalg}\ X$$

where for $\sim : \mathrm{SpanBisim}\, X$ we write

$$\begin{aligned}\mathrm{coalg}_\sim &\equiv \mathrm{pr}_0\ \sim, & \mathrm{coalg}_\sim &: \mathrm{Coalg}_P\\ \rho_{0,\sim} &\equiv \mathrm{pr}_0\ (\mathrm{pr}_1\ \sim), & \rho_{0,\sim} &: \mathrm{CoalgMor}_P\ \mathrm{coalg}_\sim\ X,\\ \rho_{1,\sim} &\equiv \mathrm{pr}_1\ (\mathrm{pr}_1\ \sim), & \rho_{1,\sim} &: \mathrm{CoalgMor}_P\ \mathrm{coalg}_\sim\ X,\\ \sim_{\mathrm{spanRel}} &\equiv (\mathrm{ty}_{\mathrm{coalg}_\sim},\ \mathrm{fun}_{\rho_{0,\sim}},\ \mathrm{fun}_{\rho_{1,\sim}}); & \sim_{\mathrm{spanRel}} &: \mathrm{SpanRel}\ \mathrm{ty}_X;\end{aligned}$$

as shown below.

$$\begin{array}{ccccc}
\mathrm{ty}_X & \xleftarrow{\mathrm{fun}_{\rho_{0,\sim}}} & \mathrm{ty}_{\mathrm{coalg}_\sim} & \xrightarrow{\mathrm{fun}_{\rho_{1,\sim}}} & \mathrm{ty}_X \\
{\scriptstyle \mathrm{obs}_X}\downarrow & \mathrm{com}_{\rho_{0,\sim}} & \downarrow{\scriptstyle \mathrm{obs}_{\mathrm{coalg}_\sim}} \mathrm{com}_{\rho_{1,\sim}} & & \downarrow{\scriptstyle \mathrm{obs}_X} \\
P\,\mathrm{ty}_X & \xleftarrow[P\,\mathrm{fun}_{\rho_{0,\sim}}]{} & P\,\mathrm{ty}_{\mathrm{coalg}_\sim} & \xrightarrow[P\,\mathrm{fun}_{\rho_{1,\sim}}]{} & P\,\mathrm{ty}_X
\end{array}$$

For $x_0, x_1 : \mathrm{ty}_X$ we write

$$(x_0 \sim x_1) \equiv (x_0 \sim_{\mathrm{spanRel}} x_1).$$

Example. (equality) For $X : \mathrm{Coalg}_P$ we take $=_{\mathrm{spanBisim}} : \mathrm{SpanBisim}_P$ by

$$\begin{aligned}\mathrm{coalg}_{=_{\mathrm{spanBisim}}} &\equiv X,\\ \rho_{0,=_{\mathrm{spanBisim}}} &\equiv (\mathrm{id}_{\mathrm{ty}_X},\ \mathrm{refl}_{\mathrm{obs}_X}),\\ \rho_{1,=_{\mathrm{spanBisim}}} &\equiv (\mathrm{id}_{\mathrm{ty}_X},\ \mathrm{refl}_{\mathrm{obs}_X}).\end{aligned}$$

We see for $x_0,\ x_1 : \mathrm{ty}_X$ that $(x_0 =_{\mathrm{spanBisim}} x_1) \equiv (x_0 =_{\mathrm{spanRel}} x_1) \simeq (x_0 = x_1)$.

We can also define canonical relation lifting for dependent-relations. For $X : \mathrm{Ty}$ we define the functor $P_{\mathrm{depRel}} : \mathrm{DepRel}\, X \to \mathrm{DepRel}\,(P\, X)$ given by

$$\begin{aligned}P_{\mathrm{depRel}}\ \sim &\equiv \lambda(a_0,\ d_0).\ \lambda(a_1,\ d_1).\ \textstyle\sum_{(p\,:\,a_0 = a_1)} \prod_{(b_0\,:\,B\,a_0)} (d_0\ b_0 \sim d_1\ (\mathrm{tra}_B\ p\ b_0)),\\ P_{\mathrm{depRel}}\ f &\equiv \lambda(a_0,\ d_0).\ \lambda(a_1,\ d_1).\ \lambda(p,\ e).\ (p,\ \lambda b_0.\ f\ (d_0\ b_0)\ (d_1\ (\mathrm{tra}_B\ p\ b_0))\ (e\ b_0)).\end{aligned}$$

Now we can define dependent-bisimulations, these are also known as HJ-bisimulations, see [Sta11].

Definition. (dependent-bisimulation) For $X : \mathrm{Coalg}_P$ we define

$$\begin{aligned}&\mathrm{DepBisim}_P\ X \equiv\\ &\quad \textstyle\sum_{(\mathrm{depRel}\,:\,\mathrm{DepRel}\,\mathrm{ty}_X)}\\ &\quad \textstyle\prod_{(x_0, x_1\,:\,\mathrm{ty}_X)} \mathrm{depRel}\ x_0\ x_1 \to\\ &\quad P_{\mathrm{depRel}}\ \mathrm{depRel}\ (\mathrm{obs}_X\ x_0)\ (\mathrm{obs}_X\ x_1).\end{aligned}$$

where for $\sim : \mathrm{SpanBisim}\, X$ we write

$$\begin{aligned}\sim_{\mathrm{depRel}} &\equiv \mathrm{pr}_0\ \sim, & \sim_{\mathrm{depRel}} &: \mathrm{DepRel}\ \mathrm{ty}_X,\\ \mathrm{funLift}_\sim &\equiv \mathrm{pr}_1\ \sim; & \mathrm{funLift}_\sim &:\\ & & &\quad \textstyle\prod_{(x_0, x_1\,:\,\mathrm{ty}_X)} (x_0 \sim_{\mathrm{depRel}} x_1) \to\\ & & &\quad ((\mathrm{obs}_X\ x_0)\ (P_{\mathrm{depRel}}\ \sim_{\mathrm{depRel}})\ (\mathrm{obs}_X\ x_1)).\end{aligned}$$

For $x_0,\ x_1 : \mathrm{ty}_X$ we write

$$(x_0 \sim x_1) \equiv (x_0 \sim_{\mathrm{depRel}} x_1).$$

Example. (equality) For $X : \mathrm{Coalg}_P$ we define $(=_{\mathrm{depBisim}}) \equiv (=_{\mathrm{spanBisim}}, \mathrm{funLift}_=)$ where

$$\begin{aligned}&\mathrm{funLift}_= :\\ &\quad \textstyle\prod_{(x_0, x_1\,:\,\mathrm{ty}_X)} (x_0 = x_1) \to \sum_{(p\,:\,\mathrm{pr}_0\,(\mathrm{obs}_X\,x_0) = \mathrm{pr}_0\,(\mathrm{obs}_X\,x_1))} \prod_{(b_0\,:\,B\,(\mathrm{pr}_0\,(\mathrm{obs}_X\,x_0)))}\\ &\quad (\mathrm{pr}_1\ (\mathrm{obs}_X\ x_0)\ b_0 = \mathrm{pr}_1\ (\mathrm{obs}_X\ x_1)\ (\mathrm{tra}_B\ p\ b_0)),\\ &\mathrm{funLift}_=\ x\ x\ \mathrm{refl}_x \equiv (\mathrm{refl}_{\mathrm{pr}_0\,(\mathrm{obs}_X\,x)},\ \lambda b.\ \mathrm{refl}_{\mathrm{pr}_1\,(\mathrm{obs}_X\,x)\,b}).\end{aligned}$$

Now we see for $x_0,\ x_1 : \mathrm{ty}_X$ that we have $(x_0 =_{\mathrm{depBisim}} x_1) \equiv (x_0 =_{\mathrm{depRel}} x_1) \equiv (x_0 = x_1)$.

For $X : \mathrm{Ty}$ we have functions

$$\mathrm{SpanBisim\_to\_DepBisim} : \mathrm{SpanBisim}_P\ X \to \mathrm{DepBisim}_P\ X;$$

and

$$\mathrm{DepBisim\_to\_SpanBisim} : \mathrm{DepBisim}_P\ X \to \mathrm{SpanBisim}_P\ X.$$

Such that for every $\sim : \mathrm{SpanBisim}_P\ X$ we have

$$(\mathrm{SpanBisim\_to\_DepBisim}\ \sim)_{\mathrm{depRel}} \equiv \mathrm{SpanRel\_to\_DepRel}\ \sim_{\mathrm{spanRel}},$$

and for every $\sim : \mathrm{DepBisim}_P\ X$ we have that

$$(\mathrm{DepBisim\_to\_SpanBisim}\ \sim)_{\mathrm{spanRel}} \equiv \mathrm{DepRel\_to\_SpanRel}\ \sim_{\mathrm{depRel}}.$$

The definitions for these functions are quite long, we would need two pages to define them. Therefore we will not state them here, instead they can be found in our `Agda` code, see appendix A.

4.3 Different Definitions of M-types

Let $P : \mathrm{Ty} \to \mathrm{Ty}$ be the polynomial functor for $A : \mathrm{Ty}$ and $B : A \to \mathrm{Ty}$. In section 2.7 we have already seen that we can define an M-types as the final coalgebra for such a P. In this section we will give four additional ways to define M-types. We will express each of these ways as a type containing all M-types that meet the definition. These definitions will all start with a coalgebra $\mathsf{M} : \mathrm{Coalg}_P$ and a dependent function $\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\ X\ \mathsf{M}$. Each of these five definitions will then add additional requirements on M and coiter.

We start with the definition of M-types as final coalgebras.

Definition. (final M-types) Suppose that we have $\mathsf{M} : \mathrm{Coalg}_P$ and a dependent function $\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\ X\ \mathsf{M}$, then we define

$$\begin{aligned}&\mathrm{IsFin}\mathsf{M}_P\ \mathsf{M}\ \mathrm{coiter} \equiv \\ &\qquad \textstyle\prod_{(X : \mathrm{Coalg}_P)} \prod_{(f : \mathrm{CoalgMor}_P\ X\ \mathsf{M})} (f = \mathrm{coiter}\ X);\end{aligned}$$

and we take

$$\begin{aligned}&\mathrm{Fin}\mathsf{M}_P \equiv \\ &\qquad \textstyle\sum_{(\mathsf{M} : \mathrm{Coalg}_P)} \sum_{(\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\ X\ M)} \\ &\qquad \mathrm{IsFin}\mathsf{M}_P\ \mathsf{M}\ \mathrm{coiter}.\end{aligned}$$

The postulate from section 2.7 shows that IsFinM_P has a term. Note again that we will do not need this postulate, it is enough to postulate the natural numbers, see [AAG05, BM07, ACS15].

The next definition is similar but gives a form that is somewhat easier to work with, it consists of a uniqueness principle with a coherence law.

Definition. (coherent M-types) Suppose that we have $\mathsf{M} : \mathrm{Coalg}_P$ and a dependent function $\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, \mathsf{M}$, then we define

$$\begin{aligned}&\mathrm{IsCoh}\mathsf{M}_F\ \mathsf{M}\ \mathrm{coiter} \equiv \\ &\quad \prod_{(X : \mathrm{Coalg}_P)} \prod_{(f_0, f_1 : \mathrm{CoalgMor}_P\, X\, \mathsf{M})} \sum_{(p : \mathrm{fun}_{f_0} = \mathrm{fun}_{f_1})} \prod_{(x : \mathrm{ty}_X)} \\ &\quad (\mathrm{ap}_{(\lambda f.\, \mathrm{obs}_{\mathsf{M}}\, (f\, x))}\ p \cdot \mathrm{apply}^{=}\ \mathrm{com}_{f_1}\ x = \mathrm{apply}^{=}\ \mathrm{com}_{f_0}\ x \cdot \mathrm{ap}_{(\lambda f.\, P\, f\, (\mathrm{obs}_X\, x))}\ p);\end{aligned}$$

and we take

$$\begin{aligned}&\mathrm{Coh}\mathsf{M}_F \equiv \\ &\quad \sum_{(\mathsf{M} : \mathrm{Coalg}_P)} \sum_{(\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, M)} \\ &\quad \mathrm{IsCoh}\mathsf{M}_P\ \mathsf{M}\ \mathrm{coiter}.\end{aligned}$$

We will give a diagram for this definition. Let $(\mathsf{M}, \mathrm{coiter}, \mathrm{isCoh}) : \mathrm{Coh}\mathsf{M}_P$ and suppose that we have $X : \mathrm{Coalg}_P$ with $f_0,\ f_1 : \mathrm{CoalgMor}_P\, X\, \mathsf{M}$ and $x : X$. Then we take $p \equiv \mathrm{pr}_0\, (\mathrm{isCoh}\, X\, f_0\, f_1)$ and $\mathrm{coh} : \mathrm{pr}_0\, (\mathrm{isCoh}\, X\, f_0\, f_1)\, x$ to get the following diagram of paths.

$$\begin{array}{ccc} \mathrm{obs}_{\mathsf{M}}\, (\mathrm{fun}_{f_0}\, x) & \xrightarrow{\mathrm{apply}^{=}\, \mathrm{com}_{f_0}\, x} & P\, \mathrm{fun}_{f_0}\, (\mathrm{obs}_X\, x) \\ \Big\downarrow{\scriptstyle \mathrm{ap}_{(\lambda f.\, \mathrm{obs}_{\mathsf{M}}\, (f\, x))}\, p} & \mathrm{coh} & \Big\downarrow{\scriptstyle \mathrm{ap}_{(\lambda f.\, P\, f\, (\mathrm{obs}_X\, x))}\, p} \\ \mathrm{obs}_{\mathsf{M}}\, (\mathrm{fun}_{f_1}\, x) & \xrightarrow[\mathrm{apply}^{=}\, \mathrm{com}_{f_1}\, x]{} & P\, \mathrm{fun}_{f_1}\, (\mathrm{obs}_X\, x) \end{array}$$

The next definition we will consider is similar to that of coherent M-types. The difference is that we only have the uniqueness principle and not the coherence law.

Definition. (bare M-types) Suppose that we have $\mathsf{M} : \mathrm{Coalg}_P$ and a dependent function $\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, \mathsf{M}$, then we define

$$\begin{aligned}&\mathrm{IsBare}\mathsf{M}_F\ \mathsf{M}\ \mathrm{coiter} \equiv \\ &\quad \prod_{(X : \mathrm{Coalg}_P)} \prod_{(f_0, f_1 : \mathrm{CoalgMor}_P\, X\, \mathsf{M})} (\mathrm{fun}_{f_0} = \mathrm{fun}_{f_1});\end{aligned}$$

and we take

$$\begin{aligned}&\mathrm{Bare}\mathsf{M}_F \equiv \\ &\quad \sum_{(\mathsf{M} : \mathrm{Coalg}_P)} \sum_{(\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, M)} \\ &\quad \mathrm{IsBare}\mathsf{M}_P\ \mathsf{M}\ \mathrm{coiter}.\end{aligned}$$

The last two definitions require that every bisimulation of $\mathcal{M}$ has a morphism to equality. We get two definitions, one for each or our definitions of bisimulation.

The first definition uses span-bisimulations.

Definition. (span-bisimulation $\mathcal{M}$-types) Suppose that we have $\mathcal{M} : \mathrm{Coalg}_P$ and a dependent function coiter : $\prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, \mathcal{M}$, then we define

$$\begin{aligned}&\mathrm{IsSpanBisim}\mathcal{M}_P\ \mathcal{M}\ \mathrm{coiter} \equiv \\ &\quad \textstyle\prod_{(\sim : \mathrm{SpanBisim}_P)} \mathrm{SpanRelMor}\ (\sim_{\mathrm{spanRel}})\ (=_{\mathrm{spanRel}});\end{aligned}$$

and we take

$$\begin{aligned}&\mathrm{SpanBisim}\mathcal{M}_P \equiv \\ &\quad \textstyle\sum_{(\mathcal{M} : \mathrm{Coalg}_P)} \sum_{(\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, M)} \\ &\quad \mathrm{IsSpanBisim}\mathcal{M}_P\ \mathcal{M}\ \mathrm{coiter}.\end{aligned}$$

The second definition uses dependent-bisimulations.

Definition. (dependent-bisimulation $\mathcal{M}$-types) Suppose that we have $\mathcal{M} : \mathrm{Coalg}_P$ and a dependent function coiter : $\prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, \mathcal{M}$, then we define

$$\begin{aligned}&\mathrm{IsDepBisim}\mathcal{M}_P\ \mathcal{M}\ \mathrm{coiter} \equiv \\ &\quad \textstyle\prod_{(\sim : \mathrm{DepBisim}_P)} \mathrm{DepRelMor}\ (\sim_{\mathrm{depRel}})\ (=_{\mathrm{depRel}});\end{aligned}$$

and we take

$$\begin{aligned}&\mathrm{DepBisim}\mathcal{M}_P \equiv \\ &\quad \textstyle\sum_{(\mathcal{M} : \mathrm{Coalg}_P)} \sum_{(\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, M)} \\ &\quad \mathrm{IsDepBisim}\mathcal{M}_P\ \mathcal{M}\ \mathrm{coiter}.\end{aligned}$$

These definitions immediately show that bisimilarity implies equality. In the next section we will show that this is true for each of our five definitions. We do this by showing that all five definitions have an implication to both of these last two definitions.

4.4 Implications between Definitions

In this section we will give implications between the five different definitions of $\mathcal{M}$-types from the last section. Once again we let P be the polynomial functor for $A : \mathrm{Ty}$ and $B : A \to \mathrm{Ty}$. Now suppose that we have $\mathcal{M} : \mathrm{Coalg}_P$ and $\mathrm{coiter} : \prod_{(X : \mathrm{Coalg}_P)} \mathrm{CoalgMor}_P\, X\, \mathcal{M}$. We will prove the implications we see on the right. For implications between different definitions of bisimulation see [Sta11].

These results have all been formalised in `Agda`. For this and other formalisations see appendix A. We will give the conclusion and discus possible future work in chapter 5. Here we will also discus the 'missing' implication $\mathrm{IsBare}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter} \to \mathrm{IsCoh}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter}$.

$\mathrm{IsFin}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter}$
$\downarrow\uparrow$
$\mathrm{IsCoh}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter}$
$\downarrow$
$\mathrm{IsBare}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter}$
$\downarrow\uparrow$
$\mathrm{IsSpanBisim}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter}$
$\downarrow\uparrow$
$\mathrm{IsDepBisim}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter}$

Firstly we will need the next two functions, suppose that $X : \mathrm{Ty}$ and $x_0, x_1 : X$, then we take

$$\begin{aligned}
&\mathrm{neutr}^{\cdot}_0 : \textstyle\prod_{(p : x_0 = x_1)} \mathrm{refl}_{x_0} \cdot p = p,\\
&\mathrm{neutr}^{\cdot}_0\ \mathrm{refl}_x \equiv \mathrm{refl}_x;
\end{aligned}$$

$$\begin{aligned}
&\mathrm{neutr}^{\cdot}_1 : \textstyle\prod_{(p : x_0 = x_1)} p \cdot \mathrm{refl}_{x_1} = p,\\
&\mathrm{neutr}^{\cdot}_1\ \mathrm{refl}_x \equiv \mathrm{refl}_x.
\end{aligned}$$

Secondly we note for $X : \mathrm{Ty}$ and $Y : X \to \mathrm{Ty}$ that by the function extensionality axiom we have for every $f_0, f_1 : \prod_{(x : X)} Y\, x$ that

$$\begin{aligned}
&\mathrm{funext\text{-}hom}_0 : \textstyle\prod_{(p : f_0 = f_1)} \mathrm{funext}\ (\mathrm{apply}^{=}\ p) = p;\\
&\mathrm{funext\text{-}hom}_1 : \textstyle\prod_{(h : \prod_{(x : X)} f_0\, x = f_1\, x)} \mathrm{apply}^{=}\ (\mathrm{funext}\ h) = h.
\end{aligned}$$

Now we are ready to show $\mathrm{IsFin}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter} \to \mathrm{IsCoh}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter}$. First suppose that we have $X : \mathrm{Coalg}_P$ with $f_0, f_1 : \mathrm{CoalgMor}_P\, X\, \mathcal{M}$, then we take

$$\begin{aligned}
&\mathrm{coh} : (f_0 = f_1) \to (\\
&\qquad \textstyle\sum_{(p : \mathrm{fun}\, f_0 = \mathrm{fun} f_1)} \prod_{(x : \mathrm{ty}_X)})\\
&\qquad \mathrm{ap}_{(\lambda f.\, \mathrm{obs}_{\mathcal{M}}\, (f\, x))}\ p \cdot \mathrm{apply}^{=}\ \mathrm{com}_{f_1}\ x =\\
&\qquad \mathrm{apply}^{=}\ \mathrm{com}_{f_0}\ x \cdot \mathrm{ap}_{(\lambda f.\, P\, f\, (\mathrm{obs}_X\, x))} p\\
&),\\
&\mathrm{coh}\ \mathrm{refl}_f \equiv (\mathrm{refl}_{\mathrm{fun}_f}, \lambda x.\ (\mathrm{neutr}^{\cdot}_0\ (\mathrm{apply}^{=}\ \mathrm{com}_f\ x)) \cdot (\mathrm{neutr}^{\cdot}_1\ (\mathrm{apply}^{=}\ \mathrm{com}_f\ x))^{-1}).
\end{aligned}$$

Now using this function we define

$$\begin{aligned}
&\mathrm{Fin}\mathcal{M}\mathrm{_to_Coh}\mathcal{M}_P : \mathrm{IsFin}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter} \to \mathrm{IsCoh}\mathcal{M}_P\, \mathcal{M}\, \mathrm{coiter},\\
&\mathrm{Fin}\mathcal{M}\mathrm{_to_Coh}\mathcal{M}_P\ \mathrm{isFin} \equiv \lambda X.\ \lambda f_0.\ \lambda f_1.\ \mathrm{coh}\ ((\mathrm{isFin}\ X\ f_0) \cdot (\mathrm{isFin}\ X\ f_1)^{-1}).
\end{aligned}$$

We will show $\mathrm{IsCoh}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter} \to \mathrm{IsFin}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter}$ in a similar way. First suppose that we have $X : \mathrm{Coalg}_P$ with $f : \mathrm{CoalgMor}_P\,X\,\mathcal{M}$, then we take

$$
\begin{aligned}
&\mathrm{fin} : (\\
&\qquad \textstyle\prod_{(p :\, \mathrm{fun}_f = \mathrm{fun}_{(\mathrm{coiter}\,X)})} \prod_{(x :\, \mathrm{ty}_X)} \\
&\qquad \mathrm{ap}_{(\lambda f.\, \mathrm{obs}_{\mathcal{M}}\,(f\,x))}\; p \cdot \mathrm{apply}^{=}\; \mathrm{com}_{(\mathrm{coiter}\,X)}\; x = \\
&\qquad \mathrm{apply}^{=}\; \mathrm{com}_f\; x \cdot \mathrm{ap}_{(\lambda f.\, P\,f\,(\mathrm{obs}_X\,x))}\; p \\
&) \to (f = \mathrm{coiter}\; X),\\
&\mathrm{fin}\;(\mathrm{refl}_{\mathrm{fun}_f},\; \mathrm{coh}) \equiv \mathrm{pair}^{=}\;(\\
&\qquad \mathrm{refl}_{\mathrm{fun}_f},\\
&\qquad (\text{funext-hom}_0\;\mathrm{com}_f)^{-1} \cdot \\
&\qquad \mathrm{ap}_{\mathrm{funext}}\;(\mathrm{funext}\;(\lambda x.\\
&\qquad\qquad (\mathrm{neutr}_1^{\cdot}\;(\mathrm{apply}^{=}\;\mathrm{com}_f\;x))^{-1} \cdot \\
&\qquad\qquad (\mathrm{coh}\;x)^{-1} \cdot \\
&\qquad\qquad (\mathrm{neutr}_0^{\cdot}\;(\mathrm{apply}^{=}\;\mathrm{com}_{(\mathrm{coiter}\,X)}\;x))\\
&\qquad))\cdot \\
&\qquad (\text{funext-hom}_1\;\mathrm{com}_{(\mathrm{coiter}\,X)})\\
&).
\end{aligned}
$$

Now using this function we define

$$
\begin{aligned}
&\mathrm{Coh}\mathcal{M}\text{\_to\_Fin}\mathcal{M}_P : \mathrm{IsCoh}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter} \to \mathrm{IsFin}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter},\\
&\mathrm{Coh}\mathcal{M}\text{\_to\_Fin}\mathcal{M}_P\;\mathrm{isCoh} \equiv \lambda X.\;\lambda f.\;\mathrm{fin}\;(\mathrm{isCoh}\;X\;f\;(\mathrm{coiter}\;X)).
\end{aligned}
$$

It is easy to show $\mathrm{IsCoh}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter} \to \mathrm{IsBare}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter}$, we define

$$
\begin{aligned}
&\mathrm{Coh}\mathcal{M}\text{\_to\_Bare}\mathcal{M}_P : \mathrm{IsCoh}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter} \to \mathrm{IsBare}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter},\\
&\mathrm{Coh}\mathcal{M}\text{\_to\_Bare}\mathcal{M}_P\;\mathrm{isCoh} \equiv \lambda X.\;\lambda f_0.\;\lambda f_1.\;\mathrm{pr}_0\;(\mathrm{isCoh}\;X\;f_0\;f_1).
\end{aligned}
$$

Next we show that we have the implications $\mathrm{IsBare}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter} \to \mathrm{IsSpanBisim}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter}$ and $\mathrm{IsSpanBisim}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter} \to \mathrm{IsBare}\mathcal{M}_P\,\mathcal{M}\,\mathrm{coiter}$. We define

$$
\begin{aligned}
&\mathrm{Bare}\mathcal{M}\text{\_to\_SpanBisim}\mathcal{M}_P : \mathrm{IsBare}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter} \to \mathrm{IsSpanBisim}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter},\\
&\mathrm{Bare}\mathcal{M}\text{\_to\_SpanBisim}\mathcal{M}_P\;\mathrm{isBare} \equiv \lambda \sim .\;(\mathrm{fun}_{\rho_{0,\sim}},\; \mathrm{refl}_{\rho_{0,\sim}},\; \mathrm{isBare}\;\mathrm{ty}_X\;\rho_{0,\sim}\;\rho_{1,\sim});
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{SpanBisim}\mathcal{M}\text{\_to\_Bare}\mathcal{M}_P : \mathrm{IsSpanBisim}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter} \to \mathrm{IsBare}\mathcal{M}_P\;\mathcal{M}\;\mathrm{coiter},\\
&\mathrm{SpanBisim}\mathcal{M}\text{\_to\_Bare}\mathcal{M}_P\;\mathrm{isSpanBisim} \equiv \lambda X.\;\lambda f_0.\;\lambda f_1.\;\mathrm{funext}\;(\\
&\qquad (\mathrm{apply}^{=}\;\mathrm{com}_{0,\,\mathrm{isSpanBisim}\,(X,\,f_0,\,f_1)}\;x)^{-1} \cdot \\
&\qquad (\mathrm{apply}^{=}\;\mathrm{com}_{1,\,\mathrm{isSpanBisim}\,(X,\,f_0,\,f_1)}\;x)\\
&).
\end{aligned}
$$

Now we start proving the two implications IsSpanBisim$\mathcal{M}_P\,\mathcal{M}$ coiter $\to$ IsDepBisim$\mathcal{M}_P\,\mathcal{M}$ coiter and IsDepBisim$\mathcal{M}_P\,\mathcal{M}$ coiter $\to$ IsSpanBisim$\mathcal{M}_P\,\mathcal{M}$ coiter. For this we first need for X : Ty the functions

SpanRel_to_SpanRel-mor :
$\prod_{(\sim : \mathrm{SpanRel}\, X)}$SpanRelMor $\sim$ (DepRel_to_SpanRel (SpanRel_to_DepRel $\sim$)),
SpanRel_to_SpanRel-mor $\sim \equiv$ (
$\lambda s.\ (\rho_{0,\sim}\ s,\ \rho_{1,\sim}\ s,\ (s,\ \mathrm{refl}_{\rho_{0,\sim}},\ \mathrm{refl}_{\rho_{1,\sim}}))$,
$\mathrm{refl}_{\rho_{0,\sim}}$,
$\mathrm{refl}_{\rho_{1,\sim}}$
);

DepRel_to_DepRel-mor :
$\prod_{(\sim : \mathrm{DepRel}\, X)}$DepRelMor $\sim$ (SpanRel_to_DepRel (DepRel_to_SpanRel $\sim$)),
DepRel_to_DepRel-mor $\sim \equiv \lambda x_0.\ \lambda x_1.\ \lambda s.\ ((x_0,\ x_1,\ s),\ \mathrm{refl}_{x_0},\ \mathrm{refl}_{x_1})$;

and the functions

SpanRel_to_DepRel-mor$^=$: DepRelMor (SpanRel_DepRel $=_{\mathrm{spanRel}}$) $=_{\mathrm{depRel}}$,
SpanRel_to_DepRel-mor$^=$ $\equiv \lambda x_0.\ \lambda x_1.\ \lambda(s,\ p_0,\ p_1).\ p_0^{-1} \cdot p_1$;

DepRel_to_SpanRel-mor$^=$: SpanRelMor (DepRel_SpanRel $=_{\mathrm{depRel}}$) $=_{\mathrm{spanRel}}$,
DepRel_to_SpanRel-mor$^=$ $\equiv$ (
$\lambda(x_0,\ x_1,\ p).\ x_0$,
$\mathrm{refl}_{(\lambda(x_0, x_1, p).\, x_0)}$,
funext $(\lambda(x_0,\ x_1,\ p).\ p)$
).

Now we can define

SpanBisim$\mathcal{M}$_to_DepBisim$\mathcal{M}_P$: IsSpanBisim$\mathcal{M}_P\,\mathcal{M}$ coiter $\to$ IsDepBisim$\mathcal{M}_P$,
SpanBisim$\mathcal{M}$_to_DepBisim$\mathcal{M}_P$ isSpanBisim $\equiv \lambda\sim$.
SpanRel_to_DepRel-mor$^=$ $\circ$
SpanRelMor_to_DepRelMor (isSpanBisim (DepBisim_to_SpanBisim $\sim$)) $\circ$
DepRel_to_DepRel-mor ($\sim_{\mathrm{depRel}}$);

DepBisim$\mathcal{M}$_to_SpanBisim$\mathcal{M}_P$: IsDepBisim$\mathcal{M}_P\,\mathcal{M}$ coiter $\to$ IsSpanBisim$\mathcal{M}_P$,
DepBisim$\mathcal{M}$_to_SpanBisim$\mathcal{M}_P$ isDepBisim $\equiv \lambda\sim$.
DepRel_to_SpanRel-mor$^=$ $\circ$
DepRelMor_to_SpanRelMor (isDepBisim (SpanBisim_to_DepBisim $\sim$)) $\circ$
SpanRel_to_SpanRel-mor ($\sim_{\mathrm{spanRel}}$).

This shows the last of our implications.

Chapter 5

Conclusion and Future Work

We have seen five different definitions of M-types and we have seen the implications between these definitions. This can be summarised in the following graph:

final $\leftrightarrows$ coherent $\longrightarrow$ bare $\leftrightarrows$ span-bisimulation $\leftrightarrows$ dependent-bisimulation

With these implications we are ready to answer our research question. To repeat:

Assuming univalence: Does bisimilarity imply equality for M-types?

We see for each of our definitions that we have an implication to span-bisimulation M-types and to dependent-bisimulation M-types. Therefore we can conclude for each of our five definitions of M-types and for each of our two definitions of bisimulation that bisimilarity implies equality. Moreover, for this we only require the axiom of function extensionality; not the axiom of univalence.

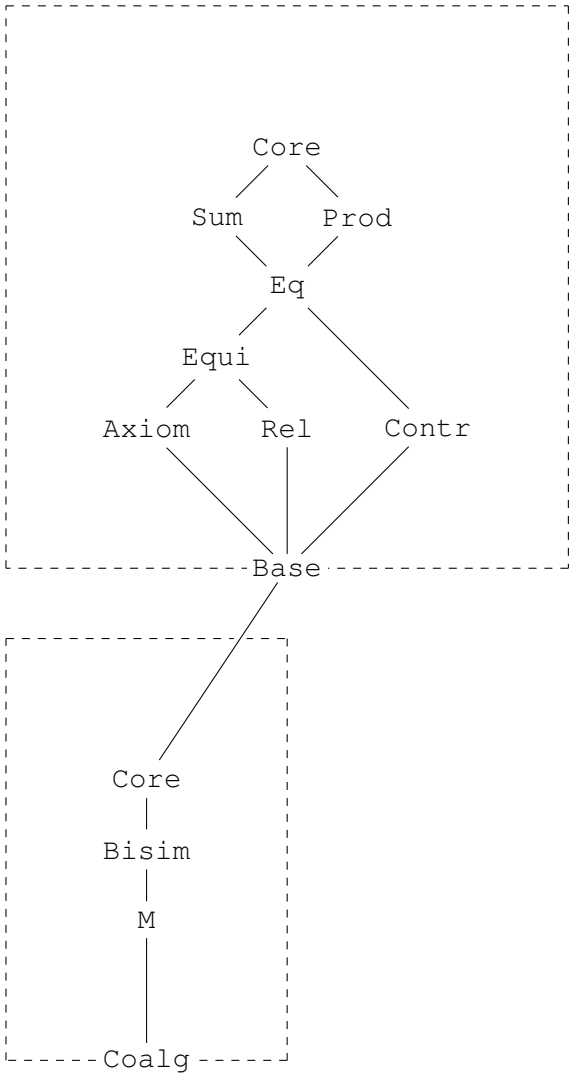
A direction for future work would be to determine if the implications we have given are equivalences. One implication is still missing between our five definitions. This is the implication from bare M-types to coherent M-types. We do not expect this implication to hold. This is because we assume a coherence law in the definition of coherent M-types which is missing from the definition of bare M-types. An interesting direction for future work would be to add such coherence laws to the definitions of span-bisimulation M-types and dependent-bisimulation M-types. With such additions these two definitions might be made equivalent to the definitions of final M-types and coherent M-types.

Another direction for future work is to further explore our definitions of relations and bisimulations. We have given functions between the two types of relations and we have given functions between the two types of relation morphisms. These might give rise to a category equivalences. Additionally we have given functions between the two types of bisimulations. We have defined bisimulation morphisms in `Agda` but we have not given functions between the two types of bisimulation morphisms. This could also be done and might lead to another equivalence of categories.

Appendix A

Agda Code

The results of this thesis are formalised in the computer system `Agda` [Nor07, Agd20]. `Agda` is a programming language and proof assistant based on Martin-Löf type theory. This means that the type theory of `Agda` is strong enough to formalise the results of this thesis. On the right we show the dependency structure of our `.agda` files. They are organised in two folders, `Base` in which we define our basic type theory, and `Coalg` in which we define coalgebras for polynomial functors. Every node in the graph represents a `.agda` file and every file is dependent on the files it is connected to above.



We will give some notes for this code. Firstly `Agda` uses pattern matching which is strong enough to prove axiom K by default, therefore we start every file with the option `--without-K` to disable this. Secondly in this thesis we have opted to use type levels implicitly, `Agda` also allows this with the option `--type-in-type` however we have not used this and have instead defined type levels explicitly in the formalisation. Thirdly `Agda` has a standard library but we have opted not to use this and instead define all necessary types ourselves to achieve better consistency with this thesis. We only use the auto-imported `Agda.Primitive` in which levels are defined. Lastly `Agda` uses $=$ for definitional equality and we follow the convention to define propositional equality as $\equiv$, this means that the symbols $=$ and $\equiv$ are flipped.

Access the code at: `https://github.com/DDOtten/M-types`

Bibliography

[AAG03] M. Abbott, T. Altenkirch, and N. Ghani. Categories of containers. In *International Conference on Foundations of Software Science and Computation Structures*, pages 23–38. Springer, 2003.

[AAG05] M. Abbott, T. Altenkirch, and N. Ghani. Containers: constructing strictly positive types. *Theoretical Computer Science*, 342(1):3–27, 2005.

[ACS15] B. Ahrens, P. Capriotti, and R. Spadotti. Non-wellfounded trees in homotopy type theory. *Proceedings of TLCA 2015*, 38, 2015.

[Agd20] Programming Language group on Agda. *Agda Documentation.* `http://wiki.portal.chalmers.se/agda/`, Accessed: 2020.

[AGS12] S. Awodey, N. Gambino, and K. Sojakova. Inductive types in homotopy type theory. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pages 95–104. IEEE, 2012.

[Bas18] H. Basold. *Mixed Inductive-Coinductive Reasoning.* Nijmegen University, 2018.

[BM07] B. van den Berg and F. De Marchi. Non-well-founded trees in categories. *Annals of Pure and Applied Logic*, 146(1):40 – 59, 2007. doi:https://doi.org/10.1016/j.apal.2006.12.001.

[BPPR14] F. Bonchi, D. Petrişan, D. Pous, and J. Rot. Coinduction up-to in a fibrational setting. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–9, 2014.

[Coq04] The Coq development team. *The Coq proof assistant reference manual.* `http://coq.inria.fr`, 2004. Version 8.0.

[Coq92] T. Coquand. The paradox of trees in type theory. *BIT Numerical Mathematics*, 32(1):10–14, 1992.

[Cpp20] Cppreference. *C Language, Type.* `https://en.cppreference.com/w/c/language/type`, Accessed: 2020.

[GK13] N. GAMBINO and J. KOCK. Polynomial functors and polynomial monads. *Mathematical Proceedings of the Cambridge Philosophical Society*, 154(1):153–192, 2013. doi:10.1017/S0305004112000394.

[ID16] A. D. Irvine and H. Deutsch. Russell's paradox. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[Jec03] T. J. Jech. *Set theory*. Springer monographs in mathematics. Springer, Berlin, 3rd ed. edition, 2003.

[KL12] C. Kapulkin and P. L. Lumsdaine. The simplicial model of univalent foundations (after voevodsky), 2012, 1211.2851.

[Lic14] D. Licata. *Another proof that univalence implies function extensionality*. `https://homotopytypetheory.org/2014/02/17/another-proof-that-univalence-implies-function-extensionality/`, 2014.

[ML84] P. Martin-Löf. *Intuitionistic type theory*. Studies in proof theory. Lecture notes; 1 861180607. Bibliopolis, Napoli, 1984.

[MP00] I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104(1):189 – 218, 2000. doi:https://doi.org/10.1016/S0168-0072(00)00012-9.

[Nor07] U. Norell. *Towards a practical programming language based on dependent type theory*, volume 32. Citeseer, 2007.

[Raa20] P. Raatikainen. Gödel's incompleteness theorems. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2020 edition, 2020.

[Rut19] J. Rutten. *The Method of Coalgebra: exercises in coinduction*. C, Feb. 2019.

[Sta11] S. Staton. Relating coalgebraic notions of bisimulation. *Logical Methods in Computer Science*, Volume 7, Issue 1, Mar. 2011. doi:10.2168/LMCS-7(1:13)2011.

[Str93] T. Streicher. Investigations into intensional type theory. *Habilitiation Thesis, Ludwig Maximilian Universität*, 1993.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.