DE JONGH'S THEOREM FOR TYPE THEORY

MSc Thesis *(Afstudeerscriptie)*

written by

Daniël D. Otten
(born 20 July 1997 in Amsterdam, The Netherlands)

under the supervision of **Dr. Benno van den Berg**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:
12 October 2022

Members of the Thesis Committee:
Dr. Ekaterina Shutova (chair)
Dr. Benno van den Berg (supervisor)
Prof. Dr. Dick de Jongh
Robert Passmann MSc

Institute for Logic, Language and Computation

Abstract

We investigate the relation between arithmetic and type theory, in particular, we will consider: a constructive axiomatisation of the natural numbers known as Second-order Heyting Arithmetic ($\mathsf{HA2}$), and a version of type theory known as Second-order Propositional Lambda Calculus ($\lambda\mathsf{P2}$). We give an embedding of $\mathsf{HA2}$ in $\lambda\mathsf{P2}$, as well as an interpretation of $\lambda\mathsf{P2}$ in $\mathsf{HA2}$. This will allow us to prove our main result: $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ proves exactly the same first-order arithmetical formulas as $\mathsf{HA2}$. We will use this result to analyse the logic of $\lambda\mathsf{P2}$. De Jongh's Theorem shows us that the propositional logic of $\mathsf{HA2}$ is constructive, and we can translate this result to show that the propositional logic of $\lambda\mathsf{P2}$ is also constructive.

Acknowledgements

First and foremost I would like to thank Benno van den Berg for his ample supervision and guidance. He has helped draw me deeper into the captivating and perplexing world of constructive reasoning, type theory, and categorical semantics. I relish looking at problems together, and I am always excited to continue with new ideas after a meeting. I am grateful for the opportunity to start a PhD under his supervision and look forward to further collaboration: both in extending the results of this thesis and in type theory, category theory, and logic in general. I have greatly enjoyed the process of slowly uncovering more of the the connections between natural numbers and types and I want to continue research in this direction.

Furthermore, I am thankful to the members of the thesis committee. Especially to Dick de Jongh for his foundational work in this area, and to Robert Passmann who's recent work in set theory was a direct inspiration for this thesis. Robert was also very helpful at the start of this thesis with pointers to literature and he has helped with setting the right course. I also want to thank Nick Bezhanishvili for taking the time to help me with some of my questions.

Lastly I want to thank my peers. Lide Grotenhuis for proofreading the thesis meticulously, improving some explanations, and correcting many typos. Bart van Kootwijk for helping me prepare my defense and for always being a willing subject to test my presentations on. And finally, Myriam Navarro Beyl for listening to my endless ramblings about numbers, cubes, and the possibility that $1 + 1 = 0$.

Contents

Chapter 0

Introduction and Background

Our motivation comes from the following phenomenon. Suppose that we have formulated a theory in constructive logic (also known as intuitionistic logic). Then it can happen that, in practice, the logic of the theory becomes larger. This is best explained through means of an example. A famous one is Diaconescu's theorem [Dia75]. It shows that the set-theoretic axioms of extensionality, pairing, choice, and the axiom scheme of separation imply that the law of the excluded middle is valid. That is, for every sentence A in the language of set theory, we can prove $A \vee \neg A$.

To reason about this phenomenon, we quickly introduce some terminology. Suppose that we have a theory T, formulated in constructive logic. We say that a logical formula A is *valid* for T iff we have $T \vdash \sigma(A)$ for every every substitution σ. Such a substitution replaces every n-ary relation symbol with a formula in the language of T that has at most n free variables. We already saw an example of this: the law of the excluded middle $P \vee \neg P$ is valid for T iff for every sentence A in the language of T we have $T \vdash A \vee \neg A$. The set of formulas that are valid for T is called the *(De Jongh) logic* of T.

Friedman and Ščedrov [FŠ86] give another example of this phenomenon. They show that the logic becomes larger even when we replace the contentious axiom of choice with the more benign axiom of finite union: extensionality, pairing, finite union, and separation imply that the (De Jongh) propositional logic of the theory is strictly larger than constructive propositional logic. We say that the propositional logic of the theory is not constructive. These results show that the distinction between the underlying logic, and the theory formulated in it, is not always clear cut. For a constructively-minded mathematician, this indicates that we have to be careful with our axioms: not only to obtain a consistent theory, but also to preserve the constructive nature of our logic.

The first results in this area are about Heyting Arithmetic (HA) and Second-order Heyting Arithmetic (HA2). These theories are the constructive versions of Peano Arithmetic (PA) and Second-order Peano Arithmetic (PA2), they have the exact same axioms but the underlying logic is constructive instead of classical. We have the original theorem by Dick de Jongh (an unpublished paper, for an extended abstract see [dJ70]):

the propositional logic of HA is constructive.

This was extended by Daniel Leivant [Lei75] in his PhD thesis:

the first-order logic of HA is constructive.

De Jongh and Smorynski [dJS76] also obtain a result for second-order logic:

the propositional logic of HA2 is constructive.

For a more detailed overview, and a brief history of De Jongh's theorem, see [dJVV09].

Recently, Robert Passmann also achieved results for the set theories IZF and CZF. These are constructive versions of ZF: the axioms are modified, however, if we add the law of the excluded middle to either IZF or CZF, we get ZF [Cro20]. Passmann showed [Pas20; Pas22]:

the propositional logic of IZF is constructive,
the first-order logic of CZF + powerset + choice is constructive.

By the result of Friedman and Ščedrov that we have discusses earlier, we see that the first-order logic of IZF is not constructive. This is because, unlike CZF, the theory IZF contains the full axiom scheme of separation.

In this thesis we will prove similar results for type theory. Type theory can be seen in many different ways: (a) as a foundation of mathematics and alternative for set theory, (b) as a powerful logic that is able to talk about, and quantify over, the proofs themselves, or (c) as a theory of abstract computation and computer programs. It has a number of properties that make it very suitable for constructive mathematics. Important examples of this are the disjunction and existence properties: if we can prove a sentence $A \vee B$ then we can prove A or we can prove B, and if we can prove a sentence $\exists x\, B(x)$ then there actually exists a term a such that we can prove $B(a)$. These properties are true for constructive logic itself, however if we introduce axioms, then it is possible to lose these properties. They remain true for arithmetic: HA and HA2 both have the disjunction and existence properties [Kle45; Bee85, Section IX.2]. However for set theory this is more intricate: CZF and IZF both satisfy the disjunction property [Rat05; Bee85, Chapter VIII], but not the existence property [Swa14; FŠ85]. It is possible but quite difficult to obtain the existence property in set theory: Michael Rathjen showed that this property holds for a number of variations of CZF [Rat12]. For type theories, these properties follow relatively easily from normalisation: a proof of a sentence $A \vee B$ in type theory consists of a proof of A or a proof of B, while a proof of a sentence $\exists x\, A(x)$ consists of a term a and a proof of $A(a)$.

To investigate the logic of type theory, we start with the following observation: if the logic of a theory T is constructive, then for every theory T' that proves no more than T in the language of T, we also have that the logic of T' is constructive. Examples include subtheories and conservative extensions. We can see this as follows: suppose a formula A is not provable constructively. Then A is not an element of the logic of T so there exists a substitution σ to the language of T such that $T \nvdash \sigma(A)$. But then, because T' proves no more than T in the language of T, we also have $T' \nvdash \sigma(A)$. So A is not an element of the logic of T'.

This observation gives us a fruitful way to prove that the logic of type theory is constructive: we prove that a particular type theory proves no more than a theory for which we have already established a 'De Jongh theorem'. This immediately gives us the logic of a simple version of type theory: ML0. This is a fragment of Martin-Löf type theory [ML84]. It is the fragment given by $\mathbb{0}, \mathbb{1}, +, \times, \Sigma, \Pi, \mathbb{N}, =$; so we leave out type universes and W-types. It is known that ML0 proves no more arithmetical formulas than HA [Bee85, Theorem 7.5.1]. It actually proves less arithmetical formulas, see [Smi88]. So, because we already know that the first-order logic of HA is constructive, we see the following:

the first-order logic of ML0 is constructive.

We will use a similar approach for a stronger type theory known as λP2. It is a subsystem of the calculus of constructions, and can be seen as an extension of ML0. The main new addition comes in the form of type universes. In λP2 we have an impredicative universe, which means that types are able to quantify over other types, including over themselves. We will construct a model of λP2 and some of its extensions. This model gives us an interpretation of λP2 in HA2, which we will use to show our main result:

λP2 + ind + funext + uip proves exactly the same first-order arithmetical formulas as HA2.

As a corollary, because the propositional logic of HA2 is constructive, we will see:

the propositional logic of λP2 is constructive.

As we will see, our choice for λP2 has two justifications: it is the minimal system of the lambda cube in which we can embed the formulas of HA2, and it is the maximal system of the lambda cube that we can interpret and realize in HA2.

The structure of this thesis is as follows. In Chapter 1, we start with an overview of arithmetic. Here we introduce second-order logic, and formulate first-order and second-order versions of Heyting Arithmetic (HA and HA2). In Chapter 2, we move on to type theory. We give an overview of the calculus of constructions and focus on λP2. In addition, we will see how the formulas of HA2 can be interpreted as types in λP2.

In the last two chapters we will prove that λP2 + ind + funext + uip and HA2 prove the same first-order arithmetical formulas. In Chapter 3, we will extend arithmetic in a conservative way. This will make it easier to interpret our type theory in arithmetic. We will add lambda functions, recursion, pairs, and a computational choice principle. In Chapter 4, we construct a model for λP2 which gives us an interpretation of λP2 in HA2. This is a modification of a well known model that uses partial equivalence relations (PER's) and assemblies.

Chapter 1

Arithmetic

In this chapter we will introduce constructive theories of arithmetic. We start with an overview of propositional, first-order, and second-order logic in Section 1.0. Second-order logic is an extension of first-order logic that allows us to quantify over formulas. In Section 1.1, we will see that, in second-order logic, we only need $\to$ and $\forall$ to define all the other logical connectives. In Section 1.2 and Section 1.3, we introduce first-order and second-order Heyting Arithmetic respectively. We finish the chapter in Section 1.4 with an overview of De Jongh's Theorem for arithmetic.

1.0 Second-order Logic

Arithmetic, like set theory, is built on top of logic. This means that we start by establishing the logical inference rules, and then state a number of axioms. Often, this is done in classical first-order logic. However, this is not the only option, and there are good reasons to use alternatives. In our case, we will use constructive second-order logic: restricting ourselves to constructive logic will give rise to constructive proofs while extending to second-order logic allows for an elegant, minimal presentation and allows for a finite axiomatisation.

When we introduce type theory in Chapter 2, we will go even further than changing the underlying logic: we will not formulate axioms in an existing logic but build type theory from the ground up using inference rules. This will be done in a way that nicely parallels the inference rules of logic, and, for that reason, we will see that logic can be straightforwardly and canonically interpreted in type theory. So, with this correspondence in mind, we start by introducing second-order logic in a sequent style that mirrors the notation we will use for type theory.

First, we introduce the grammar. We will use lowercase letters like $x, y, z, \ldots$ for first-order variables. The terms are built up from first-order variables, their grammar is given by:

$$a, b, c, \ldots ::= x \mid f^n(a_0, \ldots, a_{n-1}),$$

where f^n stands for a function symbol of arity n in our language. Uppercase letters like $X, Y, Z, \ldots$ are used for second-order variables. When second-order variables are used, they are annotated with a natural number indicating the arity. For example, X^1 stands for a second-order variable of arity 1 which we can view as: a set, a unitary relation symbol, or as a formula with at most one free variable. The formulas are given by:

$$A, B, C, \ldots ::= X^n(a_0, \ldots, a_{n-1}) \mid a = b \mid \bot \mid \top \mid A \vee B \mid A \wedge B \mid A \to B \mid \exists x\, A \mid \forall x\, A \mid \exists X^n\, A \mid \forall X^n\, A.$$

We use letters like Γ and Δ for finite lists of formulas. A sequent $\Gamma \vdash A$ means that from the formulas in Γ we can prove the formula A; we say A holds in context Γ.

Because we are using a sequent style, we begin with the following inference rules:

$$\frac{}{\Gamma, A \vdash A}\,\text{start}, \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\,\text{weak}.$$

For equality, which we consider to be part of our logic, we have the following rules:

$$\frac{}{\Gamma \vdash a = a}\,{=}\text{refl}, \qquad \frac{\Gamma \vdash a = b}{\Gamma \vdash b = a}\,{=}\text{sym}, \qquad \frac{\Gamma \vdash a = b \qquad \Gamma \vdash b = c}{\Gamma \vdash a = c}\,{=}\text{trans},$$

$$\frac{\Gamma \vdash a = b \qquad \Gamma \vdash A(a)}{\Gamma \vdash A(b)}\,{=}\text{subst}.$$

For propositional logic, we add the following introduction and elimination rules:

$$\frac{}{\Gamma \vdash \top}\,\top\mathcal{I}, \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash C}\,\bot\mathcal{E},$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee\mathcal{I}_0, \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee\mathcal{I}_1, \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\,\vee\mathcal{E},$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge\mathcal{I}, \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge\mathcal{E}_0, \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,\wedge\mathcal{E}_1,$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\to\mathcal{I}, \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\,\to\mathcal{E}.$$

As is convention for constructive logic, we do not consider negation to be a primitive notion: we define $\neg A \coloneqq A \to \bot$. To get classical logic, we could add the law of the excluded middle stating $\Gamma \vdash A \vee \neg A$.

Now to get first-order logic we add the following inference rules:

$$\frac{\Gamma \vdash B(a)}{\Gamma \vdash \exists x\, B(x)}\,\exists_1\mathcal{I}, \qquad z \text{ not free in } \Gamma, C\ \frac{\Gamma \vdash \exists x\, B(x) \qquad \Gamma, B(z) \vdash C}{\Gamma \vdash C}\,\exists_1\mathcal{E},$$

$$z \text{ not free in } \Gamma\ \frac{\Gamma \vdash B(z)}{\Gamma \vdash \forall x\, B(x)}\,\forall_1\mathcal{I}, \qquad \frac{\Gamma \vdash \forall x\, B(x)}{\Gamma \vdash B(a)}\,\forall_1\mathcal{E}.$$

On the left side we list restrictions that determine when we are allowed to apply the rules. In this case we have the restriction that z may not occur free, which makes sure that we do not confuse different uses of a variable. Without these restrictions we would be able to prove things like $\forall x\,(A(x) \to \forall y\, A(y))$ and $\forall x(\exists y\, A(y) \to A(x))$. As a note on our notation: $A(x)$ can be any formula, we have no restriction that x is the only free variable, or that x is actually a free variable at all. The reason we include x in the notation is so that we can easily replace it with a term a: $B(a)$ means the formula $B(x)$ except every free occurrence of x is replaced with a. This goes a bit against convention, however, we feel it makes the presentation clearer; in $\exists x\, B(x)$ and $\forall x\, B(x)$ we highlight the fact that x may occur free in $B(x)$ without saying anything about other free variables.

In second-order logic we add the ability to quantify over second-order variables:

$$\frac{\Gamma \vdash B(A(x_0, \ldots, x_{n-1}))}{\Gamma \vdash \exists X^n\, B(X^n)}\,\exists_2\mathcal{I}, \qquad Z^n \text{ not free in } \Gamma, C\ \frac{\Gamma \vdash \exists X^n\, B(X^n) \qquad \Gamma, B(Z^n) \vdash C}{\Gamma \vdash C}\,\exists_2\mathcal{E},$$

$$Z^n \text{ not free in } \Gamma\ \frac{\Gamma \vdash B(Z^n)}{\Gamma \vdash \forall X^n\, B(X^n)}\,\forall_2\mathcal{I}, \qquad \frac{\Gamma \vdash \forall X^n\, B(X^n)}{\Gamma \vdash B(A(x_0, \ldots, x_{n-1}))}\,\forall_2\mathcal{E}.$$

With $B(A(x_0, \ldots, x_{n-1}))$ we mean the formula $B(X^n)$ except every atom of the form $X^n(a_0, \ldots, a_{n-1})$ is replaced by the formula $A(a_0, \ldots, a_{n-1})$. For example, for the formulas $B(X^1) \coloneqq X(x) \wedge \neg X(f^1(y))$ and $A(x) \coloneqq (x = z)$ we have:

$$B(A(x)) = (x = z) \wedge \neg(f^1(y) = z).$$

One helpful way to think about the second-order variable X^n is as an arbitrary abstract formula with n free first-order variables. From this viewpoint, $B(A(x_0, \ldots, x_{n-1}))$ means that we substitute the concrete formula A for the abstract formula X^n where the variables $x_0, \ldots, x_{n-1}$ in A are treated as the n variables of X^n. For example, if $A(x, y)$ is a formula, then from $\forall X^1\,(X(a) \to X(b))$ we can conclude both $A(a, y) \to A(b, y)$ and $A(x, a) \to A(x, b)$ by treating x and y as the single free variable in X^1 respectively. Furthermore, we can conclude $A(x, y) \to A(x, y)$ by treating some other variable z as the free variable in X^1. Note that in the formula $\forall X^1\,(X(a) \to X(b))$ we have stopped writing the arity in places where it can easily be induced, we will follow this convention from now on because it makes formulas less cluttered.

1.1 Defining Logical Connectives

One of the nice properties of second-order logic is that we really only need $\to$ and $\forall$, all other logical connectives can be defined. This means that we can define formulas that satisfy the introduction and elimination rules of the other logical connectives:

$$\begin{aligned}
\bot &\coloneqq \forall Z^0\, Z, && \text{(false)}\\
\top &\coloneqq \forall Z^0\,(Z \to Z), && \text{(true)}\\
A \vee B &\coloneqq \forall Z^0\,((A \to Z) \to (B \to Z) \to Z), && \text{(disjunction)}\\
A \wedge B &\coloneqq \forall Z^0\,((A \to (B \to Z)) \to Z), && \text{(conjunction)}\\
\exists x\, B(x) &\coloneqq \forall Z^0\,(\forall x\,(B(x) \to Z) \to Z), && \text{(first-order existential quantifier)}\\
\exists X^n\, B(X) &\coloneqq \forall Z^0\,(\forall X^n\,(B(X) \to Z) \to Z). && \text{(second-order existential quantifier)}
\end{aligned}$$

This way of defining logical connectives is standard in second-order logic so we will not dwell on this for too long. It is nice to note however that these definitions capture precisely how these logical connectives are used to prove other formulas. It is an insightful exercise to show that the introduction and elimination rules are valid for these definitions, as an example we show this for $\exists X^n\, B(X)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \forall X^n\,(B(X) \to Z) \vdash \forall X^n\,(B(X) \to Z)}\text{start}}{\Gamma, \forall X^n\,(B(X) \to Z) \vdash B(A(\vec{x})) \to Z}\forall_2\mathcal{E} \qquad \dfrac{\Gamma \vdash B(A(\vec{x}))}{\Gamma, \forall X^n\,(B(X) \to Z) \vdash B(A(\vec{x}))}\text{weak}}{\Gamma, \forall X^n\,(B(X) \to Z) \vdash Z}\to\mathcal{E}}{\Gamma \vdash \forall X^n\,(B(X) \to Z) \to Z}\to\mathcal{I}}{\Gamma \vdash \exists X^n\, B(X)}\forall_2\mathcal{I},$$

$$\dfrac{\dfrac{\Gamma \vdash \exists X^n\, B(X)}{\Gamma \vdash (\forall X^n\,(B(X) \to C)) \to C}\forall_2\mathcal{E} \qquad Z \text{ not free in } \Gamma, C \dfrac{\dfrac{\Gamma, B(Z) \vdash C}{\Gamma \vdash B(Z) \to C}\to\mathcal{I}}{\Gamma \vdash \forall X^n\,(B(X) \to C)}\forall_2\mathcal{I}}{\Gamma \vdash C}\to\mathcal{E}.$$

We can also define first-order equality in second-order logic:

$$(a = b) \coloneqq \forall Z^1\,(Z(a) \to Z(b)).$$

This is called Leibniz equality, terms are equal if they satisfy the same formulas. It is easy to show =refl, =trans, and =subst are valid. However, showing that =sym is valid is quite tricky:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash a = b}{\Gamma \vdash (Z(a) \to Z(a)) \to (Z(b) \to Z(a))}\forall_2\mathcal{E} \qquad \dfrac{\dfrac{}{\Gamma, Z(a) \vdash Z(a)}\text{start}}{\Gamma \vdash Z(a) \to Z(a)}\to\mathcal{I}}{\Gamma \vdash Z(b) \to Z(a)}\to\mathcal{E}}{\Gamma \vdash b = a}\forall_2\mathcal{I}.$$

In the $\forall_2\mathcal{E}$ step we fill in the formula $A(x) \coloneqq Z(x) \to Z(a)$.

We can define second-order equality using the principle of extensionality: if $\vec{x}$ are among the free variables of $A(\vec{x})$ and $B(\vec{x})$, then we define:

$$(A(\vec{x}) = B(\vec{x})) \coloneqq \forall \vec{x}\,(A(\vec{x}) \leftrightarrow B(\vec{x})).$$

1.2 Heyting Arithmetic (HA)

Now we are ready to introduce arithmetic. We start with first-order Heyting Arithmetic (HA), which is an axiomatisation of the natural numbers. It has the same axioms as Peano Arithmetic (PA), the only difference is that it uses the inference rules of constructive first-order logic instead of classical first-order logic. The language of HA and PA consist of a nullary function symbol 0, a unary function symbol S, and two binary function symbols $+$ and $\times$. We have two axioms stating that 0 and S are jointly injective:

$$\forall y\,(0 \neq \mathsf{S}(y)), \qquad \forall x\,\forall y\,(\mathsf{S}(x) = \mathsf{S}(y) \to x = y).$$

In addition, we have axioms for addition and multiplication:

$$\begin{aligned} &\forall y\,(0 + y = y), && \forall x\,\forall y\,(\mathsf{S}(x) + y = \mathsf{S}(x + y)),\\ &\forall y\,(0 \times y = 0), && \forall x\,\forall y\,(\mathsf{S}(x) \times y = (x \times y) + x), \end{aligned}$$

And we have an axiom scheme for induction, for every formula $A(x)$ we have the axiom:

$$A(0) \land \forall x(A(x) \to A(\mathsf{S}(x)) \to \forall x\,A(x).$$

One might wonder why we stop at $+$ and $\times$ instead of going further and introducing exponentiation. The reason is that addition and multiplication are enough to do recursion theory. We can already define a pairing function (a bijection $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$). A nice example was given by Cantor [Can77]:

$$\langle a, b\rangle \coloneqq (\tfrac{1}{2} \times (a + b) \times \mathsf{S}(a + b)) + b.$$

This gives us arbitrary tuples:

$$\langle a_0, \ldots, a_{n-1}\rangle \coloneqq \langle a_0, \langle a_1, \ldots, \langle a_{n-2}, a_{n-1}\rangle\rangle\rangle.$$

Now we can define $y^x = z$ to hold iff there exists a sequence $\langle a_0, \ldots, a_x\rangle$ such that:

- $a_0 = 1$,
- for every $i < x$, we have $a_{i+1} = a_i \times y$,
- $a_x = z$.

Other primitive recursive functions can be defined in a similar way. For a more detailed explanation involving more subtleties and coding issues, see [Cai13; HP17; Kay91, Chapter 5].

1.3 Second-order Heyting Arithmetic (HA2)

The only difference between HA2 and HA is that we move to constructive second-order logic and replace the axiom scheme of induction with a single axiom:

$$\forall X^1\,(X(0) \wedge \forall x(X(x) \to X(\mathsf{S}(x)) \to \forall x\, X(x)).$$

Working in second-order logic also allows us to leave out addition and multiplication because we can already do primitive recursion (we will show this in the following proposition). We can define addition, multiplication, and exponentiation with primitive recursion on x:

$$\begin{aligned} 0 + y &\coloneqq y, & \mathsf{S}(x) + y &\coloneqq \mathsf{S}(x + y);\\ 0 \times y &\coloneqq 0, & \mathsf{S}(x) \times y &\coloneqq (x \times y) + y;\\ y^0 &\coloneqq 1, & y^{\mathsf{S}(x)} &\coloneqq y^x \times y. \end{aligned}$$

Proposition 1.3.0 (Primitive Recursion)**.** In HA2 without addition and multiplication, we can construct, for any terms a and $b(x, y)$, the primitive recursive function f defined by:

$$f(0) \coloneqq a, \qquad f(\mathsf{S}(x)) \coloneqq b(x, f(x)).$$

With this we mean that we can prove $\exists F^2\, R(F)$, where:

$$R(F) \coloneqq \forall x\, \exists! y\, F(x, y) \wedge F(0, a) \wedge \forall x\, \forall y\, (F(x, y) \to F(\mathsf{S}(x), b(x, y))).$$

Here $\exists!$ means 'there exists a unique' and is considered to be an abbreviation:

$$\exists! x\, A(x) \coloneqq \exists x\, (A(x) \wedge \forall x'\, (A(x') \to x = x')).$$

Proof. From this point onward we will be less formal when working in second-order logic and only give the important steps of the proofs. We will build this function one piece at a time, starting with a function that is only defined on 0 and in each step extending it to be defined for the next natural number. First we need to define the order on natural numbers:

$$\begin{aligned} &(x \leq y) \coloneqq \forall Z^1\, (Z(y) \wedge \forall w\, (Z(\mathsf{S}(w)) \to Z(w)) \to Z(x)),\\ &(x < y) \coloneqq (\mathsf{S}(x) \leq y). \end{aligned}$$

These definitions are quite straightforward to work with once you get used to second-order logic. For instance, we can prove $(x \leq \mathsf{S}(y)) \to (x \leq y) \vee (x = \mathsf{S}(y))$ by filling in $A(w) \coloneqq (w \leq y) \vee (w = \mathsf{S}(y))$ for Z^1 when eliminating $x \leq \mathsf{S}(y)$. Now because $A(\mathsf{S}(y))$ and $\forall w(A(\mathsf{S}(w)) \to A(w))$ hold we get $A(x)$.

We prove the existence of initial functions g_m by showing $\forall m\, \exists G^2\, R_\leq(G, m)$ where:

$$R_\leq(G, m) \coloneqq \forall (x \leq m)\, \exists! y\, G(x, y) \wedge G(0, a) \wedge \forall (x < m)\, \forall y\, (G(x, y) \to G(\mathsf{S}(x), b(x, y))).$$

We show $\forall m\, \exists G^2\, R_\leq(G, m)$ with induction:

(0) For the base case we take $G_0(x, y) \coloneqq (y = a)$ and we see $R_\leq(G_0, 0)$;

(S) For the successor case, suppose that we have $R_\leq(G_m, m)$, then we can define:

$$G_{\mathsf{S}(m)}(x, y) \coloneqq (x \leq m \wedge G_m(x, y)) \vee (x = \mathsf{S}(m) \wedge \forall w(G_m(m, w) \to y = b(m, w))),$$

and we see $R_\leq(G_{\mathsf{S}(m)}, \mathsf{S}(m))$.

Now we define:

$$F(x, y) \coloneqq \forall G^2(R_\leq(G, x) \to G(x, y)),$$

and we can use this to prove the proposition $\exists F^2\, R(F)$. □

HA2 is often formulated in a fragment of second-order logic: monadic second-order logic, where we only allow quantifying over unary predicates. This means that we only allow second-order quantifiers of the form $\forall X^1$ or $\exists X^1$. How restrictive this fragment is, depends on the availability of tuples. We can define tuples with addition and multiplication as we have seen at the end of Section 1.2. With tuples, the difference between full second-order logic and monadic second-order logic becomes insignificant: we can just replace $X^n(a_0, \ldots, a_{n-1})$ with $X^1(\langle a_0, \ldots, a_{n-1}\rangle)$. Without access to addition and multiplication however, the difference is significant: HA2 without addition and multiplication is decidable in monadic second-order logic [Büc90]. This is of course not true for the full second-order version (or even the first order version with addition and multiplication) as we can do recursion theory and encode the halting problem [Bee85, Proposition 2.4]. In fact, it follows that addition is not definable using monadic second-order logic. This is because we can define multiplication with addition [Avr03]:

$$\begin{aligned}
(x \mid y) &\coloneqq \forall Z^1\,(Z(0) \wedge \forall w(Z(w) \to Z(w+x)) \to Z(y)),\\
(\mathsf{lcm}(x,y) = z) &\coloneqq (z \neq 0) \wedge (x \mid z) \wedge (y \mid z) \wedge \forall w((x \mid w) \wedge (y \mid w) \to (z \mid w)),\\
(x^2 = z) &\coloneqq (x = 0 \wedge z = 0) \vee (\mathsf{lcm}(x, \mathsf{S}(x)) = z + x),\\
(x \times y = z) &\coloneqq ((x+y)^2 = x^2 + z + z + y^2).
\end{aligned}$$

1.4 De Jongh's Theorem

Recall the following terminology from Chapter 0:

- a formula A is called *valid* for a theory T iff for every substitution σ to the language of T we have $T \vdash \sigma(A)$;
- the set of all propositional formulas that are valid for a language T is called the *propositional (De Jongh) logic* of T.

Now, we can formulate two more precise versions of De Jongh's Theorem for HA2, which both show that the propositional logic of HA2 is constructive. To do this, we introduce the arithmetical hierarchy, which can be seen as a measure of the complexity of first-order formulas:

Definition (arithmetical hierarchy)**.** We define sets of first-order formulas Σ^0_n and Π^0_n simultaneously with induction on n:

$$\begin{aligned}
\Sigma^0_0 = \Pi^0_0 &\coloneqq \{A \mid A \text{ is logically equivalent to a quantifier-free formula}\},\\
\Sigma^0_{n+1} &\coloneqq \{A \mid A \text{ is logically equivalent to } \exists x\, B(x) \text{ where } B(x) \in \Pi^0_n\},\\
\Sigma^0_{n+1} &\coloneqq \{A \mid A \text{ is logically equivalent to } \forall x\, B(x) \text{ where } B(x) \in \Sigma^0_n\},
\end{aligned}$$

This hierarchy makes sense both for classical logic, and for constructive logic, which we use here. In constructive logic, unlike classical logic, there are formulas that do no appear in the arithmetical hierarchy [Bur04].

The following theorems are Corollary 4.3 and 4.4 in [dJS76]:

Theorem 1.4.0 (Σ^0_1 version of De Jongh's Theorem for HA2)**.** For every propositional formula A that is not provable constructively, there exists a substitution σ to HA2-formulas in Σ^0_1, such that we have $\mathsf{HA2} \nvdash \sigma(A)$.

Theorem 1.4.1 (uniform Π^0_2 version of De Jongh's Theorem for HA2)**.** There exists a substitution σ to HA2-formulas in Π^0_2, such that for every propositional formula A that is not provable constructively, we have $\mathsf{HA2} \nvdash \sigma(A)$.

Both of these theorems hold for HA as well (this also follows from these theorems), for HA we also know that the first-order logic is constructive [Lei75].

Chapter 2

Type Theory

In this chapter we will introduce a version of type theory called the calculus of constructions, this version is minimalistic but nonetheless captures much of the expressibility of type theory. We start with a general introduction to type theory in Section 2.0. In Section 2.1, we introduce the calculus of constructions, and in Section 2.2, we will discuss the main feature that sets this type theory apart from other versions of type theory: impredicativity. After this, in Section 2.3, we will show how the calculus of constructions can be disassembled in a number of subsystems: the lambda cube.

In Section 2.4, we will focus on a particular subsystem, $\lambda\mathsf{P2}$, and in Section 2.5, we will see that the logical formulas of Second-order Heyting arithmetic ($\mathsf{HA2}$) can be canonically embedded as types in $\lambda\mathsf{P2}$. After this, in Section 2.6 we will see how we inductive and coinductive types can be defined in $\lambda\mathsf{P2}$. We finish our discussion in Section 2.7, by showing that these inductive and coinductive types are quite limited, and giving a number of ways in which we can extend our type theory.

2.0 Introduction

Type theory is an alternative for set theory and can also be used as a formal foundation of mathematics. It has a number of advantages that make it more suitable for computers. Unlike set theory, which is built on top of first-order logic, type theory is constructed from the ground up using inference rules. This means that a lot of important concepts, like functions, disjoint unions, and Cartesian products, are primitive notions instead of definitions. Consequently, it is much easier to represent these concepts inside a computer without falling back to special casing. In addition, in type theory it is decidable whether a given object is an element/term of a given set/type. This is very important because, through the Curry-Howard correspondence, logical propositions are interpreted as types, while proofs are interpreted as terms. Therefore, computers can automatically check the correctness of our proofs. This has led to computer automated proof assistants in the form of dependently typed programming languages, notable examples include: Coq [Coq89], Agda [Agd99], and Lean [Mic13]. For a more thorough discussion of type theory as a foundation, see [Uni13, chapter 1].

Type theory revolves around statements of the form $a : A$. The statement $a : A$ is the analogue of $a \in A$ in set theory and is read as 'a is a term of the type A'. Such statements appear in judgements of the form $x_0 : A_0, \ldots, x_{n-1} : A_{n-1} \vdash a : A$ which is read as 'from $x_0 : A_0, \ldots, x_{n-1} : A_{n-1}$ we can derive $a : A$'. Examples from the natural numbers are given by $\vdash 0 : \mathbb{N}$ and $x : \mathbb{N}, y : \mathbb{N} \vdash x + y : \mathbb{N}$. A particular type theory is given by stating the inference rules that we can use to derive such judgements.

An important distinction between different type theories is the way they handle type universes, that is, types whose terms are again types. Some theories do not include universes while others include an infinite hierarchy of universes. Such a hierarchy can be represented as follows:

$$\underbrace{\underbrace{a : A : \underbrace{\mathcal{A} : \cdots}_{\text{universes}}}_{\text{types}}}_{\text{terms}}$$

We will follow the following notational convention: we use lowercase letters like $a, b, c, \ldots$ in general, uppercase letters like $A, B, C, \ldots$ only when talking about types, and calligraphic letters like $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ only when talking about universes.

2.1 Calculus of Constructions

In the calculus of constructions [CH88; Bar91] we have two universes: Type_0 and Type_1. We think of the types in Type_0 as 'small' types while the types in Type_1 are 'large' types. We use letters like $x, y, z, \ldots$ for variables; the terms, types, and universes are introduced together by the following grammar:

$$a, b, c, \ldots ::= x \mid \Pi(x : a)\, b \mid \lambda(x : a)\, b \mid a\, b \mid \mathsf{Type}_0 \mid \mathsf{Type}_1.$$

Again, note that we will use uppercase and calligraphic letters if we refer to types and universes respectively; this also applies to variables. In particular, $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ and $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \ldots$ always refer to either Type_0 or Type_1. The intuition for the concepts we have just introduced is as follows:

- We view $\Pi(x : A)\, B(x)$ as the type of dependent functions that send every term a in A to a term in $B(a)$; it is called a product. We call these functions dependent because the codomain depends on the specific term in the domain. $\Pi(x : A)\, B(x)$ binds the variable x in $B(x)$. If x does not appear free in $B(x)$ then this is just the type of functions from A to B in which case we will write $\Pi(x : A)\, B$ as $A \to B$.
- We view $\lambda(x : A)\, b(x)$ as the dependent function that sends every term a in A to $b(a)$; we call it a lambda function. $\lambda(x : A)\, b(x)$ binds the variable x in $b(x)$.
- We view $f\, a$ as the application of the dependent function f on the argument a.

Lastly, contexts are denoted by letters like Γ and Δ, and are lists of the form $x_0 : A_0, \ldots, x_{n-1} : A_{n-1}$.

Now we are ready to state the inference rules of the calculus of constructions. First of, the universe Type_0 is itself a term of the universe Type_1:

$$\frac{}{\vdash \mathsf{Type}_0 : \mathsf{Type}_1}\,\text{univ.}$$

In addition, we have the following two rules that allow us to establish basic judgements:

$$x \text{ not free in } \Gamma \quad \frac{\Gamma \vdash A : \mathcal{A}}{\Gamma, x : A \vdash x : A}\,\text{start}, \qquad x \text{ not free in } \Gamma \quad \frac{\Gamma \vdash A : \mathcal{A} \qquad \Gamma \vdash b : B}{\Gamma, x : A \vdash b : B}\,\text{weak}.$$

The restrictions make sure that there are no repetitions among the variables we introduce, the types however can have repetitions. Indeed, a judgement like $x : A, y : A \vdash x : A$ makes sense while a judgement like $x : A, x : B \vdash x : A$ does not. So a type can have multiple terms but a term has in principle only one type, this is one of the mayor differences with set theory.

Now for the product, we have a formation rule, which determines when the product is a well-formed type and in which universe it lives:

$$\frac{\Gamma \vdash A : \mathcal{A} \qquad \Gamma, x : A \vdash B(x) : \mathcal{B}}{\Gamma \vdash \Pi(x : A)\, B(x) : \mathcal{B}}\, \Pi\mathcal{F}.$$

Note that $\Pi(x : A)\, B(x)$ always lives in the same universe as $B(x)$, this is different in other versions of type theory and we will discuss this in the next section. For now, we go on with the introduction, and elimination rules, which determine how we can construct and destruct terms of the product. The introduction rule shows us how we can construct lambda functions:

$$\frac{\Gamma \vdash \Pi(x : A)\, B(x) : \mathcal{B} \qquad \Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda(x : A)\, b(x) : \Pi(x : A)\, B(x)}\, \Pi\mathcal{I}.$$

And the elimination rule shows us how we can evaluate a dependent function:

$$\frac{\Gamma \vdash f : \Pi(x : A)\, B(x) \qquad \Gamma \vdash a : A}{\Gamma \vdash f\, a : B(a)}\, \Pi\mathcal{E}.$$

Lastly we have the concept of β-reduction, which corresponds to reducing an expression by computation. For the product, β-reduction is given by $(\lambda(x : A)\, b(x))\, a \rightsquigarrow_\beta b(a)$. We write $\equiv_\beta$ for β-equivalence, which is the congruence relation induced by β-reduction. More explicitly we have the inference rule:

$$\frac{}{(\lambda(x : A)\, b(x))\, a \equiv_\beta b(a)}\, \Pi\beta.$$

And we have inference rules stating that $\equiv_\beta$ is an equivalence relation:

$$\frac{}{a \equiv_\beta a}\equiv_\beta\text{refl}, \qquad \frac{a \equiv_\beta b}{b \equiv_\beta a}\equiv_\beta\text{sym}, \qquad \frac{a \equiv_\beta b \qquad b \equiv_\beta c}{a \equiv_\beta c}\equiv_\beta\text{trans}.$$

And inference rules stating that $\equiv_\beta$ is compatible with the algebraic structure:

$$\frac{A \equiv_\beta A' \qquad B(x) \equiv_\beta B'(x)}{\Pi(x : A)\, B(x) \equiv_\beta \Pi(x : A')\, B'(x)} \equiv_\beta \Pi\mathcal{F},$$

$$\frac{A \equiv_\beta A' \qquad b(x) \equiv_\beta b'(x)}{\lambda(x : A)\, b(x) \equiv_\beta \lambda(x : A')\, b'(x)} \equiv_\beta \Pi\mathcal{I},$$

$$\frac{f \equiv_\beta f' \qquad a \equiv_\beta a'}{f\, a \equiv_\beta f'\, a'} \equiv_\beta \Pi\mathcal{E}.$$

For β-equivalence we also add the following rule:

$$\frac{\Gamma \vdash a : A \qquad A \equiv_\beta B \qquad \Gamma \vdash B : \mathcal{A}}{\Gamma \vdash a : B}\,\text{conv},$$

Note. We will write the type of functions right associative and function application left associative. So if we have a function $f : A \to B \to C$, then for $a : A$ and $b : B$ we get $f\, a\, b : C$. This technique (writing a function taking multiple arguments, as a function that takes a single argument and returns a new function) is called Currying after Haskell Curry. Moses Schönfinkel used the technique 6 years earlier however the name 'Schönfinkelling' never caught on [Cur80]. It will be very useful to us because we have yet to define the Cartesian product.

2.2 Impredicativity

Now that we have introduced the calculus of construction, let us take the time to discuss the most important aspect that sets it apart from other type theories: that it is impredicative (types can quantify over themselves). This is best explained with an example: if we have $\Gamma, X : \mathsf{Type}_0 \vdash B(X) : \mathsf{Type}_0$, then we get $\Gamma \vdash \Pi(X : \mathsf{Type}_0)\, B(X) : \mathsf{Type}_0$. So, although $\Pi(X : \mathsf{Type}_0)\, B(X)$ quantifies over the types in Type_0 (we view Π as a quantifier), we see that $\Pi(X : \mathsf{Type}_0)\, B(X)$ is itself a term of Type_0 in context Γ. This aspect is very useful because it allows us to define new types in Type_0 analogously to the way that we can define logical connectives in second-order logic:

$$\begin{aligned}
\mathbb{0} &\coloneqq \Pi(Z : \mathsf{Type}_0)\, Z, && \text{(initial type)}\\
\mathbb{1} &\coloneqq \Pi(Z : \mathsf{Type}_0)\, (Z \to Z), && \text{(final type)}\\
A + B &\coloneqq \Pi(Z : \mathsf{Type}_0)\, ((A \to Z) \to (B \to Z) \to Z), && \text{(binary sum)}\\
A \times B &\coloneqq \Pi(Z : \mathsf{Type}_0)\, ((A \to B \to Z) \to Z), && \text{(binary product)}\\
\Sigma(x : A)\, B(x) &\coloneqq \Pi(Z : \mathsf{Type}_0)\, (\Pi(x : A)\, (B(x) \to Z) \to Z), && \text{(sum)}\\
(a =_A b) &\coloneqq \Pi(Z : A \to \mathsf{Type}_0)\, (Z\, a \to Z\, b) && \text{(equality)}
\end{aligned}$$

Note the similarity to the logical connectives:

$$\begin{aligned}
\bot &\coloneqq \forall Z^0\, Z, && \text{(false)}\\
\top &\coloneqq \forall Z^0\, (Z \to Z), && \text{(true)}\\
A \vee B &\coloneqq \forall Z^0\, ((A \to Z) \to (B \to Z) \to Z), && \text{(disjunction)}\\
A \wedge B &\coloneqq \forall Z^0\, ((A \to B \to Z) \to Z), && \text{(conjunction)}\\
\exists x\, B(x) &\coloneqq \forall Z^0\, (\forall x\, (B(x) \to Z) \to Z), && \text{(existential quantifier)}\\
(a = b) &\coloneqq \forall Z^1\, (Z(a) \to Z(b)). && \text{(equality)}
\end{aligned}$$

In addition, we can define the natural numbers:

$$\mathbb{N} \coloneqq \Pi(Z : \mathsf{Type}_0)\, (Z \to (Z \to Z) \to Z). \qquad \text{(natural numbers)}$$

We will discuss these new types in more detail in Section 2.4.

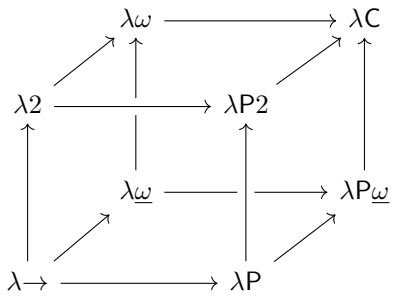
This impredicativity does make it harder to find models for the calculus of construction. Because of cardinality issues, there only exists a very simple classical set theoretic model. With such a classical model we mean that we interpret the universe Type_0 as a set $\mathcal{U}$ such that for every set $A \in \mathcal{U} \cup \{\mathcal{U}\}$ and family of sets $(B_a \in \mathcal{U})_{a \in A}$ we have $\prod_{a \in A} B_a \in \mathcal{U}$. In particular this means for sets $A, B \in \mathcal{U}$ that we have $A \to B \coloneqq \prod_{a \in A} B \in \mathcal{U}$. Suppose there exists such a model, and that there exists an $A \in \mathcal{U}$ with at least two elements; then we get a contradiction:

$$\begin{aligned}
|\textstyle\prod_{B \in \mathcal{U}} (B \to A)| &= |(\textstyle\sum_{B \in \mathcal{U}} B) \to A| && \text{(by Currying)}\\
&\geq |\mathcal{P}(\textstyle\sum_{B \in \mathcal{U}} B)| && \text{(because } |A| \geq 2\text{)}\\
&\geq |\mathcal{P}(\textstyle\prod_{B \in \mathcal{U}} (B \to A))| && \text{(because } \textstyle\prod_{B \in \mathcal{U}} (B \to A) \in \mathcal{U}\text{)}\\
&> |\textstyle\prod_{B \in \mathcal{U}} (B \to A)|.
\end{aligned}$$

This proof comes from lectures of Hyland and Streicher, see also [LM91; Reu99]. Note that there does exist a model where every $A \in \mathcal{U}$ has at most one element [Smi88]. This means that it is consistent to assume that all terms of a small type are equal. We can also construct less trivial models where this is not the case, however, we have to carefully avoid cardinality problems. In Chapter 4 we will construct a model that does this by putting a restriction on function, we only consider functions that are in some sense computable.

2.3 The Lambda Cube

Barendregt [Bar91] showed that the calculus of constructions can be restricted in a systematic way to yield less expressive type theories. In this way, the calculus of constructions λC can be viewed as a natural extension of type theories which were already studied, most notably: simply-typed lambda calculus $\lambda\!\!\rightarrow$ and second-order lambda calculus $\lambda 2$ (also known as polymorphic lambda calculus or System F). These subsystems are represented in the following diagram, which explains the name 'lambda cube'; each arrow represents an inclusion:



To explain this diagram we begin with by describing simply typed lambda-calculus. This subsystem is the restriction of the calculus of constructions where we only allow the product formation rule for $(\mathcal{A}, \mathcal{B}) = (\mathsf{Type}_0, \mathsf{Type}_0)$:

$$\frac{\Gamma \vdash A : \mathcal{A} \qquad \Gamma, x : A \vdash B(x) : \mathcal{B}}{\Gamma \vdash \Pi(x : A)\, B(x) : \mathcal{B}}\, \Pi\mathcal{F}.$$

Now, in each axis of the lambda cube we add a different way to form product types by allowing an additional pair $(\mathcal{A}, \mathcal{B})$ in $\Pi\mathcal{F}$:

$(\rightarrow)$ we allow $(\mathsf{Type}_0, \mathsf{Type}_1)$ to add types that depend on terms (dependent types),

$(\uparrow)$ we allow $(\mathsf{Type}_1, \mathsf{Type}_0)$ to add terms that depend on types (polymorphism),

$(\nearrow)$ we allow $(\mathsf{Type}_1, \mathsf{Type}_1)$ to add types that depend on types (type operators).

This can be summarised in the following table:

System	Pairs $(\mathcal{A}, \mathcal{B})$ allowed in the product formation rule $\Pi\mathcal{F}$			
$\lambda\!\!\rightarrow$	$(\mathsf{Type}_0, \mathsf{Type}_0)$			
$\lambda\mathsf{P}$	$(\mathsf{Type}_0, \mathsf{Type}_0)$	$(\mathsf{Type}_0, \mathsf{Type}_1)$		
$\lambda 2$	$(\mathsf{Type}_0, \mathsf{Type}_0)$		$(\mathsf{Type}_1, \mathsf{Type}_0)$	
$\lambda\mathsf{P}2$	$(\mathsf{Type}_0, \mathsf{Type}_0)$	$(\mathsf{Type}_0, \mathsf{Type}_1)$	$(\mathsf{Type}_1, \mathsf{Type}_0)$	
$\lambda\underline{\omega}$	$(\mathsf{Type}_0, \mathsf{Type}_0)$			$(\mathsf{Type}_1, \mathsf{Type}_1)$
$\lambda\mathsf{P}\underline{\omega}$	$(\mathsf{Type}_0, \mathsf{Type}_0)$	$(\mathsf{Type}_0, \mathsf{Type}_1)$		$(\mathsf{Type}_1, \mathsf{Type}_1)$
$\lambda\omega$	$(\mathsf{Type}_0, \mathsf{Type}_0)$		$(\mathsf{Type}_1, \mathsf{Type}_0)$	$(\mathsf{Type}_1, \mathsf{Type}_1)$
λC	$(\mathsf{Type}_0, \mathsf{Type}_0)$	$(\mathsf{Type}_0, \mathsf{Type}_1)$	$(\mathsf{Type}_1, \mathsf{Type}_0)$	$(\mathsf{Type}_1, \mathsf{Type}_1)$

In fact, the lambda cube can be generalised further into what we call pure type systems. These are type theories where we only have products as primitive notions (like the calculus of constructions). We specify a pure type system by picking a set U of universes, a set $\mathsf{A} \subseteq \mathsf{U} \times \mathsf{U}$ of universe axioms, and a set $\mathsf{R} \subseteq \mathsf{U} \times \mathsf{U} \times \mathsf{U}$ of formation rules. A pure type system uses the same rules as the calculus of constructions except the univ and $\Pi\mathcal{F}$ rules are modified as follows:

$$(\mathcal{A}, \mathcal{B}) \in \mathsf{A} \frac{}{\Gamma \vdash \mathcal{A} : \mathcal{B}}\,\text{univ}, \qquad (\mathcal{A}, \mathcal{B}, \mathcal{C}) \in \mathsf{R} \frac{\Gamma \vdash A : \mathcal{A} \qquad \Gamma, x : A \vdash B(x) : \mathcal{B}}{\Gamma \vdash \Pi(x : A)\, B(x) : \mathcal{C}}\,\Pi\mathcal{F}.$$

Now we can generalise the calculus of constructions to have an infinite amount of universes:

- $\mathsf{U} = \{\mathsf{Type}_u \mid u \in \mathbb{N}\}$,
- $\mathsf{A} = \{(\mathsf{Type}_u, \mathsf{Type}_{u+1}) \mid u \in \mathbb{N}\}$,
- $\mathsf{R} = \{(\mathsf{Type}_u, \mathsf{Type}_0, \mathsf{Type}_0) \mid u \in \mathbb{N}\} \cup \{(\mathsf{Type}_u, \mathsf{Type}_v, \mathsf{Type}_{\max(u,v)}) \mid u, v \in \mathbb{N}, v > 0\}$.

Confusingly, this system (with an infinite amount of universes) is also called the calculus of constructions. Note that Type_0 is still impredicative but that the other universes are predicative (types in these universes can not quantify over themselves). This is necessary, only the lowest universe can be impredicative, otherwise the system is inconsistent by Girard's paradox [Gir72; Coq86; Hur95] which can be viewed as an encoding of Russell's paradox in type theory. Now we can also explain the fundamental difference between the calculus of constructions and another important version of type theory: Martin-Löf type theory [ML84]. Martin-Löf type theory is completely predicative, it can be described with the same universes U and universe axioms A but with formation rules:

$$\mathsf{R}' = \{(\mathsf{Type}_u, \mathsf{Type}_v, \mathsf{Type}_{\max(u,v)}) \mid u, v \in \mathbb{N}\}.$$

Besides products, Martin-Löf type theory also includes other primitive ways to construct types. These can already be defined in the calculus of constructions as we will see in the next section.

2.4 Second-order Predicate Lambda Calculus (λP2)

In this thesis we will focus mainly on $\lambda\mathsf{P2}$. This has two reasons:

- $\lambda\mathsf{P2}$ is the minimal system of the lambda cube in which we can embed $\mathsf{HA2}$ in a canonical way:
  - We need terms dependent on types in order to define the natural numbers $\mathbb{N}$ and to define the type constructors $\mathbb{0}, \mathbb{1}, +, \times, \Sigma$ that we will use to interpret the first-order logical connectives $\bot, \top, \vee, \wedge, \exists$ respectively.
  - We need types dependent on terms to form $\Pi(X : \mathbb{N} \to \cdots \to \mathbb{N} \to \mathsf{Type}_0)\, B(X)$, which is used to interpret the second-order $\forall$. In addition, we need it to define equality inside of our type theory.
- $\lambda\mathsf{P2}$ is the maximal system in the lambda cube that we can interpret in $\mathsf{HA2}$, this interpretation is based on partial equivalence relations and is discussed in detail in Chapter 4.

For our purposes, it is more convenient to consider some of the notions that can be defined in $\lambda\mathsf{P2}$ as primitive. This is intended to mirror the approach we took for $\mathsf{HA2}$ and this means that the embedding of $\lambda\mathsf{P2}$ in $\mathsf{HA2}$ and the interpretation of $\lambda\mathsf{P2}$ in $\mathsf{HA2}$ are simplified slightly. This means that, for the rest of this thesis, we will be working with a definitional extension of $\lambda\mathsf{P2}$. In particular, we will add $\mathbb{N}$, $\mathbb{0}$, $\mathbb{1}$, $A + B$, $\Sigma(x : A)\, B(x)$, and $a =_A a'$ as primitive notions with associated inference rules. $A \times B$ will be a special case of $\Sigma(x : A)\, B(x)$ where x does not appear free in $B(x)$. In the remainder of this section we will state the different additional rules we assume, and then show how these concepts can already be defined in $\lambda\mathsf{P2}$:

(ℕ) We start by adding a type of natural numbers $\mathbb{N}$, it has a very simple formation rule:

$$\frac{}{\vdash \mathbb{N} : \mathsf{Type}_0}\,\mathbb{N}\mathcal{F}.$$

In addition, we have introduction rules for 0 and S:

$$\frac{}{\vdash 0 : \mathbb{N}}\,\mathbb{N}\mathcal{I}_0, \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{S}\, n : \mathbb{N}}\,\mathbb{N}\mathcal{I}_{\mathsf{S}}.$$

The elimination rule is more complicated, it shows that we can define a function from the natural numbers to any other small type using recursion:

$$\frac{\Gamma \vdash C : \mathsf{Type}_0 \qquad \Gamma \vdash c : C \qquad \Gamma \vdash f : C \to C \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{rec}_{\mathbb{N}}\, C\, c\, f\, n : C}\,\mathbb{N}\mathcal{E}.$$

The term $\mathsf{rec}_{\mathbb{N}}\, C\, c\, f\, n$ is the result of taking the term c and applying the function f a total of n times, we can see this with the following β-reduction rules:

$$\frac{}{\mathsf{rec}_{\mathbb{N}}\, C\, c\, f\, 0 \equiv_\beta c}\,\mathbb{N}\beta_0, \qquad \frac{}{\mathsf{rec}_{\mathbb{N}}\, C\, c\, f\, (\mathsf{S}\, n) \equiv_\beta f(\mathsf{rec}_{\mathbb{N}}\, C\, c\, f\, n)}\,\mathbb{N}\beta_{\mathsf{S}}.$$

Types and terms satisfying these rules can already be defined:

$$\begin{aligned}
\mathbb{N} &:= \Pi(Z : \mathsf{Type}_0)\,(Z \to (Z \to Z) \to Z),\\
0 &:= \lambda(Z : \mathsf{Type}_0)\,\lambda(z : Z)\,\lambda(f : Z \to Z)\, z,\\
\mathsf{S}\, n &:= \lambda(Z : \mathsf{Type}_0)\,\lambda(z : Z)\,\lambda(f : Z \to Z)\, f\,(n\, Z\, z\, f),\\
\mathsf{rec}_{\mathbb{N}}\, C\, c\, f\, n &:= n\, C\, c\, f.
\end{aligned}$$

It is very insightful to show that these types and terms indeed satisfy the inference rules.

In fact, we will even go one step further and add addition and multiplication as primitive concepts, for this we add the following rules:

$$\frac{\Gamma \vdash n : \mathbb{N} \qquad \Gamma \vdash n' : \mathbb{N}}{\Gamma \vdash n + n' : \mathbb{N}}\,\mathbb{N}\mathcal{I}_+, \qquad \frac{\Gamma \vdash n : \mathbb{N} \qquad \Gamma \vdash n' : \mathbb{N}}{\Gamma \vdash n \times n' : \mathbb{N}}\,\mathbb{N}\mathcal{I}_\times,$$

$$\frac{}{0 + n' \equiv_\beta n'}\,\mathbb{N}\beta_{+,0}, \qquad \frac{}{0 \times n' \equiv_\beta 0}\,\mathbb{N}\beta_{\times,0},$$

$$\frac{}{\mathsf{S}\, n + n' \equiv_\beta \mathsf{S}\,(n + n')}\,\mathbb{N}\beta_{+,\mathsf{S}}, \qquad \frac{}{\mathsf{S}\, n \times n' \equiv_\beta (n \times n') + n'}\,\mathbb{N}\beta_{\times,\mathsf{S}}.$$

Types and terms satisfying these rules can also be defined:

$$\begin{aligned}
n + n' &:= \mathsf{rec}_{\mathbb{N}}\, \mathbb{N}\, n'\, (\lambda(x : \mathbb{N})\, \mathsf{S}\, x)\, n,\\
n \times n' &:= \mathsf{rec}_{\mathbb{N}}\, \mathbb{N}\ 0\ (\lambda(x : \mathbb{N})\, x + n')\, n.
\end{aligned}$$

This shows us how we can use recursion to define new functions.

(𝟘) We also add an initial type $\mathbb{0}$, with formation rule:

$$\frac{}{\Gamma \vdash \mathbb{0} : \mathsf{Type}_0}\,\mathbb{0}\mathcal{F}.$$

This will be the empty type so there should be no way to construct a term of $\mathbb{0}$. In fact, if we are able to find a term of $\mathbb{0}$ then we say that the type theory is inconsistent, this is similar to how a logical theory is called inconsistent if it can prove $\bot$. We have no introduction rule (for obvious reasons), but we do have an elimination rule:

$$\frac{\Gamma \vdash C : \mathsf{Type}_0 \qquad \Gamma \vdash o : \mathbb{0}}{\Gamma \vdash \mathsf{rec}_{\mathbb{0}}\, C\, o : C}\,\mathbb{0}\mathcal{E}.$$

This rule is similar to the principle of explosion, if we can find a term of $\mathbb{0}$ then we can find a term of any small type. This also holds in set theory: there exists a function from the empty set to any other set (this function is also the empty set). We do not need $\mathbb{0}\beta$ rules as there should be no way to construct a term of type $\mathbb{0}$.

Again, types and terms satisfying these rules can already be defined:

$$\begin{aligned} \mathbb{0} &\coloneqq \Pi(Z : \mathsf{Type}_0)\, Z, \\ \mathsf{rec}_{\mathbb{0}}\, C\, o &\coloneqq o\, C. \end{aligned}$$

(𝟙) Next, we add the final type $\mathbb{1}$ with formation rule:

$$\frac{}{\Gamma \vdash \mathbb{1} : \mathsf{Type}_0}\,\mathbb{1}\mathcal{F}.$$

This type will be a singleton, we have the following introduction rule:

$$\frac{}{\Gamma \vdash * : \mathbb{1}}\,\mathbb{1}\mathcal{I}.$$

The elimination rule is not very useful, we add it anyway for completeness:

$$\frac{\Gamma \vdash C : \mathsf{Type}_0 \qquad \Gamma \vdash c : C \qquad \Gamma \vdash i : \mathbb{1}}{\Gamma \vdash \mathsf{rec}_{\mathbb{1}}\, C\, c\, i : C}\,\mathbb{1}\mathcal{E}.$$

The term $\mathsf{rec}_{\mathbb{1}}\, c\, i$ will be the same as c as the following β-reduction rule states:

$$\frac{}{\mathsf{rec}_{\mathbb{1}}\, C\, c * \equiv_\beta c}\,\mathbb{1}\beta.$$

Again, types and terms satisfying these rules can already be defined:

$$\begin{aligned} \mathbb{1} &\coloneqq \Pi(Z : \mathsf{Type}_0)\, (Z \to Z), \\ * &\coloneqq \lambda(Z : \mathsf{Type}_0)\, \lambda(z : Z)\, z, \\ \mathsf{rec}_{\mathbb{1}}\, C\, c\, i &\coloneqq i\, C\, c. \end{aligned}$$

(+) For the binary sum we have the following introduction rule:

$$\frac{\Gamma \vdash A : \mathsf{Type}_0 \qquad \Gamma \vdash B : \mathsf{Type}_0}{\Gamma \vdash A + B : \mathsf{Type}_0} +\mathcal{F}.$$

We can view $A + B$ as the disjoint union of A and B, the terms of $A + B$ will be given by the injections of terms in A and the injections of terms in B. This is shown in the introduction rules:

$$\frac{\Gamma \vdash A + B : \mathsf{Type}_0 \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{in}_0\, a : A + B} +\mathcal{I}_0, \qquad \frac{\Gamma \vdash A + B : \mathsf{Type}_0 \qquad \Gamma \vdash b : B}{\Gamma \vdash \mathsf{in}_1\, b : A + B} +\mathcal{I}_1.$$

The elimination rule shows that for any small type C we can define a function $A + B \to C$ if we have a function $A \to C$ and a function $B \to C$:

$$\frac{\Gamma \vdash C : \mathsf{Type}_0 \qquad \Gamma \vdash f : A \to C \qquad \Gamma \vdash g : B \to C \qquad \Gamma \vdash t : A + B}{\Gamma \vdash \mathsf{rec}_+\, C\, f\, g\, t : C} +\mathcal{E}.$$

The term $\mathsf{rec}_+\, C\, f\, g\, t$ is calculated by applying f if t is the injection of a term in A and by applying g if t is the injection of a term in B:

$$\frac{}{\mathsf{rec}_+\, C\, f\, g\, (\mathsf{in}_0\, a) \equiv_\beta f\, a} +\beta_0, \qquad \frac{}{\mathsf{rec}_+\, C\, f\, g\, (\mathsf{in}_1\, b) \equiv_\beta g\, b} +\beta_1.$$

Again, types and terms satisfying these rules can already be defined:

$$\begin{aligned} A + B &\coloneqq \Pi(Z : \mathsf{Type}_0)\, ((A \to Z) \to (B \to Z) \to Z), \\ \mathsf{in}_0\, a &\coloneqq \lambda(Z : \mathsf{Type}_0)\, \lambda(f : A \to Z)\, \lambda(g : B \to Z)\, f\, a, \\ \mathsf{in}_1\, b &\coloneqq \lambda(Z : \mathsf{Type}_0)\, \lambda(f : A \to Z)\, \lambda(g : B \to Z)\, g\, b, \\ \mathsf{rec}_+\, C\, f\, g\, t &\coloneqq t\, C\, f\, g. \end{aligned}$$

(Σ) For the sum we have the following introduction rule:

$$\frac{\Gamma \vdash A : \mathsf{Type}_0 \qquad \Gamma, x : A \vdash B(x) : \mathsf{Type}_0}{\Gamma \vdash \Sigma(x : A)\, B(x) : \mathsf{Type}_0} \Sigma\mathcal{F}.$$

This will be the type of dependent pairs: the specific element on the left determines the type that the element on the right can have. We can also view it as the disjoint union of the types $B(x)$ over $x : A$. We have the introduction rule:

$$\frac{\Gamma \vdash \Sigma(x : A)\, B(x) : \mathsf{Type}_0 \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B(a)}{\Gamma \vdash \langle a, b \rangle : \Sigma(x : A)\, B(x)} \Sigma\mathcal{I}.$$

The elimination rule shows that for any small type C, we can define a function $\Sigma(x : A)\, B(x) \to C$, if we have a function $\Pi(x : A)\, (B(x) \to C)$:

$$\frac{\Gamma \vdash C : \mathsf{Type}_0 \qquad \Gamma \vdash f : \Pi(x : A)\, (B(x) \to C) \qquad \Gamma \vdash p : \Sigma(x : A)\, B(x)}{\Gamma \vdash \mathsf{rec}_\Sigma\, C\, f\, p : C} \Sigma\mathcal{E},$$

The term $\mathsf{rec}_\Sigma\, C\, f\, p$ applies the function f to the two elements of p:

$$\frac{}{\mathsf{rec}_\Sigma\, C\, f\, (\langle a, b \rangle) \equiv_\beta f\, a\, b} \Sigma\beta.$$

Again, types and terms satisfying these rules can already be defined:

$$\begin{aligned}\Sigma(x : A)\, B(x) &\coloneqq \Pi(Z : \mathsf{Type}_0)\, (\Pi(x : A)\, (B(x) \to Z) \to Z),\\ \langle a, b\rangle &\coloneqq \lambda(Z : \mathsf{Type}_0)\, \lambda(f : \Pi(x : A)\, (B(x) \to Z))\, f\, a\, b,\\ \mathsf{rec}_\Sigma\, C\, f\, p &\coloneqq p\, C\, f,\end{aligned}$$

Similar to how $A \to B$ is special notation for $\Pi(x : A)\, B$ if x does not appear free in B, we will write $A \times B$ as special notation for $\Sigma(x : A)\, B$ if x does not appear free in B. In this case it is just the type of pairs (the Cartesian product).

(=) The last new type we add is equality. Because we will interpret formulas as types, equality of two terms $a : A$ and $a' : A$ will also be a type $a =_A a'$:

$$\frac{\Gamma \vdash A : \mathcal{A} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash a =_A a' : \mathsf{Type}_0} =\mathcal{F}.$$

The way to show that equality holds is by reflexivity:

$$\frac{\Gamma \vdash a =_A a \ : \mathsf{Type}_0}{\Gamma \vdash \mathsf{refl}\, A\, a : a =_A a} =\mathcal{I}.$$

The elimination rule for equality is the concept of substitution:

$$\frac{\Gamma \vdash C : A \to \mathsf{Type}_0 \qquad \Gamma \vdash c : C\, a \qquad \Gamma \vdash e : a =_A a'}{\Gamma \vdash \mathsf{rec}_=\, C\, c\, e : C\, a'} =\mathcal{E}.$$

As β-reduction rule we see that substituting by reflexivity is the identity:

$$\frac{}{\mathsf{rec}_=\, C\, c\, (\mathsf{refl}\, A\, a)\ \equiv_\beta\ c} =\beta.$$

Again, types and terms satisfying these rules can already be defined:

$$\begin{aligned}(a =_A a') &\coloneqq \Pi(Z : A \to \mathsf{Type}_0)\, (Z\, a \to Z\, a'),\\ \mathsf{refl}\, A\, a &\coloneqq \lambda(Z : A \to \mathsf{Type}_0)\, \lambda(z : Z\, a)\, z,\\ \mathsf{rec}_=\, C\, c\, e &\coloneqq e\, C\, c.\end{aligned}$$

We also have extra $\equiv_\beta$ rules stating that $\equiv_\beta$ is compatible with the new algebraic structures we have added, these rules are all analogous to the $\equiv_\beta \Pi\mathcal{F}$, $\equiv_\beta \Pi\mathcal{I}$, and $\equiv_\beta \Pi\mathcal{E}$ rules at the end of Section 2.1 and we will not write them out.

These new types are examples of inductive types, which we will discuss further in Section 2.6. First, we want to show the embedding of HA2 in λP2 in the next section.

2.5 Embedding Arithmetic in Type Theory

We can interpret propositions of $\mathsf{HA2}$ as types in $\lambda\mathsf{P2}$ through the Curry-Howard correspondence [dG95]. Note that we have already introduced the natural numbers in $\lambda\mathsf{P2}$, and we see that the terms of $\mathsf{HA2}$ are also terms of $\mathbb{N}$ in $\lambda\mathsf{P2}$ (because we have added $0, \mathsf{S}, +, \times$ as primitive notions). We embed the formulas of $\mathsf{HA2}$ as types in Type_0 as follows:

$$\begin{aligned}
(X^n(a_0, \ldots, a_{n-1}))^* &:= X\, a_0\, \ldots\, a_{n-1},\\
(a = a')^* &:= (a =_{\mathbb{N}} a'),\\
\bot^* &:= \mathbb{0},\\
\top^* &:= \mathbb{1},\\
(A \vee B)^* &:= A^* + B^*\\
(A \wedge B)^* &:= A^* \times B^*\\
(A \to B)^* &:= A^* \to B^*,\\
(\exists x\, B(x))^* &:= \Sigma(x : \mathbb{N})\, B(x)^*,\\
(\forall x\, B(x))^* &:= \Pi(x : \mathbb{N})\, B(x)^*,\\
(\exists X^n\, B(X))^* &:= \Sigma(X : \mathbb{N} \to \cdots \to \mathbb{N} \to \mathsf{Type}_0)\, B(X)^*,\\
(\forall X^n\, B(X))^* &:= \Pi(X : \underbrace{\mathbb{N} \to \cdots \to \mathbb{N}}_{n} \to \mathsf{Type}_0)\, B(X)^*.
\end{aligned}$$

This is a special example of the propositions-as-types and proofs-as-terms interpretation. With this interpretation we view the types in Type_0 as propositions and we view $=, \mathbb{0}, \mathbb{1}, +, \times, \Sigma, \Pi$ as the logical connectives $=, \bot, \top, \vee, \wedge, \exists, \forall$. The terms are then seen as proofs that the proposition is true.

In fact, the Curry-Howard correspondence gives us a general way to relate type theory to logic. Each subsystem of the lambda cube corresponds to a logic that is implicative (it only includes $\to$ and $\forall$) and constructive (no law of the excluded middle):

System	Corresponding Implicative Constructive Logic
$\lambda\!\to$	First-Order Propositional Logic
$\lambda 2$	Second-Order Propositional Logic
$\lambda\underline{\omega}$	Weak Higher-Order Propositional Logic
$\lambda\omega$	Higher-Order Propositional Logic
$\lambda\mathsf{P}$	First-Order Predicate Logic
$\lambda\mathsf{P2}$	Second-Order Predicate Logic
$\lambda\mathsf{P}\underline{\omega}$	Weak Higher-Order Predicate Logic
λC	Higher-Order Predicate Logic

Note that $\lambda\mathsf{P2}$ does indeed correspond to Second-order Predicate Logic. Here 'predicate' means that we can talk about arbitrary predicates over a type $A : \mathsf{Type}_0$, such a predicate is interpreted as a term of $A \to \cdots \to A \to \mathsf{Type}_0$. In contrast, $\lambda 2$ which is only 'propositional' can only express nullary predicates (truth values) which are interpreted as terms of Type_0. This is still enough for most of our definitions of the last section, only equality can not be defined in $\lambda 2$.

2.6 (Co)inductive Types

In $\lambda 2$, we can define an inductive type I with constructors of the form $C_0(I), \ldots, C_{n-1}(I)$ as follows:

$$I \coloneqq \Pi(Z : \mathsf{Type}_0)\,(C_0(Z) \to \cdots \to C_{n-1}(Z) \to Z).$$

The inductive type I is seen as the type generated by the constructors $C_0(I), \ldots, C_{n-1}(I)$. We can see it as the initial object in a category whose objects are types $X : \mathsf{Type}_0$ with constructors $C_0(X), \ldots, C_{n-1}(X)$. This is a very powerful way to define new types, and gives the intuition behind the new types of Section 2.4 (except for equality which is a bit more complicated):

($\mathbb{N}$) For the natural numbers $\mathbb{N}$, we want constructors $0 : \mathbb{N}$ and $\mathsf{S} : \mathbb{N} \to \mathbb{N}$, so we define:

$$\mathbb{N} \coloneqq \Pi(Z : \mathsf{Type}_0)\,(Z \to (Z \to Z) \to Z).$$

($\mathbb{0}$) For the initial type $\mathbb{0}$, we want no constructors, so we define:

$$\mathbb{0} \coloneqq \Pi(Z : \mathsf{Type}_0)\,Z.$$

($\mathbb{1}$) For the final type $\mathbb{1}$, we want a constructor $* : \mathbb{1}$, so we define:

$$\mathbb{1} \coloneqq \Pi(Z : \mathsf{Type}_0)\,(Z \to Z).$$

($+$) For $A + B$, we want constructors $\mathsf{in}_0 : A \to A + B$ and $\mathsf{in}_1 : B \to A + B$, so we define:

$$A + B \coloneqq \Pi(Z : \mathsf{Type}_0)\,((A \to Z) \to (B \to Z) \to Z).$$

(Σ) For $\Sigma(x : A)\,B(x)$, we want a constructor $\mathsf{pair} : \Pi(x : A)\,(B(x) \to \Sigma(x : A)\,B(x))$, so we define:

$$\Sigma(x : A)\,B(x) \coloneqq \Pi(Z : \mathsf{Type}_0)\,(\Pi(x : A)\,(B(x) \to Z) \to Z)$$

It is important to note that there do exist some limitations on the types of the constructors that are allowed, see [Gir89, Chapter 11].

If we dualize inductive types which gives the concept of coinductive types. A coinductive type C is defined by giving the forms $D_0(C), \ldots, D_{n-1}(C)$ of its destructors [Geu94, chapter 3]. We can define the coinductive type C as follows:

$$C \coloneqq \Sigma(Z : \mathsf{Type}_0)\,(D_0(Z) \times \cdots \times D_{n-1}(Z) \times Z).$$

An example of a coinductive type is the type of streams (infinite lists of natural numbers), we want destructors $\mathsf{head} : \mathsf{Stream} \to \mathbb{N}$ and $\mathsf{tail} : \mathsf{Stream} \to \mathsf{Stream}$ so we take:

$$\mathsf{Stream} \coloneqq \Sigma(Z : \mathsf{Type}_0)\,((Z \to \mathbb{N}) \times (Z \to Z) \times Z).$$

2.7 Extensions

In this section we will introduce some of the additional inference rules that we can add to the calculus of constructions. A reason for considering such extensions is that, although the calculus of constructions is quite expressive, it still leaves quite some statements undecided.

On the one hand, we can define inductive types as we have seen in the previous section. In fact, all algorithmic functions on inductive types whose termination can be proven in higher-order logic can already be expressed in second-order lambda calculus ($\lambda 2$) [Str92]. For the natural numbers this means that the functions representable in $\lambda 2$ are precisely those that are provably total in $\mathsf{HA2}$ (which are precisely those that are provably total in $\mathsf{PA2}$) [Gir89, Chapter 15].

However on the other hand, when we try to prove things, we run into limitations, consider for example the three axioms of $\mathsf{HA2}$, their embeddings in $\lambda\mathsf{P2}$ are:

$$\Pi(x : \mathbb{N})\,((\mathsf{S}\,x = 0) \to \mathbb{0}),$$
$$\Pi(x : \mathbb{N})\,\Pi(y : \mathbb{N})\,(\mathsf{S}\,x = \mathsf{S}\,y \to x = y),$$
$$\Pi(X : \mathbb{N} \to \mathsf{Type}_0)\,(X\,0 \times \Pi(x : \mathbb{N})\,(X\,x \to X\,(\mathsf{S}\,x) \to \Pi(x : \mathbb{N})\,X\,x)).$$

We can not prove these axioms in $\lambda\mathsf{P2}$ (we can not construct terms of these types). In fact, Smith showed that there exists a model of the calculus of constructions where types in Type_0 are interpreted as sets with at most one element [Smi88]. Because we have defined the natural numbers in $\lambda\mathsf{P2}$ as type in Type_0 this means in particular that we can not prove the first axiom in $\lambda\mathsf{P2}$. Furthermore, Geuvers [Geu01] showed for any $N : \mathsf{Type}_0$ with terms $o : N$ and $s : N \to N$ that we can not construct a term of the type:

$$\Pi(X : N \to \mathsf{Type}_0)\,(X\,o \times \Pi(x : N)\,(X\,x \to X\,(s\,x)) \to \Pi(x : N)\,X\,x),$$

showing that the third axiom is not provable in $\lambda\mathsf{P2}$, and that we have no hope for an alternative definition of the natural numbers such that the third axiom is provable.

These limitations can be viewed as a feature rather than a bug. The calculus of constructions is a general framework that still leaves options open for the treatment of Type_0. One option is to treat the types in Type_0 as propositions, for instance by adding the $\mathsf{propext}$ rule we will discuss in the next paragraph. Another option are the ind-rules we will discuss on the next page.

Smiths model shows that the propositional extensionality rule is consistent:

$$\frac{\Gamma \vdash A : \mathsf{Type}_0 \qquad \Gamma \vdash B : \mathsf{Type}_0 \qquad \Gamma \vdash f : A \to B \qquad \Gamma \vdash g : A \to B}{\Gamma \vdash \mathsf{propext}\,f\,g : A =_{\mathsf{Type}_0} B}\ \text{propext.}$$

By adding this rule, we postulate that types in Type_0 are equal as soon as we can find functions between them. If we view the types in Type_0 as propositions, then this means that propositions are equal if they are equivalent.

Alternatively we can strengthen the elimination rules for the types we have introduced in Section 2.4. The elimination rules we have seen there only allow us to define functions, there is no way to define dependent functions. Recall for example the $\Sigma\mathcal{E}$-rule:

$$\frac{\Gamma \vdash C : \mathsf{Type}_0 \qquad \Gamma \vdash f : \Pi(x : A)\,(B(x) \to C) \qquad \Gamma \vdash p : \Sigma(x : A)\,B(x)}{\Gamma \vdash \mathsf{rec}_\Sigma\, C\, f\, p : C}\,\Sigma\mathcal{E}.$$

It allows us to define the left projection:

$$\begin{aligned}&\mathsf{pr}_0 : \quad \Sigma(x : A)\,B(x) \to A,\\ &\mathsf{pr}_0 \coloneqq \lambda(p : \Sigma(x : A)\,B(x))\,\mathsf{rec}_\Sigma\, A\,(\lambda(x : A)\,\lambda(y : B(x))\,x)\,p.\end{aligned}$$

However we can not define the right projection [Str92]. In order to define this projection we have to strengthen the elimination rule for small types to allow for dependent functions:

$$\frac{\Gamma, z : \Sigma(x : A)\,B(x) \vdash C(z) : \mathcal{C} \qquad \Gamma \vdash f : \Pi(x : A)\,\Pi(y : B(x))\,C(\langle x, y\rangle) \qquad \Gamma \vdash p : \Sigma(x : A)\,B(x)}{\Gamma \vdash \mathsf{ind}_\Sigma\, f\, p : C(p)}\,\Sigma\mathcal{E}'.$$

With this strengthened elimination rule we can define:

$$\begin{aligned}&\mathsf{pr}_1 : \quad \Pi(p : \Sigma(x : A)\,B(x))\,B(\mathsf{pr}_0\, p),\\ &\mathsf{pr}_1 \coloneqq \lambda(p : \Sigma(x : A)\,B(x))\,\mathsf{ind}_\Sigma\,(\lambda(x : A)\,\lambda(y : B(x))\,y)\,p.\end{aligned}$$

We can strengthen the elimination rules of the other types we have introduced in Section 2.4:

$$\frac{\Gamma, z : \mathbb{N} \vdash C(z) : \mathcal{C} \qquad \Gamma \vdash c : C(0) \qquad \Gamma \vdash f : \Pi(n : \mathbb{N})\,(C(n) \to C(\mathsf{S}\,n)) \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{ind}_\mathbb{N}\, c\, f\, n : C(n)}\,\mathbb{N}\mathcal{E}',$$

$$\frac{\Gamma, z : \mathbb{0} \vdash C(z) : \mathcal{C} \qquad \Gamma \vdash o : \mathbb{0}}{\Gamma \vdash \mathsf{ind}_\mathbb{0}\, o : C(o)}\,\mathbb{0}\mathcal{E}',$$

$$\frac{\Gamma, z : \mathbb{1} \vdash C(z) : \mathcal{C} \qquad \Gamma \vdash c : C(*) \qquad \Gamma \vdash i : \mathbb{1}}{\Gamma \vdash \mathsf{ind}_\mathbb{1}\, c\, i : C(i)}\,\mathbb{1}\mathcal{E}',$$

$$\frac{\Gamma, z : A + B \vdash C(z) : \mathcal{C} \qquad \Gamma \vdash f : \Pi(x : A)\,C(\mathsf{in}_0\, x) \qquad \Gamma \vdash g : \Pi(y : B)\,C(\mathsf{in}_1\, y) \qquad \Gamma \vdash t : A + B}{\Gamma \vdash \mathsf{ind}_+\, f\, g\, t : C(t)}\,+\mathcal{E}',$$

$$\frac{\Gamma, z : \Sigma(x : A)\,B(x) \vdash C(z) : \mathcal{C} \qquad \Gamma \vdash f : \Pi(x : A)\,\Pi(y : B(x))\,C(\langle x, y\rangle) \qquad \Gamma \vdash p : \Sigma(x : A)\,B(x)}{\Gamma \vdash \mathsf{ind}_\Sigma\, f\, p : C(p)}\,\Sigma\mathcal{E}'.$$

$$\frac{\Gamma, x : A, x' : A, z : x = x' \vdash C(x, x', z) : \mathcal{C} \qquad \Gamma \vdash c : C(a, a, \mathsf{refl}\, a) \qquad \Gamma \vdash e : a =_A a'}{\Gamma \vdash \mathsf{ind}_=\, c\, e : C(a, a', e)}\,=\mathcal{E}'.$$

The β-reduction rules for the various versions of ind are the same as those of rec. We use the language of the HoTT Book [Uni13] where they make the same distinction between recursion and induction.

We can use ind_Σ to prove the induction axiom from $\mathsf{HA2}$, it also proves the other two axioms:

- For the first axiom we first construct a predicate C on $\mathbb{N}$ such that $C\,0 \equiv_\beta \mathbb{0}$ and $C\,(\mathsf{S}\,n) \equiv_\beta \mathbb{1}$:

$$\begin{aligned}&C : \quad \mathbb{N} \to \mathsf{Type}_0,\\ &C \coloneqq \lambda(m : \mathbb{N})\,\mathsf{ind}_\mathbb{N}\,m\;\mathbb{0}\;(\lambda(n : \mathbb{N})\,\lambda(X : \mathsf{Type}_0)\,\mathbb{1}).\end{aligned}$$

  Now we have the following proof:

$$\begin{aligned}&\mathsf{ax}_0 : \quad \Pi(x : \mathbb{N})\,((\mathsf{S}\,x = 0) \to \mathbb{0}),\\ &\mathsf{ax}_0 \coloneqq \lambda(x : \mathbb{N})\,\lambda(e : \mathsf{S}\,x = 0)\,\mathsf{rec}_=\,C * e.\end{aligned}$$

- For the second axiom we first construct the predecessor function:

$$\begin{aligned}&\mathsf{P} : \quad \mathbb{N} \to \mathbb{N},\\ &\mathsf{P} \coloneqq \lambda n\,\mathsf{pr}_0(\mathsf{rec}_\mathbb{N}\,(\mathbb{N} \times \mathbb{N})\,(\langle 0, 0\rangle)\,(\lambda(p : \mathbb{N} \times \mathbb{N})\,\langle \mathsf{pr}_1\,p, \mathsf{S}(\mathsf{pr}_1\,p)\rangle)\,n).\end{aligned}$$

  Here we make clever use of pairs to remember the previous value.

  We also prove symmetry and transitivity for equality:

$$\begin{aligned}&\mathsf{sym} : \quad \Pi(A : \mathsf{Type}_0)\,\Pi(a : A)\,\Pi(a' : A)\,\Pi(e : a =_A a')\,(a' =_A a),\\ &\mathsf{sym} \coloneqq \lambda(A : \mathsf{Type}_0)\,\lambda(a : A)\,\lambda(a' : A)\,\lambda(e : a =_A a')\\ &\qquad \mathsf{rec}_=\,(\lambda(x : A)\,(x =_A a))\,(\mathsf{refl}\,A\,a)\,e,\end{aligned}$$

$$\begin{aligned}&\mathsf{trans} : \quad \Pi(A : \mathsf{Type}_0)\,\Pi(a : A)\,\Pi(a' : A)\,\Pi(a'' : A)\,\Pi(e : a =_A a')\,\Pi(e' : a' =_A a'')\,(a' =_A a)\\ &\mathsf{trans} \coloneqq \lambda(A : \mathsf{Type}_0)\,\lambda(a : A)\,\lambda(a' : A)\,\lambda(a'' : A)\,\lambda(e : a =_A a')\,\lambda(e' : a' =_A a''),\\ &\qquad \mathsf{rec}_=\,(\lambda(x : A)\,(a =_A x))\,e\,e'.\end{aligned}$$

  Now we have the following proofs:

$$\begin{aligned}&\mathsf{ap} : \quad \Pi(A : \mathsf{Type}_0)\,\Pi(B : \mathsf{Type}_0)\,\Pi(a : A)\,\Pi(a' : A)\,\Pi(f : A \to B)\,(a =_A a' \to f\,a =_B f\,a'),\\ &\mathsf{ap} \coloneqq \lambda(A : \mathsf{Type}_0)\,\lambda(B : \mathsf{Type}_0)\,\lambda(a : A)\,\lambda(a' : A)\,\lambda(f : A \to B)\,\lambda(e : a =_A a')\\ &\qquad \mathsf{rec}_=\,(\lambda(x : A)\,(f\,a =_B f\,x))\,(\mathsf{refl}\,B\,(f\,a))\,e,\end{aligned}$$

$$\begin{aligned}&\mathsf{ps} : \quad \Pi(n : \mathbb{N})\,(\mathsf{P}(\mathsf{S}\,n) =_\mathbb{N} n),\\ &\mathsf{ps} \coloneqq \lambda(n : \mathbb{N})\,\mathsf{ind}_\mathbb{N}\,(\mathsf{refl}\,0)\,(\lambda(n : \mathbb{N})\,\lambda(e : \mathsf{P}(\mathsf{S}\,n) =_\mathbb{N} n)\\ &\qquad \mathsf{ap}\,\mathbb{N}\,\mathbb{N}\,(\mathsf{P}(\mathsf{S}\,n))\,n\,\mathsf{S}\,e)\,n.\end{aligned}$$

  With this we can prove our axiom, it looks complicated but it is just combining the equality proofs we have already seen using the symmetry and transitivity of equality. If we have $\mathsf{S}\,x = \mathsf{S}\,y$ then we see $x = \mathsf{P}\,(\mathsf{S}\,x) = \mathsf{P}\,(\mathsf{S}\,y) = y$:

$$\begin{aligned}&\mathsf{ax}_1 : \quad \Pi(x : \mathbb{N})\,\Pi(y : \mathbb{N})\,(\mathsf{S}\,x = \mathsf{S}\,y \to x = y),\\ &\mathsf{ax}_1 \coloneqq \lambda(x : \mathbb{N})\,\lambda(y : \mathbb{N})\,\lambda(e : \mathsf{S}\,x = \mathsf{S}\,y)\\ &\qquad \mathsf{trans}\,\mathbb{N}\,x\,(\mathsf{P}\,(\mathsf{S}\,x))\,y\\ &\qquad\quad (\mathsf{sym}\,\mathbb{N}\,(\mathsf{ps}\,x))\\ &\qquad\quad (\mathsf{trans}\,\mathbb{N}\,(\mathsf{P}\,(\mathsf{S}\,x))\,(\mathsf{P}\,(\mathsf{S}\,y))\,y\\ &\qquad\qquad (\mathsf{ap}\,\mathbb{N}\,\mathbb{N}\,(\mathsf{S}\,x)\,(\mathsf{S}\,y)\,\mathsf{P}\,e)\\ &\qquad\qquad (\mathsf{ps}\,y)).\end{aligned}$$

So we have seen that there are two directions in which we can extend the treatment of Type_0: as propositions with propext, or as inductive types with the various versions of ind. In fact, we do not have to choose between these two options, the calculus of inductive constructions [PM15] has an infinite hierarchy of universes and replaces our lowest universe Type_0 with two lowest universes: Prop and Set. For Prop we can take propext and use notation from logic, while for Set we allow a general way to define inductive types with corresponding versions of ind. This is used to great effect in the proof assistants Coq [Coq89], and Lean [Mic13].

We end this section with two other axioms which are interesting both for the calculus of constructions, which we have discussed, and for Martin-Löf type theory. They influence the treatment of equality, which is an important topic in type theory. First we have function extensionality:

$$\frac{\Gamma \vdash f : \Pi(x : A)\, B(x) \qquad \Gamma \vdash f' : \Pi(x : A)\, B(x) \qquad \Gamma \vdash d : \Pi(x : A)\, (f\, x =_{B(x)} f'\, x)}{\Gamma \vdash \mathsf{funext}\, d : f =_{\Pi(x:A)B(x)} f'}\,\text{funext},$$

which says that functions are equal if they have the same behaviour. The other axiom is uniqueness of identity proofs [HS98] (equivalent to axiom K which was first introduced in [Str93]):

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash a' : A \qquad \Gamma \vdash e : a =_A a' \qquad \Gamma \vdash e' : a =_A a'}{\Gamma \vdash \mathsf{uip}\, e\, e' : e =_{(a =_A a')} e'}\,\text{uip}$$

We would be remiss not to mention the univalence axiom [Uni13], which is the basis for homotopy type theory. The precise statement of this axiom requires some more definitions, but it shows that bijective types are actually equal. This leads to a topological interpretation of type theory where proofs of equality are interpreted as paths. The univalence axiom proves funext and negates uip [Lic14; Uni13, Section 4.9].

Chapter 3

Conservatively Extending Arithmetic

In this chapter we will extend HA2 to allow us to construct a model of λP2 in arithmetic. Most notably, we will add lambda functions to our arithmetic. The intuition is that $\lambda x\, b(x)$ will be a natural number that encodes the partial recursive function (or Turing machine, or register machine, ...) that sends each number x to $b(x)$, while $f\, a$ will be the result of applying the partial recursive function encoded by f to a. This will give us a conservative extension HAP2 of HA2 because first-order arithmetic is already sufficient to do basic recursion theory. Then we will show that we can add a computational choice principle to HAP2, something that does not hold for partial recursive functions, and somehow still end up with a conservative extension HAP2ϵ of HA2.

Our approach is based on a paper by Benno van den Berg and Lotte van Slooten who use similar extensions of HA to prove Goodman's Theorem [vdBvS18]. Their methods carry over to second-order logic: we will construct second-order versions of the logic LPT and the theories HAP and HAPϵ.

Because we are dealing with partial recursive functions (or equivalent notions), function application will not be total. So in Section 3.0, we will first extend second-order logic to allow for partial function symbols. Then in Section 3.1, we extend HA2 with lambda functions, pairs, and recursion. Finally, in Section 3.2, we add a computation choice principle.

3.0 Second-order Logic of Partial Terms (LPT2)

We define LPT2, which is a second-order version of LPT: the logic of partial terms introduced by Beeson [Bee85, section VI.1]. In LPT and LPT2 we extend logic to permit partial function symbols, that is, we allow for terms which do not necessarily denote anything.

The terms of LPT2 are the same as those of second-order logic, they are given by:

$$a, b, c, \ldots ::= x \mid f^n(a_0, \ldots, a_{n-1}),$$

where f^n is a (partial) function symbol in our language. For every term a we have a new atomic formula $a{\downarrow}$ which is read as: 'a is defined' or 'a denotes'. For convenience, we define $a{\uparrow} := \neg(a{\downarrow})$.

The following inference rules deal with these new notions:

$$\frac{}{\Gamma \vdash x\downarrow}\downarrow\text{var}, \qquad \frac{\Gamma \vdash f^n(a_0,\ldots,a_{n-1})\downarrow}{\Gamma \vdash a_i\downarrow}\downarrow\text{fun}, \qquad \frac{\Gamma \vdash X^n(a_0,\ldots,a_{n-1})}{\Gamma \vdash a_i\downarrow}\downarrow\text{rel}.$$

The first rule states that all first-order variables denote: a variable should represent an actual object. The second rule shows that if a term denotes, that all its subterms denote. The third rule states that our second-order variables can only relate terms that denote.

The start and weak rules and the propositional rules remain the same as those in Section 1.0.

For first-order logic, we add a new requirement to the $\exists_1\mathcal{I}$ and $\forall_1\mathcal{E}$ rules:

$$\frac{\Gamma \vdash B(a) \qquad \Gamma \vdash a\downarrow}{\Gamma \vdash \exists x\, B(x)}\exists_1\mathcal{I}', \qquad z \text{ not free in } \Gamma, C\ \frac{\Gamma \vdash \exists x\, B(x) \qquad \Gamma, B(z) \vdash C}{\Gamma \vdash C}\exists_1\mathcal{E},$$

$$z \text{ not free in } \Gamma\ \frac{\Gamma \vdash B(z)}{\Gamma \vdash \forall x\, B(x)}\forall_1\mathcal{I}, \qquad \frac{\Gamma \vdash \forall x\, B(x) \qquad \Gamma \vdash a\downarrow}{\Gamma \vdash B(a)}\forall_1\mathcal{E}'.$$

These changes ensure that we only quantify over actual objects.

For second-order logic, we add a restriction to the $\exists_2\mathcal{I}$ and $\forall_2\mathcal{E}$ rules:

$$\downarrow \text{ not in } A(\vec{x})\ \frac{\Gamma \vdash B(A(\vec{x}))}{\Gamma \vdash \exists X^n\, B(X^n)}\exists_2\mathcal{I}', \qquad Z^n \text{ not free in } \Gamma, C\ \frac{\Gamma \vdash \exists X^n\, B(X^n) \qquad \Gamma, B(Z^n) \vdash C}{\Gamma \vdash C}\exists_2\mathcal{E},$$

$$Z^n \text{ not free in } \Gamma\ \frac{\Gamma \vdash B(Z^n)}{\Gamma \vdash \forall X^n\, B(X^n)}\forall_2\mathcal{I}, \qquad \downarrow \text{ not in } A(\vec{x})\ \frac{\Gamma \vdash \forall X^n\, B(X^n)}{\Gamma \vdash B(A(\vec{x}))}\forall_2\mathcal{E}'.$$

With '$\downarrow$ not in $A(\vec{x})$' we mean that $A(\vec{x})$ is a normal second-order formula, not involving $\downarrow$ (or $\uparrow$). This restriction is needed because second-order variables should only relate terms that denote. With the $\downarrow$rel rule we can prove $\forall X^1\,(X(a) \to a\downarrow)$; this is not true for arbitrary formulas, consider for example $A(x) \coloneqq x\uparrow$. The restrictions mean that we can no longer define the other logical connectives as easily as we have done in Section 1.1. Consider for example $\forall Z^0\, Z$, which we have used as a definition for $\bot$. We know that $\bot$ should imply every formula by the principle of explosion, however for $\forall Z^0\, Z$ we can only show this for formulas not involving $\downarrow$. We will sidestep this issue by considering all logical connectives as primitive.

For equality we require the terms to denote:

$$\frac{\Gamma \vdash a\downarrow}{\Gamma \vdash a = a}=\text{refl'}, \qquad \frac{\Gamma \vdash a = b}{\Gamma \vdash b = a}=\text{sym}, \qquad \frac{\Gamma \vdash a = b \qquad \Gamma \vdash b = c}{\Gamma \vdash a = c}=\text{trans},$$

$$\frac{\Gamma \vdash a = b \qquad \Gamma \vdash A(a)}{\Gamma \vdash A(b)}=\text{subst}, \qquad \frac{\Gamma \vdash a = b}{\Gamma \vdash a\downarrow \wedge b\downarrow}=\text{den}.$$

It is also useful to consider a weaker version of equality which also holds if both terms are undefined:

$$(a \simeq b) \coloneqq (a\downarrow \vee b\downarrow \to a = b).$$

For this version we can show that the following rules are valid:

$$\frac{\Gamma \vdash a \simeq b}{\Gamma \vdash b \simeq a}\simeq\text{sym}, \qquad \frac{\Gamma \vdash a \simeq b \qquad \Gamma \vdash b \simeq c}{\Gamma \vdash a \simeq c}\simeq\text{trans},$$

$$\frac{\Gamma \vdash a \simeq b \qquad \Gamma \vdash A(a)}{\Gamma \vdash A(b)}\simeq\text{subst}.$$

This concludes our introduction of LPT2. Note that we can view LPT2 is an extension of second-order logic, it is more expressive because we are now able to talk about function symbols which are not total. Every second-order theory (where we only have total function symbols) can also be formulated in LPT2; we just have to add the following axiom for every function symbol f^n:

$$\forall x_0 \ldots \forall x_{n-1}\, f^n(x_0, \ldots, x_{n-1})\!\downarrow.$$

For HA and HA2 this means that we add the axioms:

$$0\!\downarrow, \qquad \forall x\, \mathsf{S}(x)\!\downarrow, \qquad \forall x\, \forall y\, (x + y)\!\downarrow, \qquad \forall x\, \forall y\, (x \times y)\!\downarrow.$$

LPT2 is in some sense conservative over second-order logic. Recall that in a logic with only total function symbols, we can replace every function symbol f^n by a relation symbol F^{n+1} and an axiom:

$$\forall \vec{x}\, \exists! y\, F(\vec{x}, y),$$

see for instance [TvD98a, Section 2.7]. A similar thing is true in LPT and LPT2, we can replace every partial function symbol f^n by a relation symbol F^{n+1} and an axiom:

$$\forall \vec{x}\, !y\, F(\vec{x}, y).$$

Here ! means 'there is at most one' and is an abbreviation:

$$!x\, A(x) \coloneqq \forall x\, \forall x'\, (A(x) \wedge A(x') \to x = x').$$

In this way we can interpret every theory in LPT2 as a theory in second-order logic. However using LPT2 is more insightful, since it provides a convenient language to talk about partial function symbols and uses notation that is familiar from recursion theory.

3.1 Second-order Heyting Arithmetic of Partial Terms (HAP2)

In HAP2 we extend HA2 with a binary partial function symbol eval, we read $\mathsf{eval}(a, b)$ as 'evaluating the function encoded by a on the argument b'. Here we need the expressiveness of LPT2 because this application is not always guaranteed to produce a result. Instead of $\mathsf{eval}(a, b)$ we will follow our type theory notation and simply write $a\, b$. Like in type theory, we will write evaluation left associative, so $a\, b\, c$ means $(a\, b)\, c$. We also add constants (nullary function symbols) $\mathsf{k}, \mathsf{s}, \mathsf{suc}, \mathsf{pair}, \mathsf{pr}_0, \mathsf{pr}_1, \mathsf{rec}$ which stand for natural numbers encoding basic functions.

The constants k and s will be used to define lambda functions, they make the natural numbers into what is called a partial combinatory algebra [Bet88]. We view them as simple functions with the following behaviour:

$$\begin{aligned}
&\forall x\, \forall y\, \mathsf{k}\, x\, y = x,\\
&\forall x\, \forall y\, \mathsf{s}\, x\, y\!\downarrow,\\
&\forall x\, \forall y\, \forall z\, \mathsf{s}\, x\, y\, z \simeq x\, z\, (y\, z).
\end{aligned}$$

In addition, the constant suc encodes a function that computes the successor:

$$\forall x\, \mathsf{suc}\, x = \mathsf{S}(x).$$

We also have functions for pairs:

$$\begin{aligned}
&\forall x\, \forall y\, \mathsf{pr}_0\, (\mathsf{pair}\, x\, y) = x,\\
&\forall x\, \forall y\, \mathsf{pr}_1\, (\mathsf{pair}\, x\, y) = y,\\
&\forall x\, \mathsf{pair}\, (\mathsf{pr}_0\, x)\, (\mathsf{pr}_1\, x) = x.
\end{aligned}$$

And lastly, a function for recursion:

$$\forall x\,\forall y\,\mathsf{rec}\,x\,y\,0 = x,$$
$$\forall x\,\forall y\,\forall z\,\mathsf{rec}\,x\,y\,\mathsf{S}(z) \simeq y\,z\,(\mathsf{rec}\,x\,y\,z).$$

Note that we have not explicitly required the constants $\mathsf{k}, \mathsf{s}, \mathsf{suc}, \mathsf{pair}, \mathsf{pr}_0, \mathsf{pr}_1, \mathsf{rec}$ to denote, but that this is a consequence of our axioms. With the partial combinatory operators k and s we can define lambda abstraction in a simple recursive fashion:

$$\begin{aligned}
\lambda x\,x &:= \mathsf{s}\,\mathsf{k}\,\mathsf{k}, \\
\lambda x\,b &:= \mathsf{k}\,b, && \text{(if } b \text{ is a constant or variable other than } x\text{)} \\
\lambda x\,(b(x)\,b'(x)) &:= \mathsf{s}\,(\lambda x\,b(x))\,(\lambda x\,b'(x)), \\
\lambda x\,\mathsf{S}(b(x)) &:= \lambda x\,(\mathsf{suc}\,b(x)), \\
\lambda x\,(b(x) + b'(x)) &:= \lambda x\,(\mathsf{add}\,b(x)\,b'(x)), \\
\lambda x\,(b(x) \times b'(x)) &:= \lambda x\,(\mathsf{mul}\,b(x)\,b'(x)),
\end{aligned}$$

where add and mul are defined as follows:

$$\mathsf{add} := \lambda x\,\lambda y\,\mathsf{rec}\,y\,(\lambda i\,\lambda r\,\mathsf{suc}\,r)\,x,$$
$$\mathsf{mul} := \lambda x\,\lambda y\,\mathsf{rec}\,1\,(\lambda i\,\lambda r\,\mathsf{add}\,r\,y)\,x.$$

We can prove $(\lambda x\,b(x))\!\downarrow$ and $(\lambda x\,b(x))\,a \simeq b(a)$, see for instance [TvD98b, Theorem 9.3.5].

Furthermore, we can use pair to define arbitrary length tuples:

$$\langle a_0, \ldots, a_{n-1}\rangle := \mathsf{pair}\,a_0\,(\mathsf{pair}\,a_1\,(\ldots\,\mathsf{pair}\,a_{n-2}\,a_{n-1})).$$

Theorem 3.1.0. HAP2 is conservative over HA2.

Proof. This is a consequence of the fact that HAP is a conservative extension of HA [vdBvS18, Proposition 2.4]. More details for this proof can be found in [TvD98b, Proposition 9.3.12]. Because HAP and HAP2 have the same terms we see that HAP2 is a concervative extension of HA2. In particular, eval can be defined as Kleene application, that is, to calculate $a\,b$ we consider the partial recursive function encoded by a and evaluate it on input b. □

3.2 Adding Computational Choice (HAP2ϵ)

We define $\mathsf{HAP2}_\epsilon$ as the extension of HAP2 obtained by adding for each HA2-formula $\exists y\,A(\vec{x}, y)$ a constant $\epsilon_{\exists y A(\vec{x},y)}$ and the following axioms:

$$\forall \vec{x}\,(\exists y\,A(\vec{x}, y) \to \epsilon_{\exists y A(\vec{x},y)}\,\vec{x}\!\downarrow), \qquad \forall \vec{x}\,(\epsilon_{\exists y A(\vec{x},y)}\,\vec{x}\!\downarrow \to A(\vec{x}, \epsilon_{\exists y A(\vec{x},y)}\,\vec{x})).$$

This constant can be thought of as a code for a function that miraculously chooses a particular value y if we have $\exists y\,A(x, y)$ in HAP2ϵ. Note that, although we work in second-order logic, we still only add this for first-order quantifiers. This theory is conservative over HAP2 as we will prove in the remainder of this section. This is a bit surprising because such constants do not exist for the recursion theory we can already define in HA2. So HAP2ϵ is not be a definitional extension of HA2, but we can show that it is still a conservative extension. We start by adding a single relation symbol to HAP2.

Proposition 3.2.0. Suppose that $\exists y\,A(x, y)$ is a HAP2-formula, let HAP2F be the extension of HA2 with a relation symbol F^2 and the following axioms:

$$\forall x\,!y\,F(x, y), \qquad \forall x\,(\exists y\,A(x, y) \to \exists y\,F(x, y)), \qquad \forall x\,\forall y\,(F(x, y) \to A(x, y)).$$

So F is a 'partial choice function' for $A(x, y)$, that is, a partial function that sends a number x to a number y such that $A(x, y)$ holds, if such a y exists. Then $\mathsf{HAP2}_F$ is conservative over HAP2.

Proof. We prove this using forcing. We define a forcing condition p to be a finite approximation of the relation F. That is, p is an encoding of a finite set of pairs $\{\langle x_0, y_0\rangle, \dots, \langle x_{n-1}, y_{n-1}\rangle\}$ where the x_i are distinct and for every $i < n$ we have $A(x_i, y_i)$. We will not concern ourselves with the precise details of this encoding. One possible way to encode a finite set $\{x_0, \dots, x_{n-1}\}$ is as the tuple $\langle x_0, \dots, x_{n-1}\rangle$ (which is itself as a natural number). Within $\mathsf{HA2}$ we can now define notions like $x \in y$ and $x \subseteq y$ for this encoding.

Now we define the forcing relation. For a forcing condition p and an $\mathsf{HAP2}F$-formula A we define a $\mathsf{HAP2}$-formula $p \Vdash_F A$ with induction on A:

$$\begin{aligned}
p \Vdash_F A &:= A, && \text{(if } A \text{ is an atomic } \mathsf{HAP2}\text{-formula)}\\
p \Vdash_F F(x,y) &:= \forall(q \trianglerighteq p)\, \exists(r \trianglerighteq q)\, (\langle x, y\rangle \in r),\\
p \Vdash_F A \vee B &:= \forall(q \trianglerighteq p)\, \exists(r \trianglerighteq q)\, ((r \Vdash_F A) \wedge (r \Vdash_F B)),\\
p \Vdash_F A \wedge B &:= (p \Vdash_F A) \wedge (p \Vdash_F B),\\
p \Vdash_F A \to B &:= \forall(q \trianglerighteq p)\, ((q \Vdash_F A) \to (q \Vdash_F B)),\\
p \Vdash_F \exists x\, B(x) &:= \forall(q \trianglerighteq p)\, \exists(r \trianglerighteq q)\, \exists x\, (r \Vdash_F B(x)),\\
p \Vdash_F \forall x\, B(x) &:= \forall(q \trianglerighteq p)\, \forall x\, (q \Vdash_F B(x)),\\
p \Vdash_F \exists X^n\, B(X) &:= \forall(q \trianglerighteq p)\, \exists(r \trianglerighteq q)\, \exists X^n\, (r \Vdash_F B(X)),\\
p \Vdash_F \forall X^n\, B(X) &:= \forall(q \trianglerighteq p)\, \forall X^n\, (q \Vdash_F B(X)).
\end{aligned}$$

We can prove with induction for every $\mathsf{HAP2}F$-formula A that we have:

$$\begin{aligned}
&\mathsf{HAP2} \vdash \forall p\, \forall(q \trianglerighteq p)\, ((p \Vdash_F A) \to (q \Vdash_F A)),\\
&\mathsf{HAP2} \vdash \forall p\, (\forall(q \trianglerighteq p)\, \exists(r \trianglerighteq q)\, (r \Vdash_F A) \to (p \Vdash_F A)),
\end{aligned}$$

and with induction for every $\mathsf{HAP2}$-formula B that we have:

$$\mathsf{HAP2} \vdash \forall p\, ((p \Vdash_F B) \leftrightarrow B).$$

We can prove for every $\mathsf{HAP2}F$-formula A that $\mathsf{HAP2}F \vdash A$ implies $\mathsf{HAP2} \vdash \forall p\, (p \Vdash_F A)$ with induction on the proof of $\mathsf{HAP2}F \vdash A$. This is tedious but straightforward (see also the proof of [vdBvS18, Proposition 2.5] where they prove a similar statement).

This shows that $\mathsf{HAP2}F$ is conservative over $\mathsf{HAP2}$: suppose for a $\mathsf{HAP2}$-formula B that we have $\mathsf{HAP2}F \vdash B$, then we have $\mathsf{HAP2} \vdash \forall p\, (p \Vdash_F B)$ and therefore $\mathsf{HAP2} \vdash B$. □

Now we show that we can exchange this relation symbol for a function symbol.

Proposition 3.2.1. Suppose that $A(x, y)$ is a $\mathsf{HAP2}$-formula, let $\mathsf{HAP2}f$ be the extension of $\mathsf{HA2}$ with a function symbol f^1 and the following axioms:

$$\forall x\, \exists y\, (A(x,y) \to f(x)\!\downarrow), \qquad\qquad \forall x\, (f(x)\!\downarrow \to A(x, f(x))).$$

Then $\mathsf{HAP2}f$ is conservative over $\mathsf{HAP2}$.

Proof. This follows by reversing the remark we made at the end of Section 3.0: a relation symbol F^{n+1} and an axiom $\forall \vec{x}\, !y\, F(\vec{x}, y)$ can be replaced by a partial function symbol f^n. This is worked out in [TvD98a, Section 2.7]. □

And finally, that we can exchange this function symbol for a constant.

Proposition 3.2.2. Suppose that $A(x, y)$ is a HAP2-formula, let HAP2f be the extension of HA2 with a constant f and the following axioms:

$$\forall x(\exists y\, A(x, y) \to \mathsf{f}\, x{\downarrow}), \qquad \forall x(\mathsf{f}\, x{\downarrow} \to A(x, \mathsf{f}\, x)).$$

Then HAP2f is conservative over HA2. Remember that $\mathsf{f}\, x$ stands for $\mathsf{eval}(\mathsf{f}, x)$.

Proof. We work in HAP2f and use our existing evaluation $\mathsf{eval}(a, b)$ to define a new evaluation $\mathsf{eval}^f(a, b)$, which can use the function symbol f as an oracle. This is worked out in [vO06, Theorem 2.2].

The informal idea to calculate $\mathsf{eval}^f(a, b)$ is the following. We start by calculating $\mathsf{eval}(a, b)$. If this returns a value $\langle 0, x_0\rangle$ then that means that the function a wants to ask the oracle for the result of applying f to x_0. So we supply this value and run the function again, now we calculate $\mathsf{eval}(a, \langle b, f(x_0)\rangle)$. If this returns a value $\langle 0, x_1\rangle$ then the function a want another result from the oracle so we calculate $\mathsf{eval}(a, \langle b, f(x_0), f(x_1)\rangle)$. We keep doing this until a eventually returns a value $\langle 1, c\rangle$ in which case we say $\mathsf{eval}^f(a, b) = c$.

More formally, the formula $\mathsf{eval}^f(a, b) = c$ is true if there exists a sequence $\langle x_0, \ldots, x_{n-1}\rangle$ such that:

- for every $i < n$ we have $\mathsf{eval}(a, \langle b, f(x_0), \ldots, f(x_{i-1})\rangle) = \langle 0, x_i\rangle$;
- $\mathsf{eval}(a, \langle b, f(x_0), \ldots, f(x_{n-1})\rangle) = \langle 1, c\rangle$.

For this new evaluation we can define new combinators k^f and s^f which leads to a new lambda abstraction λ^f. We can also define new versions of the other constants: $\mathsf{suc}^f, \mathsf{pair}^f, \mathsf{pr}_0^f, \mathsf{pr}_1^f, \mathsf{rec}^f$. For the details, see [vO06, Theorem 2.2].

We can use this to show that HAP2f is conservative over HA. For any HAP2f-formula A we can relativize the evaluation and constants to f to get an HAP2f-formula A^f, we can prove with induction that we have HAP2f $\vdash A$ if and only if HAP2$f \vdash A^f$. For an HA2-formula B we see that B^f is the same as B so we see that HAP2f $\vdash B$ implies HAP2$f \vdash B$ which implies HA2 $\vdash B$. □

Now we are ready to prove the theorem.

Theorem 3.2.3. HAP2ϵ is conservative over HA2.

Proof. We think about the constants $\epsilon_{\exists y A(\vec{x}, y)}$ as choice functions (sending $\vec{x}$ to a value y with $A(\vec{x}, y)$ if such a value exists). We will show that every HA2-formula that can be proven using these functions can already be proven with a single one of these functions.

Suppose that we have HAP2$\epsilon \vdash A$ for an HA2-formula A. First, note that the proof for A can only use a finite amount of choice function, say $\epsilon_{\exists y\, A_i(\vec{x}, y)}$ for $i < n$. For every formula $A_i(\vec{x}, y)$ we can define a formula $B_i(x, y)$ such that $B_i(\langle\vec{x}\rangle, y)$ holds if and only if $A_i(\vec{x}, y)$. Now note that we can modify the proof of A to a proof that only uses a single choice function $\epsilon_{\exists y\, C(x,y)}$ where $C(x, y)$ is the formula:

$$C(x, y) \coloneqq \textstyle\bigwedge_{i<n}(\mathsf{pr}_0\, x = i \to B_i(\mathsf{pr}_1\, x, y)).$$

So the theorem follows from the previous proposition. □

Chapter 4

PER's and Arithmetical Assemblies

In this chapter we will construct a model for $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$, which can also be seen as an interpretation of $\lambda\mathsf{P2}$ in $\mathsf{HAP2}$. Here ind refers to all the versions of ind that we have introduced in Section 2.7 $(\mathbb{N}, \mathbb{0}, \mathbb{1}, +, \Sigma, =)$. Our construction is based on the model given by Reus [Reu99], who uses a well-known approach of interpreting small types as partial equivalence relations (PER's) and large types as assemblies due to Hyland [Hyl88]. We make two modifications to this model. Firstly, we restrict ourselves to what we call 'arithmetical' assemblies. This is a restriction on cardinality that makes sure that our model can be interpreted in second-order arithmetic. Secondly, we extend the model with special cases for the types we have introduced in Section 2.4. These special cases make sure that we satisfy the ind-rules, and make the interpretation of first-order formulas easier.

Throughout this chapter we will use general set-theoretic notation. This is because we have more established notation and intuition for sets, which makes sets easier to work with than formulas. All of these constructions will also work in $\mathsf{HAP2}$ through the following interpretation:

- with a set $A \subseteq \mathbb{N}$, we mean a formula $A(x)$;
- with a set $\mathcal{A} \subseteq \mathcal{P}(\mathbb{N})$, we mean a formula $\mathcal{A}(X^1)$;
- with a function $F : A \to B$ where $A, B \subseteq \mathbb{N}$, we mean a formula $F(x, y)$ such that:
  - for every x with $A(x)$ there exists a unique y with $B(y)$ such that $F(x, y)$,
  - for every x and y we have that $F(x, y)$ implies $A(x)$ and $B(y)$;
- with a function $\mathcal{F} : \mathcal{A} \to \mathcal{B}$ where $\mathcal{A}, \mathcal{B} \subseteq \mathcal{P}(\mathbb{N})$, we mean a formula $\mathcal{F}(X^1, Y^1)$ such that:
  - for every X^1 with $\mathcal{A}(X)$ there exists a unique Y^1 with $\mathcal{B}(Y)$ such that $\mathcal{F}(X, Y)$,
  - for every X^1 and Y^1 we have that $\mathcal{F}(X, Y)$ implies $\mathcal{A}(X)$ and $\mathcal{B}(Y)$;

We introduce PER's and assemblies in Section 4.0 and Section 4.1. In Section 4.2, we will start defining sums, products, and other notions which we will use in Section 4.3 to give our interpretation of type theory in arithmetic. In Section 4.4, we will discuss how sums work a bit differently in our model than in our type theory, and how we can deal with this. Finally, in Section 4.5, we prove our main theorem:

$\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ proves the same first-order arithmetical formulas as $\mathsf{HA2}$.

4.0 Partial Equivalence Relations (PER's)

We will interpret the small types of our type theory (the types in Type_0) as partial equivalence relations, these are given by the following definition.

Definition (PER)**.** A *partial equivalence relation (PER)* is a relation $R \subseteq \mathbb{N} \times \mathbb{N}$ that is symmetric and transitive. We define:

$$\begin{aligned} \mathrm{dom}(R) &\coloneqq \{\, n \in \mathbb{N} \mid \langle n, n \rangle \in R\} = \{n \in \mathbb{N} \mid \exists (m \in \mathbb{N})\, \langle n, m \rangle \in R\}, && \text{(domain)} \\ [n]_R &\coloneqq \{m \in \mathbb{N} \mid \langle n, m \rangle \in R\}, && \text{(equivalence class)} \\ \mathbb{N}/R &\coloneqq \{[n]_R \mid n \in \mathrm{dom}(R)\}. && \text{(quotient)} \end{aligned}$$

A *partial equivalence relation morphism (PER morphism)* from R to S is a function $F : \mathbb{N}/R \to \mathbb{N}/S$ that is 'tracked' by an $f \in \mathbb{N}$, meaning that $\langle n, m \rangle \in R$ implies $\langle f\, n, f\, m \rangle \in S$.

There are two ways to interpret $f\, n$ in this definition, depending on the theory in which we are working. In set theory we interpret $f\, n$ as Kleene application: $f\, n$ is the result of applying the partial recursive function encoded by f to n. Alternatively, when working in HAP2, we have that $f\, n$ is a primitive notion (the result of applying the binary partial function symbol eval to f and n).

The intuition of interpreting a small type $A : \mathsf{Type}_0$ as a PER R is that we consider the natural numbers as codes for the terms of A. We take $(m, n) \in R$ if and only if the natural numbers m and n encode the same term. It can happen that R is only a 'partial' equivalence relation because not every natural number has to encode a term, so we do not necessarily have reflexivity. A morphism between interpretations of types will be a function that is in some sense computable: it is tracked by a natural number sending codes to codes. As an example, we can already give the interpretations of the types $\mathbb{0}$, $\mathbb{1}$, and $\mathbb{N}$.

Definition $(\mathbf{0}, \mathbf{1}, \mathbf{N})$**.**

($\mathbf{0}$) There should be no terms of $\mathbb{0}$, so we let no natural number encode a term of $\mathbb{0}$; we get the empty partial equivalence relation $\mathbf{0} \coloneqq \emptyset$. For every PER R we see that there is a unique function $\mathbb{N}/\mathbf{0} \to \mathbb{N}/R$. This function is tracked by every natural number (there is nothing to track), so it is a morphism $\mathbf{0} \to R$. We conclude that $\mathbf{0}$ is the initial object in the category of PER's.

($\mathbf{1}$) Because $*$ is the only term of $\mathbb{1}$, we let every natural number encode $*$; this gives the total equivalence relation $\mathbf{1} \coloneqq \mathbb{N} \times \mathbb{N}$. For every PER R we see that there is a unique function $\mathbb{N}/R \to \mathbb{N}/\mathbf{1}$. This function is tracked by every natural number that encodes a total function, so it is a morphism $R \to \mathbf{1}$. We conclude that $\mathbf{1}$ is the final object in the category of PER's.

($\mathbf{N}$) We can just let every natural number encode itself; this gives the diagonal equivalence relation $\mathbf{N} \coloneqq \{(n, n) \mid n \in \mathbb{N}\}$. We see that there exists a function $O\colon \mathbb{N}/\mathbf{1} \to \mathbb{N}/\mathbf{N}$ given by $O(\mathbb{N}) \coloneqq \{0\}$. This function is tracked by $\lambda n\, 0 \in \mathbb{N}$, so it is a morphism $O : \mathbf{1} \to \mathbf{N}$. Similarly there exists a function $S\colon \mathbb{N}/\mathbf{N} \to \mathbb{N}/\mathbf{N}$ given by $S(\{n\}) \coloneqq \{\mathsf{S}(n)\}$. This function is tracked by $\lambda n\, \mathsf{S}(n) \in \mathbb{N}$, so it is a morphism $S : \mathbf{N} \to \mathbf{N}$. Now for every morphisms $A : \mathbf{1} \to R$ tracked by $a \in \mathbb{N}$, and $F : R \to R$ tracked by $f \in \mathbb{N}$ we can define a function $G : \mathbb{N}/\mathbf{N} \to \mathbb{N}/R$ by $G(\{n\}) \coloneqq F^n(A(\mathbb{N}))$. This function is tracked by $\mathsf{rec}\,(a\, 0)\,(\lambda i\, \lambda r\, f\, r) \in \mathbb{N}$, so it is a morphism $G : \mathbf{N} \to R$ and it is the unique morphism that makes the following diagram commute:

$$\begin{array}{ccccc} \mathbf{1} & \xrightarrow{O} & \mathbf{N} & \xrightarrow{S} & \mathbf{N} \\ {\scriptstyle \mathrm{id}_{\mathbf{1}}}\downarrow & & \downarrow{\scriptstyle G} & & \downarrow{\scriptstyle G} \\ \mathbf{1} & \xrightarrow[A]{} & R & \xrightarrow[F]{} & R \end{array}$$

We conclude that $\mathbf{N}$ is the natural numbers object in the category of PER's.

In general, the interpretation of a small type as a PER will depend on a context, as we will see later.

4.1 Arithmetical Assemblies

Now the large types will be interpreted as arithmetical assemblies.

Definition (assembly/$\mathbb{N}$-set)**.** An *assembly* consists of a set $\mathcal{A}$ and a relation $\Vdash_{\mathcal{A}} \subseteq \mathbb{N} \times \mathcal{A}$ such that for every $A \in \mathcal{A}$ there exists an $a \in \mathbb{N}$ such that $a \Vdash_{\mathcal{A}} A$. We call $\mathcal{A}$ the carrier and $\Vdash_{\mathcal{A}}$ the realizability relation; if we have $a \Vdash_{\mathcal{A}} A$ then we say a realizes A. An *assembly morphism* from $\mathcal{A}$ to $\mathcal{B}$ is a function $F : \mathcal{A} \to \mathcal{B}$ that is 'tracked' by an $f \in \mathbb{N}$, that is, for every $a \in \mathbb{N}$ and $A \in \mathcal{A}$ we have that $a \Vdash_{\mathcal{A}} A$ implies $f\, a \Vdash_{\mathcal{B}} F(A)$.

Because we are working towards an interpretation in HAP2, we will restrict ourselves to what we call 'arithmetical' assemblies, these are assemblies $\mathcal{A}$ where we have $\mathcal{A} \subseteq \mathcal{P}(\mathbb{N})$. This is where we deviate from the model given by Reus [Reu99], which uses general assemblies. The category of arithmetical assemblies is not as nice as the category of assemblies, most notably because it is not Cartesian closed. This means that our model will not be able to interpret the calculus of constructions completely: we can not interpret $\Pi(x : A)\, B(x)$ if A and $B(x)$ are both large (we can still talk about dependent morphisms in this case, just not about the assembly containing these dependent morphisms). If at least one of the two is small, however, then we are able to interpret this type as a PER or arithmetical assembly. So our model/interpretation does work for λP2.

Remark. The carrier of an assembly is completely determined by its realizability relation. This means that we could have simply defined an arithmetical assembly as a relation $\Vdash \subseteq \mathbb{N} \times \mathcal{P}(\mathbb{N})$ without explicitly mentioning the carrier. The reason that we have chosen for our presentation is familiarity. It is standard in the literature, and it lines up with notation for other categories like topologies, groups, and vector spaces whose objects also consist of a set with additional structure.

A partial equivalence relation R can be viewed as an arithmetical assembly by considering the set of equivalence classes $\mathbb{N}/R \subseteq \mathcal{P}(\mathbb{N})$ with realizability relation $\in$. This means that every natural number realizes its own equivalence class (if it has one). In this way the category of PER's induces a full subcategory of the category of arithmetical assemblies. We will write PER for the category of PER's, $\mathsf{PERAssem}$ for the induced full subcategory of arithmetical assemblies, and $\mathsf{ArithAssem}$ for the category of arithmetical assemblies. For $\mathcal{A} \in \mathsf{PERAssem}$ we write $\mathsf{Rel}(\mathcal{A})$ for the corresponding relation:

$$\mathsf{Rel}(\mathcal{A}) \coloneqq \{\langle a, a' \rangle \mid \exists (A \in \mathcal{A})\, (a \in A \wedge a' \in A)\}.$$

The arithmetical assemblies that are isomorphic to a PER assembly, are those where a natural number realizes at most one element of the assembly.

4.2 Type Theoretic Notions

Now that we have introduced PER's and arithmetical assemblies, we start by giving definitions that mirror our type theory concepts. In the next section we will use these definitions to give our model of type theory. First we develop some concepts in $\mathcal{P}(\mathbb{N})$. Recall for $a, b \in \mathbb{N}$ that the pair $\langle a, b \rangle$ is again a natural number. This gives us sums and products within $\mathcal{P}(\mathbb{N})$. For $A, B \in \mathcal{P}(\mathbb{N})$ we define:

$$\begin{aligned} A + B &\coloneqq \{\langle 0, a \rangle \in \mathbb{N} \mid a \in A\} \cup \{\langle 1, b \rangle \in \mathbb{N} \mid b \in B\} \in \mathcal{P}(\mathbb{N}), \\ A \times B &\coloneqq \{\langle a, b \rangle \in \mathbb{N} \mid a \in A \wedge b \in B\} \in \mathcal{P}(\mathbb{N}). \end{aligned}$$

Furthermore, for $A \in \mathcal{P}(\mathbb{N})$ and $B : A \to \mathcal{P}(\mathbb{N})$ we define:

$$\begin{aligned} \Sigma(a \in A)\, B(a) &\coloneqq \{\langle a, b \rangle \in \mathbb{N} \mid a \in A \wedge b \in B(a)\} \in \mathcal{P}(\mathbb{N}), \\ \Pi(a \in A)\, B(a) &\coloneqq \{f \in \mathbb{N} \mid \forall (a \in A)\, f\, a \in B(a)\} \in \mathcal{P}(\mathbb{N}). \end{aligned}$$

Note that every PER R is a subset of $\mathbb{N} \times \mathbb{N}$, so with this definition, we can also view it as an element of $\mathcal{P}(\mathbb{N})$. This means that we have $\mathsf{PER} \subseteq \mathcal{P}(\mathbb{N})$. For every $\mathcal{S} \subseteq \mathcal{P}(\mathbb{N})$ we will write $\nabla\mathcal{S}$ for the arithmetical assembly with carrier $\mathcal{S}$ and realizability relation $\mathbb{N} \times \mathcal{S}$. In particular we have the arithmetical assembly $\nabla\mathsf{PER}$. This arithmetical assembly will be very useful to us. We have that Type_0 is itself a large type, so we have to interpret it as an arithmetical assembly. The interpretation we will use for this is $\nabla\mathsf{PER}$.

Now we will define the small and large products of PER's and arithmetical assemblies. These will be the assemblies that contain the dependent morphisms. Here we first have to expand our notion of a morphism to that of a dependent morphism. If we have $\mathcal{A} \in \mathsf{ArithAssem}$ and a function $\mathcal{B} : \mathcal{A} \to \mathsf{ArithAssem}$, then a dependent morphism from $\mathcal{A}$ to $\mathcal{B}$ is a dependent function $F \in \Pi_{A \in \mathcal{A}} \mathcal{B}(A)$ that is tracked by an $f \in \mathbb{N}$, that is, for every $A \in \mathcal{A}$ and $a \Vdash_{\mathcal{A}} A$ we have $f\,a \Vdash_{\mathcal{B}(A)} F(A)$. Note that this definition also establishes what dependent morphisms between PER's are because we now view PER's as a special kind of arithmetical assembly. As we have already noted: for arbitrary $\mathcal{A} \in \mathsf{ArithAssem}$ and $\mathcal{B} : \mathcal{A} \to \mathsf{ArithAssem}$ we can not always construct an arithmetical assembly containing all dependent morphisms from $\mathcal{A}$ to $\mathcal{B}$. This is due to cardinality issues, suppose we have $\mathcal{A} \coloneqq \nabla\mathcal{P}(\mathbb{N})$ and for every $A \in \mathcal{A}$ we have $\mathcal{B}(A) \coloneqq \nabla\mathcal{P}(\mathbb{N})$. Then every dependent function from $\mathcal{A}$ to $\mathcal{B}$ is a dependent morphism (because it is trivially tracked), while:

$$|\Pi_{A \in \mathcal{A}} \mathcal{B}(A)| = |\mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})| \geq |\mathcal{P}(\mathcal{P}(\mathbb{N}))| > |\mathcal{P}(\mathbb{N})|,$$

so there is no way of encoding every dependent morphism from $\mathcal{A}$ to $\mathcal{B}$ as an element of $\mathcal{P}(\mathbb{N})$. We can however define two special cases which are enough to interpret λP2:

Definition (small product)**.** For $\mathcal{A} \in \mathsf{ArithAssem}$ and $\mathcal{B} : \mathcal{A} \to \mathsf{PERAssem}$ we define the small product:

$$\Pi_0(A \in \mathcal{A})\,\mathcal{B}(A) \coloneqq \mathbb{N}/\{\langle f, f' \rangle \in \mathbb{N} \mid \forall(A \in \mathcal{A})\,\forall(a, a' \Vdash_{\mathcal{A}} A)\,\langle f\,a, f'\,a' \rangle \in \mathsf{Rel}(\mathcal{B}(A))\} \in \mathsf{PERAssem}.$$

Note that this PER relates natural numbers when they encode the same dependent morphism from $\mathcal{A}$ to $\mathcal{B}$. So the equivalence classes will correspond to dependent morphisms; hence we can view the small product as the PER assembly whose objects encode dependent morphisms. We can introduce the analogies of lambda abstraction and function application. For a dependent morphism F from $\mathcal{A}$ to $\mathcal{B}$ we define $\lambda_0(A \in \mathcal{A})\,F(A)$ as the equivalence class in $\Pi_0(A \in \mathcal{A})\,\mathcal{B}(A)$ that corresponds to F:

$$\lambda_0(A \in \mathcal{A})\,F(A) \coloneqq \{f \in \mathbb{N} \mid \forall(A \in \mathcal{A})\,\forall(a \Vdash_{\mathcal{A}} A)\,f\,a \in F(A)\} \in \Pi_0(A \in \mathcal{A})\,\mathcal{B}(A).$$

We can also evaluate an element $G \in \Pi_0(A \in \mathcal{A})\,\mathcal{B}(A)$ as a dependent morphism on $A \in \mathcal{A}$:

$$\mathsf{Eval}_0(G, A) \coloneqq [g\,a] \in \mathcal{B}(A) \text{ where } g \in G \text{ and } a \Vdash_{\mathcal{A}} A.$$

Note that this is indeed well-defined.

Definition (large product)**.** For $\mathcal{A} \in \mathsf{PERAssem}$ and $\mathcal{B} : \mathcal{A} \to \mathsf{ArithAssem}$ we define the large product:

$$\begin{aligned}\Pi_1(A \in \mathcal{A})\,\mathcal{B}(A) \coloneqq \{\Sigma(a \in \cup\mathcal{A})\,B(a) \in \mathcal{P}(\mathbb{N}) \mid \exists(f \in \mathbb{N})\,\forall(a \in \cup\mathcal{A})\,f\,a \Vdash_{\mathcal{B}(A)} B(a) \wedge \\ \forall(\langle a, a' \rangle \in \mathsf{Rel}(\mathcal{A}))\,B(a) = B(a')\} \in \mathsf{ArithAssem},\end{aligned}$$
$$f \Vdash_{\Pi_1(A \in \mathcal{A})\mathcal{B}(A)} \Sigma(a \in \cup\mathcal{A})\,B(a) \text{ iff } \forall(a \in \cup\mathcal{A})\,f\,a \Vdash_{\mathcal{B}(A)} B(a).$$

Note that the elements of this arithmetical assembly are dependent morphisms from $\mathcal{A}$ to $\mathcal{B}$ encoded as an element of $\mathcal{P}(\mathbb{N})$, and realizers are natural numbers that track the morphism. For this encoding we can also give analogies of lambda abstraction and function application. For a dependent morphism F from $\mathcal{A}$ to $\mathcal{B}$ we define:

$$\lambda_1(A \in \mathcal{A})\,F(A) \coloneqq \Sigma(a \in \cup\mathcal{A})\,F([a]) \in \Pi_1(A \in \mathcal{A})\,\mathcal{B}(A).$$

For $G \in \Pi_1(A \in \mathcal{A})\,\mathcal{B}(A)$ and $A \in \mathcal{A}$ we define:

$$\mathsf{Eval}_1(G, A) \coloneqq \{b \in \mathbb{N} \mid \exists(a \in A)\,\langle a, b \rangle \in G\} \in \mathcal{B}(A).$$

With this we we have enough to interpret λP2, we will however continue to add special cases for the types introduced in Section 2.4 for two reasons, firstly because this allows us to simplify the interpretation of first-order formulas, and secondly because this allows us to add induction.

Definition (binary sum)**.** For $\mathcal{A} \in \mathsf{ArithAssem}$ and $\mathcal{B} \in \mathsf{ArithAssem}$ we define the binary sum:

$$\mathcal{A} + \mathcal{B} \coloneqq \{\mathsf{In}_0(A) \in \mathcal{P}(\mathbb{N}) \mid A \in \mathcal{A}\} \cup \{\mathsf{In}_1(B) \in \mathcal{P}(\mathbb{N}) \mid B \in \mathcal{B}\},$$
$$\langle 0, a\rangle \Vdash_{\mathcal{A}+\mathcal{B}} \mathsf{In}_0(A) \text{ iff } a \Vdash_{\mathcal{A}} A \qquad \langle 1, b\rangle \Vdash_{\mathcal{A}+\mathcal{B}} \mathsf{In}_1(B) \text{ iff } b \Vdash_{\mathcal{B}} B.$$

Here we define the injections for $A \in \mathcal{A}$ and $B \in \mathcal{B}$ as follows:

$$\mathsf{In}_0(A) \coloneqq \{\langle 0, a\rangle \mid a \in A\} \in \mathcal{A} + \mathcal{B},$$
$$\mathsf{In}_1(B) \coloneqq \{\langle 1, b\rangle \mid b \in B\} \in \mathcal{A} + \mathcal{B}.$$

Note that we have $\mathcal{A} + \mathcal{B} \in \mathsf{PERAssem}$ in the case that $\mathcal{A} \in \mathsf{PERAssem}$ and $\mathcal{B} \in \mathsf{PERAssem}$. If we have $\mathcal{C} : \mathcal{A} + \mathcal{B} \to \mathsf{ArithAssem}$, then for $F \in \Pi_u(A \in \mathcal{A})\, \mathcal{C}(\mathsf{In}_0(A))$, $G \in \Pi_u(B \in \mathcal{B})\, \mathcal{C}(\mathsf{In}_1(A))$, $T \in \mathcal{A} + \mathcal{B}$ realized by $f, g, t \in \mathbb{N}$, we define:

$$\mathsf{Ind}_+(F, G, T) \coloneqq \left\{ \begin{array}{l} \mathsf{Eval}_u(F, A) \text{ if } T = \mathsf{In}_0(A) \\ \mathsf{Eval}_u(G, B) \text{ if } T = \mathsf{In}_1(B) \end{array} \right\} \in \mathcal{C}(T).$$

This element of the arithmetical assembly $\mathcal{C}(T)$ is realized by:

$$\mathsf{ind}_+(f, g, t) \coloneqq \mathsf{rec}\,(\mathsf{pr}_0\, t)(f\,(\mathsf{pr}_1\, t))\,(\lambda i\, \lambda r\, g\,(\mathsf{pr}_1\, t)) \in \mathbb{N}.$$

Definition (sum)**.** For $\mathcal{A} \in \mathsf{ArithAssem}$ and $\mathcal{B} : \mathcal{A} \to \mathsf{ArithAssem}$ we define the sum:

$$\Sigma(A \in \mathcal{A})\, \mathcal{B}(A) \coloneqq \{A \times B \in \mathcal{P}(\mathbb{N}) \mid A \in \mathcal{A} \wedge B \in \mathcal{B}(A)\},$$
$$\langle a, b\rangle \Vdash_{\Sigma(A\in\mathcal{A})\mathcal{B}(A)} A \times B \text{ iff } a \Vdash_{\mathcal{A}} A \wedge b \Vdash_{\mathcal{B}(A)} B.$$

We define the projections for $P \in \Sigma(A \in \mathcal{A})\, \mathcal{B}(A)$ as follows:

$$\mathsf{Pr}_0(P) \coloneqq \{\mathsf{pr}_0\, p \mid p \in P\} \in \mathcal{A},$$
$$\mathsf{Pr}_1(P) \coloneqq \{\mathsf{pr}_1\, p \mid p \in P\} \in \mathcal{B}(\mathsf{Pr}_0(A)).$$

Note that we have $\Sigma(A \in \mathcal{A})\, \mathcal{B}(A) \in \mathsf{PERAssem}$ if $\mathcal{A} \in \mathsf{PERAssem}$ and $\mathcal{B}(A) \in \mathsf{PERAssem}$. If we have $\mathcal{C} : \Sigma(A \in \mathcal{A})\, \mathcal{B}(A) \to \mathsf{ArithAssem}$, then for $F \in \Pi_u(A \in \mathcal{A})\, \Pi_u(B \in \mathcal{B}(A))\, \mathcal{C}(A \times B)$, $P \in \Sigma(A \in \mathcal{A})\, \mathcal{B}(A)$ realized by $f, p \in \mathbb{N}$, we define:

$$\mathsf{Ind}_\Sigma(F, P) \coloneqq \mathsf{Eval}_u(\mathsf{Eval}_u(F, \mathsf{Pr}_0(P)), \mathsf{Pr}_1(P)) \in \mathcal{C}(P),$$

This element of the arithmetical assembly $\mathcal{C}(P)$ is realized by:

$$\mathsf{ind}_\Sigma(f, p) \coloneqq (f\,(\mathsf{pr}_0\, p))\,(\mathsf{pr}_1\, p) \in \mathbb{N}.$$

Definition (equality)**.** For $\mathcal{A} : \mathsf{ArithAssem}$ and $A, A' \in \mathcal{A}$ we define the equality assembly by:

$$\mathcal{E}(\mathcal{A}, A, A') \coloneqq \mathbb{N}/\{(e, e') \mid A = A'\}.$$

So we have $\mathcal{E}(\mathcal{A}, A, A') = \{\mathbb{N}\}$ if $A = A'$ and we have $\mathcal{E}(\mathcal{A}, A, A') = \emptyset$ if $A \neq A'$. If we have $\mathcal{C} : \Sigma(A \in \mathcal{A})\, \Sigma(A' \in \mathcal{A})\, (A = A') \to \mathsf{ArithAssem}$, then for every $C \in \mathcal{C}(A, A, \mathbb{N})$, $E \in \mathcal{E}(\mathcal{A}, A, A')$ realized by $c, e \in \mathbb{N}$, we define:

$$\mathsf{Ind}_{\mathcal{E}}(C, E) \coloneqq C \in \mathcal{C}(A, A', E).$$

This element of the arithmetical assembly $\mathcal{C}(A, A', E)$ is realized by:

$$\mathsf{ind}_{\mathcal{E}}(c, e) \coloneqq c \in \mathbb{N}.$$

We end this section by defining induction for $\mathbb{N}/\mathbf{N} = \{\{n\} \mid n \in \mathbb{N}\}$, which we have already introduced in Section 4.0. Defining induction for $\mathbb{N}/\mathbf{0}$ and $\mathbb{N}/\mathbf{1}$ is trivial so we leave it out here. If we have $\mathcal{C} : \mathbb{N}/\mathbf{N} \to \mathsf{ArithAssem}$, then for $C \in \mathcal{C}(\{0\}), F \in \Pi_u(\{n\} \in \mathbb{N}/\mathbf{N})\, \Pi_u(C \in \mathcal{C}(\{n\}))\, \mathcal{C}(\{\mathsf{S}(n)\}), \{n\} \in \mathbb{N}/\mathbf{N}$ realized by $c, f, n \in \mathbb{N}$, we define:

$$\mathsf{Ind}_{\mathbf{N}}(C, F, \{n\}) \coloneqq \mathsf{Eval}_u(\mathsf{Eval}_u(F, \{n-1\}), \dots \mathsf{Eval}_u(\mathsf{Eval}_u(F, \{0\}), C)) \in \mathcal{C}(\{n\}).$$

This element of the arithmetical assembly $\mathcal{C}(\{n\})$ is realized by

$$\mathsf{ind}_{\mathbf{N}}(c, f, n) \coloneqq \mathsf{rec}\, c\, f\, n \in \mathbb{N}.$$

Most concepts we have introduced here can be straightforwardly but laboriously translated to formulas as we explained at the start of the chapter. The one example we do want to explain further is $\mathsf{Ind}_{\mathbf{N}}(C, F, \{n\}) \subseteq \mathbb{N}$ (because it is the most difficult one), This set should be translated to a formula with a free variable x. The way we can define this formula is similar to the way we can introduce recursion theory in HA, we say that the formula holds for x iff there exists a product of sets $A_0 \times \cdots \times A_n$ such that:

- $A_0 = C$,
- for every $i < n$ we have $A_{i+1} = \mathsf{Eval}_u(\mathsf{Eval}_u(F, \{i\}), A_i)$,
- $x \in A_n$.

In this way we can develop a second-order version of recursion theory in $\mathsf{HA2}$.

4.3 Interpreting Type Theory in Arithmetic

Now we are finally ready to write out our interpretation of $\lambda\mathsf{P2}$ in $\mathsf{HA2}$. We will pick:

- for any well-formed context Γ an arithmetical assembly $\mathcal{G}[\![\Gamma]\!] \in \mathsf{ArithAssem}$,
- for any large type judgement $\Gamma \vdash A : \mathsf{Type}_1$, a function $\mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_1]\!] : \mathcal{G}[\![\Gamma]\!] \to \mathsf{ArithAssem}$,
- for any small type judgement $\Gamma \vdash A : \mathsf{Type}_0$, a function $\mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_0]\!] : \mathcal{G}[\![\Gamma]\!] \to \mathsf{PERAssem}$,
- For any term judgement $\Gamma \vdash a : A$ where $\Gamma \vdash A : \mathsf{Type}_i$, a dependent morphism $F[\![\Gamma \vdash a : A]\!]$ from $\mathcal{G}[\![\Gamma]\!]$ to $\mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_i]\!]$, and a natural number $f[\![\Gamma \vdash a : A]\!]$ that tracks $F[\![\Gamma \vdash a : A]\!]$.

Note that a judgement of the form $\Gamma \vdash A : \mathsf{Type}_0$ has multiple different interpretations because A is both a small type and a term of the large type Type_0:

- as a small type it is interpreted as a function $\mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_0]\!] : \mathcal{G}[\![\Gamma]\!] \to \mathsf{PERAssem}$,
- as a term it is interpreted as a dependent morphism $F[\![\Gamma \vdash A : \mathsf{Type}_0]\!]$ that goes from $\mathcal{G}[\![\Gamma]\!]$ to $\mathcal{A}[\![\Gamma \vdash \mathsf{Type}_0 : \mathsf{Type}_1]\!]$, and is tracked by natural number $f[\![\Gamma \vdash A : \mathsf{Type}_0]\!]$.

We have already done the legwork for this interpretation in Section 4.2 by defining sums, products, natural numbers, and equality for PER's and arithmetical assemblies. In this section we make the interpretation precise, we give our interpretation with a giant simultaneous induction on the derivation of the sequent.

For well-formed contexts we define:

$$\begin{aligned}\mathcal{G}[\![\,]\!] &\coloneqq \mathbb{N}/\mathbf{1},\\ \mathcal{G}[\![\Gamma, x : A]\!] &\coloneqq \Sigma(G \in \mathcal{G}[\![\Gamma]\!])\, \mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_u]\!](G).\end{aligned}$$

For the start and weak rules we define:

$$\begin{aligned}F[\![\Gamma, x : A \vdash x : A]\!] &\coloneqq \lambda G\, \mathsf{Pr}_1(G),\\ f[\![\Gamma, x : A \vdash x : A]\!] &\coloneqq \lambda g\ \mathsf{pr}_1\, g,\\ F[\![\Gamma, x : A \vdash b : B]\!] &\coloneqq \lambda G\, F[\![\Gamma \vdash b : B]\!](\mathsf{Pr}_0(G)),\\ f[\![\Gamma, x : A \vdash b : B]\!] &\coloneqq \lambda g\ f[\![\Gamma \vdash b : B]\!](\mathsf{pr}_0\, g).\end{aligned}$$

For our type universe we define:

$$\begin{aligned}\mathcal{A}[\![\Gamma \vdash \mathsf{Type}_0 : \mathsf{Type}_1]\!] &\coloneqq \lambda G\, \nabla\mathsf{PER},\\ F[\![\Gamma \vdash A : \mathsf{Type}_0]\!] &\coloneqq \lambda G\, \mathsf{Rel}(\mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_0]\!](G)),\\ f[\![\Gamma \vdash A : \mathsf{Type}_0]\!] &\coloneqq \lambda g\, 0.\end{aligned}$$

For the initial type we define:

$$\begin{aligned}\mathcal{A}[\![\Gamma \vdash \mathbb{0} : \mathsf{Type}_0]\!] &\coloneqq \lambda G\, \mathbb{N}/\mathbf{0},\\ F[\![\Gamma \vdash \mathsf{ind}_{\mathbb{0}}\, o : C(o)]\!] &\coloneqq \emptyset,\\ f[\![\Gamma \vdash \mathsf{ind}_{\mathbb{0}}\, o : C(o)]\!] &\coloneqq 0.\end{aligned}$$

For the final type we define:

$$\begin{aligned}\mathcal{A}[\![\Gamma \vdash \mathbb{1} : \mathsf{Type}_0]\!] &\coloneqq \lambda G\, \mathbb{N}/\mathbf{1},\\ F[\![\Gamma \vdash \mathsf{ind}_{\mathbb{1}}\, c\, i : C(i)]\!] &\coloneqq F[\![\Gamma \vdash c : C(*)]\!],\\ f[\![\Gamma \vdash \mathsf{ind}_{\mathbb{1}}\, c\, i : C(i)]\!] &\coloneqq f[\![\Gamma \vdash c : C(*)]\!].\end{aligned}$$

For the natural numbers we define:

$$\begin{aligned}\mathcal{A}[\![\Gamma \vdash \mathbb{N} : \mathsf{Type}_0]\!] &\coloneqq \lambda G\, \mathbb{N}/\mathbf{N},\\ F[\![\Gamma \vdash 0 : \mathbb{N}]\!] &\coloneqq \lambda G\, \{0\},\\ f[\![\Gamma \vdash 0 : \mathbb{N}]\!] &\coloneqq \lambda g\, 0,\\ F[\![\Gamma \vdash \mathsf{S}\, n : \mathbb{N}]\!] &\coloneqq \lambda G\, \{\mathsf{S}(n) \mid n \in F[\![\Gamma \vdash n : \mathbb{N}]\!](G)\},\\ f[\![\Gamma \vdash \mathsf{S}\, n : \mathbb{N}]\!] &\coloneqq \lambda g\, \mathsf{S}(f[\![\Gamma \vdash n : \mathbb{N}]\!]\, g),\\ F[\![\Gamma \vdash n + n' : \mathbb{N}]\!] &\coloneqq \lambda G\, \{n + n' \mid n \in F[\![\Gamma \vdash n : \mathbb{N}]\!](G) \wedge n' \in F[\![\Gamma \vdash n' : \mathbb{N}]\!](G)\},\\ f[\![\Gamma \vdash n + n' : \mathbb{N}]\!] &\coloneqq \lambda g\, (f[\![\Gamma \vdash n : \mathbb{N}]\!]\, g + f[\![\Gamma \vdash n' : \mathbb{N}]\!]\, g),\\ F[\![\Gamma \vdash n \times n' : \mathbb{N}]\!] &\coloneqq \lambda G\, \{n \times n' \mid n \in F[\![\Gamma \vdash n : \mathbb{N}]\!](G) \wedge n' \in F[\![\Gamma \vdash n' : \mathbb{N}]\!](G)\},\\ f[\![\Gamma \vdash n \times n' : \mathbb{N}]\!] &\coloneqq \lambda g\, (f[\![\Gamma \vdash n : \mathbb{N}]\!]\, g \times f[\![\Gamma \vdash n' : \mathbb{N}]\!]\, g),\\ F[\![\Gamma \vdash \mathsf{ind}_{\mathbb{N}}\, c\, f\, n : C(n)]\!] &\coloneqq \lambda G\, \mathsf{Ind}_{\mathbf{N}}(F[\![\Gamma \vdash c : C(0)]\!](G),\\ &\qquad F[\![\Gamma \vdash f : \Pi(n : \mathbb{N})\, (C(n) \to C(\mathsf{S}\, n))]\!](G),\\ &\qquad F[\![\Gamma \vdash n : \mathbb{N}]\!](G)),\\ f[\![\Gamma \vdash \mathsf{ind}_{\mathbb{N}}\, c\, f\, n : C(n)]\!] &\coloneqq \lambda g\ \mathsf{ind}_{\mathbf{N}}(f[\![\Gamma \vdash c : C(0)]\!]\, g,\\ &\qquad f[\![\Gamma \vdash f : \Pi(n : \mathbb{N})\, (C(n) \to C(\mathsf{S}\, n))]\!]\, g,\\ &\qquad f[\![\Gamma \vdash n : \mathbb{N}]\!]\, g).\end{aligned}$$

For the binary sum we define:

$$\begin{aligned}
\mathcal{A}[\![\Gamma \vdash A + B : \mathsf{Type}_0]\!] &\coloneqq \lambda G\, (\mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_0]\!](G) + \mathcal{A}[\![\Gamma \vdash B : \mathsf{Type}_0]\!](G)),\\
F[\![\Gamma \vdash \mathsf{in}_0\, a : A + B]\!] &\coloneqq \lambda G\, \mathsf{In}_0(F[\![\Gamma \vdash a : A]\!](G)),\\
f\,[\![\Gamma \vdash \mathsf{in}_0\, a : A + B]\!] &\coloneqq \lambda g\ \ \langle 0,\, f\,[\![\Gamma \vdash a : A]\!]\, g\rangle,\\
F[\![\Gamma \vdash \mathsf{in}_1\, b : A + B]\!] &\coloneqq \lambda G\, \mathsf{In}_1(F[\![\Gamma \vdash b : B]\!](G)),\\
f\,[\![\Gamma \vdash \mathsf{in}_1\, b : A + B]\!] &\coloneqq \lambda g\ \ \langle 0,\, f\,[\![\Gamma \vdash b : B]\!]\, g\rangle,\\
F[\![\Gamma \vdash \mathsf{ind}_+\, f\, g\, t : C(t)]\!] &\coloneqq \lambda G\, \mathsf{Ind}_+(F[\![\Gamma \vdash f : \Pi(a : A)\, C(\mathsf{in}_0\, a)]\!](G),\\
&\qquad F[\![\Gamma \vdash g : \Pi(b : B)\, C(\mathsf{in}_1\, b)]\!](G),\\
&\qquad F[\![\Gamma \vdash\ t : A + B]\!](G)),\\
f\,[\![\Gamma \vdash \mathsf{ind}_+\, f\, g\, t : C(t)]\!] &\coloneqq \lambda g\ \mathsf{ind}_+(f\,[\![\Gamma \vdash f : \Pi(a : A)\, C(\mathsf{in}_0\, a)]\!]\, g,\\
&\qquad f\,[\![\Gamma \vdash g : \Pi(b : B)\, C(\mathsf{in}_1\, b)]\!]\, g,\\
&\qquad f\,[\![\Gamma \vdash\ t : A + B]\!]\, g),
\end{aligned}$$

For the sum we define:

$$\begin{aligned}
\mathcal{A}[\![\Gamma \vdash \Sigma(x : A)\, B(x) : \mathsf{Type}_0]\!] &\coloneqq \lambda G\, \Sigma(X \in \mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_0]\!](G))\\
&\qquad \mathcal{A}[\![\Gamma, x : A \vdash B(x) : \mathsf{Type}_0]\!](G \times X),\\
F[\![\Gamma \vdash \langle a, b\rangle : \Sigma(x : A)\, B(x)]\!] &\coloneqq \lambda G\, F[\![\Gamma \vdash a : A]\!](G) \times\\
&\qquad F[\![\Gamma, a : A \vdash b : B(a)]\!](G \times F[\![\Gamma \vdash a : A]\!](G),\\
f\,[\![\Gamma \vdash \langle a, b\rangle : \Sigma(x : A)\, B(x)]\!] &\coloneqq \lambda g\, \langle f[\![\Gamma \vdash a : A]\!]\, g,\\
&\qquad f[\![\Gamma, a : A \vdash b : B(a)]\!]\, \langle g, f[\![\Gamma \vdash a : A]\!]\, g\rangle\rangle,\\
F[\![\Gamma \vdash \mathsf{ind}_\Sigma\, f\, p : C(p)]\!] &\coloneqq \lambda G\, \mathsf{Ind}_\Sigma(F[\![\Gamma \vdash f : \Pi(x : A)\, \Pi(y : B(x))\, C(\langle x, y\rangle)]\!](G),\\
&\qquad F[\![\Gamma \vdash p : \Sigma(x : A)\, B(x)]\!](G)),\\
f\,[\![\Gamma \vdash \mathsf{ind}_\Sigma\, f\, p : C(p)]\!] &\coloneqq \lambda g\ \mathsf{ind}_\Sigma(\, f\,[\![\Gamma \vdash f : \Pi(x : A)\, \Pi(y : B(x))\, C(\langle x, y\rangle)]\!]\, g,\\
&\qquad f\,[\![\Gamma \vdash p : \Sigma(x : A)\, B(x)]\!]\, g).
\end{aligned}$$

For the product we define:

$$\begin{aligned}
\mathcal{A}[\![\Gamma \vdash \Pi(x : A)\, B(x) : \mathsf{Type}_v]\!] &\coloneqq \lambda G\, \Pi_v(X \in \mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_u]\!](G))\\
&\qquad \mathcal{A}[\![\Gamma, x : A \vdash B(x) : \mathsf{Type}_0]\!](G \times X),\\
F[\![\Gamma \vdash \lambda(x : A)\, b(x) : \Pi(x : A)\, B(x)]\!] &\coloneqq \lambda G\ \lambda_v(X \in \mathcal{A}[\![\Gamma \vdash A : \mathsf{Type}_u]\!](G))\\
&\qquad F[\![\Gamma, x : A \vdash b(x) : B(x)]\!](G \times X),\\
f\,[\![\Gamma \vdash \lambda(x : A)\, b(x) : \Pi(x : A)\, B(x)]\!] &\coloneqq \lambda g\ \lambda x\\
&\qquad f[\![\Gamma, x : A \vdash b(x) : B(x)]\!]\, \langle g, x\rangle,\\
F[\![\Gamma \vdash f\, a : B(a)]\!] &\coloneqq \lambda G\, \mathsf{Eval}_v(F[\![\Gamma \vdash f : \Pi(x : A)\, B(x)]\!](G),\\
&\qquad F[\![\Gamma \vdash a : A]\!](G)),\\
f\,[\![\Gamma \vdash f\, a : B(a)]\!] &\coloneqq \lambda g\ (f[\![\Gamma \vdash f : \Pi(x : A)\, B(x)]\!]\, g)\\
&\qquad (f[\![\Gamma \vdash a : A]\!]\, g).
\end{aligned}$$

For equality we define:

$$\begin{aligned}
\mathcal{A}[\![\Gamma \vdash a =_A a' : \mathsf{Type}_0]\!] &\coloneqq \lambda G\, \mathcal{E}([\![\Gamma \vdash A : \mathsf{Type}_u]\!], F[\![\Gamma \vdash a : A]\!], F[\![\Gamma \vdash a' : A]\!]),\\
F[\![\Gamma \vdash \mathsf{refl}\, a : a =_A a\ ]\!] &\coloneqq \lambda G\, \mathbb{N},\\
f\,[\![\Gamma \vdash \mathsf{refl}\, a : a =_A a\ ]\!] &\coloneqq \lambda g\ 0,\\
F[\![\mathsf{ind}_=\, c\, e : C(a, a', e)]\!] &\coloneqq \lambda G\, \mathsf{Ind}_{\mathcal{E}}(F[\![\Gamma \vdash c : C(a, a, \mathsf{refl}\, a)]\!](G),\\
&\qquad\qquad F[\![\Gamma \vdash e : a =_A a']\!](G)),\\
f\,[\![\mathsf{ind}_=\, c\, e : C(a, a', e)]\!] &\coloneqq \lambda g\ \mathsf{ind}_{\mathcal{E}}(f\,[\![\Gamma \vdash c : C(a, a, \mathsf{refl}\, a)]\!]\, g,\\
&\qquad\qquad f\,[\![\Gamma \vdash e : a =_A a']\!]\, g).
\end{aligned}$$

Note that function extensionality and uniqueness of identity proofs follow easily because functions in λP2 are interpreted as morphism in HA2 while equality in λP2 is interpreted as an arithmetical assembly with at most one element, namely $\mathbb{N}$. Therefore, we can define:

$$\begin{aligned}
F[\![\Gamma \vdash \mathsf{funext}\, d : f =_{\Pi(x\in A)B(x)} f']\!] &\coloneqq \lambda G\, \mathbb{N},\\
f[\![\Gamma \vdash \mathsf{funext}\, d : f =_{\Pi(x\in A)B(x)} f']\!] &\coloneqq \lambda g\ 0,\\
F[\![\Gamma \vdash \mathsf{uip}\, e\, e' : e =_{(a=_A a')} e']\!] &\coloneqq \lambda G\, \mathbb{N},\\
f[\![\Gamma \vdash \mathsf{uip}\, e\, e' : e =_{(a=_A a')} e']\!] &\coloneqq \lambda g\ 0.
\end{aligned}$$

4.4 Size Differences

An important observation is that the size of sums behaves differently for our PER's and arithmetical assemblies than it does in λP2. First note that in both areas we have small and large objects:

- the small objects are the types in Type_0 and the PER's,
- the large objects are the types in Type_1 and the arithmetical assemblies.

Now consider the way we have originally defined sums in λP2:

$$\begin{aligned}
A + B &\coloneqq \Pi(Z : \mathsf{Type}_0)\,((A \to Z) \to (B \to Z) \to Z), && \text{(binary sum)}\\
\Sigma(x : A)\, B(x) &\coloneqq \Pi(Z : \mathsf{Type}_0)\,(\Pi(x : A)\ (B(x) \to Z) \to Z). && \text{(sum)}
\end{aligned}$$

Because Z is small and appears at the very right we see that the sums are always small, regardless of the sizes of A and B. In contrast, for our definitions for PER's and arithmetical assemblies we see that the sums are large as soon as either $\mathcal{A}$ or $\mathcal{B}$ is large:

$$\begin{aligned}
\mathcal{A} + \mathcal{B} &\coloneqq \{\mathsf{In}_0(A) \in \mathcal{P}(\mathbb{N}) \mid A \in \mathcal{A}\} \cup \{\mathsf{In}_1(B) \in \mathcal{P}(\mathbb{N}) \mid B \in \mathcal{B}\}, && \text{(binary sum)}\\
\langle 0, a\rangle \Vdash_{\mathcal{A}+\mathcal{B}} \mathsf{In}_0(A) &\text{ iff } a \Vdash_{\mathcal{A}} A \qquad \langle 1, b\rangle \Vdash_{\mathcal{A}+\mathcal{B}} \mathsf{In}_1(B) \text{ iff } b \Vdash_{\mathcal{B}} B,
\end{aligned}$$

$$\begin{aligned}
\Sigma(A \in \mathcal{A})\, \mathcal{B}(A) &\coloneqq \{A \times B \in \mathcal{P}(\mathbb{N}) \mid A \in \mathcal{A} \wedge B \in \mathcal{B}(A)\}, && \text{(sum)}\\
\langle a, b\rangle \Vdash_{\Sigma(A\in\mathcal{A})\mathcal{B}(A)} A \times B &\text{ iff } a \Vdash_{\mathcal{A}} A \wedge b \Vdash_{\mathcal{B}(A)} B.
\end{aligned}$$

There are multiple ways we can deal with this difference. We have chosen to only add the sums $A + B$ and $\Sigma(x : A)\, B$ as a primitive notion to λP2 if A and B are both small. So, in our interpretation, we only use the sums of PER's and arithmetical assemblies in the case where their size is small. Even when A and B are not small, we can still interpret the types

$$\Pi(Z : \mathsf{Type}_0)\,((A \to Z) \to (B \to Z) \to Z),$$
$$\Pi(Z : \mathsf{Type}_0)\,(\Pi(x : A)\,(B(x) \to Z) \to Z),$$

as PER's because the size of the product works the same for our PER's and arithmetical assemblies as it does in λP2.

Another option would be to add the sums $A + B$ and $\Sigma(x : A)\, B$ as a primitive notion to λP2 in every case, but with the same size behaviour as our sums for PER's and arithmetical assemblies: as soon as A or B is big we have that the sum is big. This has the benefit that we can also model/interpret ind if A or B is large. We have not chosen this approach because this behaviour is hard to justify without talking about the model first.

4.5 Conservativity

If A is a HA-formula with free variables $x_0, \ldots, x_{n-1}$ then we will write $z \Vdash_e A$ if z is a realizer for the PER that corresponds to A. In addition, we will write $z \approx z' \Vdash_e A$ if z and z' are both members of the same equivalence class of the PER that corresponds to A. More explicitly, we define:

$$(z \Vdash_e A) \coloneqq \langle z, z \rangle \in \mathsf{Rel}(\mathcal{A}[\![x_0 : \mathbb{N}, \ldots, x_{n-1} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle x_0, \ldots, x_{n-1}\rangle\})),$$
$$(z \approx z' \Vdash_e A) \coloneqq \langle z, z' \rangle \in \mathsf{Rel}(\mathcal{A}[\![x_0 : \mathbb{N}, \ldots, x_{n-1} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle x_0, \ldots, x_{n-1}\rangle\})).$$

This definition is not easy to work with. The following proposition shows that there exists an equivalent, inductive definition.

Proposition 4.5.0. In HAP2 we can prove the following bi-implications:

$$\begin{aligned}
z \Vdash_e a = a' &\leftrightarrow a = a',\\
z \approx z' \Vdash_e a = a' &\leftrightarrow a = a',\\
z \Vdash_e A \vee B &\leftrightarrow (\mathsf{pr}_0\, z = 0 \wedge \mathsf{pr}_1\, z \Vdash_e A) \vee\\
&\quad\ (\mathsf{pr}_0\, z = 1 \wedge \mathsf{pr}_1\, z \Vdash_e B),\\
z \approx z' \Vdash_e A \vee B &\leftrightarrow (\mathsf{pr}_0\, z = \mathsf{pr}_0\, z' = 0 \wedge \mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z' \Vdash_e A) \vee\\
&\quad\ (\mathsf{pr}_0\, z = \mathsf{pr}_0\, z' = 1 \wedge \mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z' \Vdash_e B),\\
z \Vdash_e A \wedge B &\leftrightarrow (\mathsf{pr}_0\, z \Vdash_e A) \wedge\\
&\quad\ (\mathsf{pr}_1\, z \Vdash_e B),\\
z \approx z' \Vdash_e A \wedge B &\leftrightarrow (\mathsf{pr}_0\, z \approx \mathsf{pr}_0\, z' \Vdash_e A) \wedge\\
&\quad\ (\mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z' \Vdash_e B),\\
z \Vdash_e A \to B &\leftrightarrow \forall w \forall w'\, ((w \approx w' \Vdash_e A) \to (z\, w \approx z\ w' \Vdash_e B)),\\
z \approx z' \Vdash_e A \to B &\leftrightarrow \forall w \forall w'\, ((w \approx w' \Vdash_e A) \to (z\, w \approx z'\, w' \Vdash_e B)),\\
z \Vdash_e \exists x\, B(x) &\leftrightarrow \mathsf{pr}_1\, z \Vdash_e B(\mathsf{pr}_0\, z),\\
z \approx z' \Vdash_e \exists x\, B(x) &\leftrightarrow (\mathsf{pr}_0\, z = \mathsf{pr}_0\, z') \wedge (\mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z' \Vdash_e B(\mathsf{pr}_0\, z)),\\
z \Vdash_e \forall x\, B(x) &\leftrightarrow \forall x\, (z\, x \Vdash_e B(x)),\\
z \approx z' \Vdash_e \forall x\, B(x) &\leftrightarrow \forall x\, (z\, x \approx z'\, x \Vdash_e B(x)).
\end{aligned}$$

Proof. To show this, we write out $\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\})$ for each of the cases for A.

For equality we have:

$$\begin{aligned}
&\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash (a = a')^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \mathcal{E}(\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash \mathbb{N} : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}), F[\![\vec{x} : \mathbb{N} \vdash a : \mathbb{N}]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}), F[\![\vec{x} : \mathbb{N} \vdash a' : \mathbb{N}]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\})) \\
&= \mathcal{E}(\mathbb{N}/\mathbf{N}, \{a\}, \{a'\}) \\
&= \mathbb{N}/\{(z, z') \mid a = a'\}.
\end{aligned}$$

For disjunction we have:

$$\begin{aligned}
&\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash (A \vee B)^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) + \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash B^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \mathbb{N}/\{(z, z') \mid (\mathsf{pr}_0\, z = \mathsf{pr}_0\, z' = 0 \wedge \mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z' \Vdash_e A) \vee (\mathsf{pr}_0\, z = \mathsf{pr}_0\, z' = 1 \wedge \mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z' \Vdash_e B)\}.
\end{aligned}$$

For conjunction we have:

$$\begin{aligned}
&\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash (A \wedge B)^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \Sigma(W \in \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}))\, \mathcal{A}[\![\vec{x} : \mathbb{N}, w : A^* \vdash B^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\} \times W) \\
&= \Sigma(W \in \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}))\, \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash B^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \times \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash B^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \mathbb{N}/\{(z, z') \mid (\mathsf{pr}_0\, z \approx \mathsf{pr}_0\, z' \Vdash_e A) \wedge (\mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z' \Vdash_e B)\}.
\end{aligned}$$

For implication we have:

$$\begin{aligned}
&\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash (A \to B)^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \Pi_0(W \in \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}))\, \mathcal{A}[\![\vec{x} : \mathbb{N}, w : A^* \vdash B^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\} \times W) \\
&= \Pi_0(W \in \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}))\, \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash B^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \to \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash B^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \mathbb{N}/\{(z, z') \mid \forall w\, \forall w'\, ((w \approx w' \Vdash_e A) \to (z\, w \approx z'\, w' \Vdash_e B)\}.
\end{aligned}$$

For the existential quantifier we have:

$$\begin{aligned}
&\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash (\exists x\, B(x))^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \Sigma(X \in \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash \mathbb{N} : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}))\, \mathcal{A}[\![\vec{x} : \mathbb{N}, x : \mathbb{N} \vdash B(x)^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\} \times X) \\
&= \Sigma(\{x\} \in \mathbf{N})\, \mathcal{A}[\![\vec{x} : \mathbb{N}, x : \mathbb{N} \vdash B(x)^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x}, x \rangle\}) \\
&= \mathbb{N}/\{(z, z') \mid (\mathsf{pr}_0\, z = \mathsf{pr}_0\, z') \wedge (\mathsf{pr}_1\, z \approx \mathsf{pr}_1\, z \Vdash_e B(x)^*)\}.
\end{aligned}$$

For the universal quantifier we have:

$$\begin{aligned}
&\mathcal{A}[\![\vec{x} : \mathbb{N} \vdash (\forall x\, B(x))^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\
&= \Pi_0(X \in \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash \mathbb{N} : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}))\, \mathcal{A}[\![\vec{x} : \mathbb{N}, x : \mathbb{N} \vdash B(x)^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\} \times X) \\
&= \Pi_0(\{x\} \in \mathbf{N})\, \mathcal{A}[\![\vec{x} : \mathbb{N}, x : \mathbb{N} \vdash B(x)^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x}, x \rangle\}) \\
&= \mathbb{N}/\{(z, z') \mid \forall x\, (z\, x \approx z'\, x \Vdash_e B)\}.
\end{aligned}$$

Now, if we consider the definitions of $z \Vdash_e A$ and $z \approx z' \Vdash_e A$, we see that the proposition follows. □

Theorem 4.5.1. For every HA-formula A we have: $\mathsf{HAP2}\epsilon \vdash A \leftrightarrow \exists z\,(z \Vdash_e A)$.

Proof. This theorem is almost identical to [vdBvS18, Theorem 3.5]. First, we can prove:

$$\mathsf{HAP2} \vdash A \vee B \leftrightarrow \exists i\,((i = 0 \to A) \wedge (i = 1 \to B)).$$

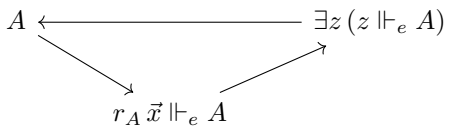
This allows us to rewrite every formula to a formula without disjunction. With the last proposition we also see that $A \vee B$ is realized if and only if $\exists i\,((i = 0 \to A) \wedge (i = 1 \to B))$ is realized:

$$\mathsf{HAP2} \vdash \exists z\,(z \Vdash A \vee B) \leftrightarrow \exists z\,(z \Vdash \exists i\,((i = 0 \to A) \wedge (i = 1 \to B))).$$

So it is sufficient to prove the theorem for every HA-formula without disjunction. For such a formula with free variables $\vec{x}$ we define a 'canonical realizer' inductively. This is the part where we need ϵ, to allow us to make choices for the existential quantifier:

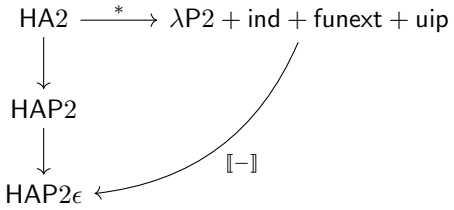
$$\begin{aligned}
r_{a=a'} &:= \lambda\vec{x}\,0,\\
r_{A\wedge B} &:= \lambda\vec{x}\,\langle r_A\,\vec{x}, r_B\,\vec{x}\rangle,\\
r_{A\to B} &:= \lambda\vec{x}\,\lambda w\,(r_B\,\vec{x}),\\
r_{\exists x\,B(x)} &:= \lambda\vec{x}\,\lambda x\,\langle \epsilon_{\exists x B(x)}\,\vec{x}, r_B\,\vec{x}\,(\epsilon_{\exists x B(x)}\,\vec{x})\rangle,\\
r_{\forall x\,B(x)} &:= \lambda\vec{x}\,\lambda x\,(r_B\,\vec{x}\,x).
\end{aligned}$$

Now using the last proposition and induction on A, we can prove the following implications in $\mathsf{HAP2}\epsilon$:



It follows that we have $\mathsf{HAP2}\epsilon \vdash A \leftrightarrow \exists z\,(z \Vdash_e A)$. □

This shows that the following diagram commutes for HA-formulas (up to logical equivalence):



With this we can prove our main theorem:

Theorem 4.5.2. For every HA-formula A we have:

$$\mathsf{HA2} \vdash A \qquad \text{iff} \qquad \lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip} \vdash A^*.$$

Here $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip} \vdash A^*$ means the following: if $\vec{x}$ are the free variables of A then there exists a term a such that we can derive $\vec{x} : \mathbb{N} \vdash a : A^*$ in $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$.

Proof. The direction from left to right follows easily. This is because the logical inference rules of second-order logic are valid in $\lambda\mathsf{P2}$ and because we have already seen that $\lambda\mathsf{P2} + \mathsf{ind}_{\mathbb{N}}$ proves the three axioms of $\mathsf{HA2}$ in Section 2.7.

For the direction from right to left, assume that we have a term a such that $\vec{x} : \mathbb{N} \vdash a : A^*$. Then our interpretation of $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ in $\mathsf{HAP2}$ gives us the following:

$$\begin{aligned} & f\,[\![\vec{x} : \mathbb{N} \vdash a : A^*]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\ \in\ & F[\![\vec{x} : \mathbb{N} \vdash a : A^*]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}) \\ \in\ & \mathcal{A}[\![\vec{x} : \mathbb{N} \vdash A^* : \mathsf{Type}_0]\!](\mathbb{N} \times \{\langle \vec{x} \rangle\}). \end{aligned}$$

So by our definition of $\Vdash_e$ we have $\mathsf{HAP2} \vdash \exists z\,(z \Vdash_e A)$ so $\mathsf{HAP2}\epsilon \vdash \exists z\,(z \Vdash_e A)$. Now by Theorem 4.5.1 we get $\mathsf{HAP2}\epsilon \vdash A$, and because $\mathsf{HAP2}\epsilon$ is a conservative extension of $\mathsf{HA2}$ we get $\mathsf{HA2} \vdash A$. □

This theorem also shows that $\mathsf{HA2}$ and $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ are equiconsistent. We can not extend this theorem to second-order formulas, this is shown in the following theorem:

Theorem 4.5.3. There exists an $\mathsf{HA2}$-formula A such that we have:

$$\mathsf{HA2} \nvdash A \qquad \text{but} \qquad \lambda\mathsf{P2} + \mathsf{ind}_\Sigma \vdash A^*.$$

Proof. The $\mathsf{HA2}$-formula we consider is the axiom of choice:

$$\forall Z^2\,(\forall x\,\exists y\,Z(x,y) \to \exists F^2\,(\forall x\,\exists! y\,F(x,y) \wedge \forall x\,\forall y\,(F(x,y) \to Z(x,y)))$$

Recall that $\exists!$ means 'there exists a unique' which is considered to be an abbreviation:

$$\exists! x\,A(x) \coloneqq \exists x\,(A(x) \wedge \forall x'\,(A(x') \to x = x')).$$

This means that the embedding of the axiom of choice in $\lambda\mathsf{P2}$ is:

$$\begin{aligned} & \Pi(Z : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}_0)\,\Pi(g : \Pi(x : \mathbb{N})\,\Sigma(y : \mathbb{N})\,Z\,x\,y) \\ & \Sigma(Z : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}_0)\,(\\ & \quad \Pi(x : \mathbb{N})\,\Sigma(y : \mathbb{N})\,(\\ & \quad\quad F\,x\,y \times \\ & \quad\quad \Pi(y' : \mathbb{N})(F\,x\,y' \to y = y')) \times \\ & \quad \Pi(x : \mathbb{N})\,\Pi(y : \mathbb{N})\,(F\,x\,y \to Z\,x\,y)), \end{aligned}$$

and this type is inhabited by the following closed term (which constitutes a proof):

$$\begin{aligned} & \lambda(Z : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}_0)\,\lambda(g : \Pi(x : \mathbb{N})\,\Sigma(y : \mathbb{N})\,Z\,x\,y) \\ & \langle \lambda(x : \mathbb{N})\,\lambda(y : \mathbb{N})\,(Z\,x\,y \times (\mathsf{pr}_0\,(g\,x) = y)),\ \langle \\ & \quad \lambda(x : \mathbb{N})\,\langle \mathsf{pr}_0\,(g\,x),\ \langle \\ & \quad\quad \langle \mathsf{pr}_1\,(g\,x), \mathsf{refl}\,(\mathsf{pr}_0\,(g\,x))\rangle, \\ & \quad\quad \lambda(y' : \mathbb{N})\,\lambda(f' : Z\,x\,y' \times (\mathsf{pr}_0\,(g\,x) = y'))\,\mathsf{pr}_1\,f'\rangle\rangle \\ & \quad \lambda(x : \mathbb{N})\,\lambda(y : \mathbb{N})\,\lambda(f : Z\,x\,y \times (\mathsf{pr}_0\,(g\,x) = y))\,\mathsf{pr}_0\,f\rangle\rangle. \end{aligned}$$

Note that we use pr_1 for which we need the ind_Σ axiom. So, because the axiom of choice is not provable in $\mathsf{HA2}$ [CR12] we have found a second-order formula that is provable in $\lambda\mathsf{P2} + \mathsf{ind}_\Sigma$ but not in $\mathsf{HA2}$. □

The obvious follow-up question is whether λP2 itself can already prove the axiom of choice. This is not true as shown by Streicher [Str92]. In fact, pr_1 is called the axiom of choice by some authors [BB96]. This still leaves us with an open question: can λP2 already prove a second-order formula that HA2 cannot prove? Our model that we have given here does not help us answer this question. This is because it satisfies ind_Σ, so it also satisfies the axiom of choice. We can therefore not use it to show that λP2 and HA2 satisfy the same second-order formulas.

We conclude by showing two versions of De Jongh Theorem for λP2. Using Theorem 4.5.2, we can translate Theorem 1.4.0 and Theorem 1.4.1 to get:

Corollary 4.5.4 (Σ^0_1 version of De Jongh's Theorem for $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$)**.** For every propositional formula A that is not provable constructively, there exists a substitution σ to HA2-formulas in Σ^0_1, such that we have $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip} \nvdash \sigma(A)^*$.

Corollary 4.5.5 (uniform Π^0_2 version of De Jongh's Theorem for $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$)**.** There exists a substitution σ to HA2-formulas in Π^0_2, such that for every propositional formula A that is not provable constructively, we have: $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip} \nvdash \sigma(A)^*$.

So the propositional logic of $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ is constructive. This means in particular that the propositional logic of λP2 is constructive.

Chapter 5

Conclusion

To conclude, let us summarise what we have proven about the relation between $\lambda\mathsf{P2}$ and $\mathsf{HA2}$:

- $\lambda\mathsf{P2}$ does not prove all arithmetical formulas that $\mathsf{HA2}$ proves. Notable examples are the axioms $\forall x\,(\mathsf{S}(x) \neq 0)$ and $\forall X^1\,(X(0) \wedge \forall x\,(X(x) \to X(\mathsf{S}(x))) \to \forall x\,X(x))$, see Section 2.7.
- $\lambda\mathsf{P2} + \mathsf{ind}_{\mathbb{N}}$ proves at least the same arithmetical formulas as $\mathsf{HA2}$, see Section 2.7.
- $\lambda\mathsf{P2} + \mathsf{ind}_{\Sigma}$ proves more second-order arithmetical formulas than $\mathsf{HA2}$. A notable example is the axiom of choice $\forall Z^2\,(\forall x\,\exists y\,Z(x,y) \to \exists F^2\,(\forall x\,\exists!y\,F(x,y) \wedge \forall x\,\forall y\,(F(x,y) \to Z(x,y)))$, see Theorem 4.5.3.
- $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ proves exactly the same first-order arithmetical formulas as $\mathsf{HA2}$, see Theorem 4.5.2.

As a consequence, we see that $\mathsf{HA2}$ and $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ are equiconsistent, and we can prove a version of De Jongh's theorem: the propositional logic of $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ is constructive.

This is an important improvement of our initial result for $\mathsf{ML0}$, which does not include type universes and inductive types. By proving that the propositional logic of $\lambda\mathsf{P2}$ is also constructive, we have shown that an impredicative universe and simple inductive types are compatible with a constructive logic.

This also leaves quite some room for future work. Firstly, we can look at stronger versions of type theory. One interesting question is: can we extend these results to the full calculus of constructions? This type theory corresponds to higher-order logic so it would be natural to investigate the relation with Higher-order Heyting Arithmetic [Hay80].

Another option would be to look at more general inductive types: our model works for some specific inductive types ($\mathbb{0}, \mathbb{1}, \mathbb{N}, +, \Sigma, =$ and their ind-extensions), so: can we extend our model and interpretation to more general inductive types? Type theories to consider are: Martin-Löf type theory [ML84] with its inductive W-types, and the calculus of inductive constructions [BC13; PM15] with its universe of inductive types. We can also look at coinductive types, for instance at M-types. These types can already be constructed using the natural numbers and function extensionality [AAG05; vdBDM07; ACS15]

Another direction would be to look at stronger versions of De Jongs's Theorem. A first option would be to look at a the first-order logic of $\mathsf{HA2}$. If the first-order logic of $\mathsf{HA2}$ is constructive then this would also carry over to $\lambda\mathsf{P2} + \mathsf{ind} + \mathsf{funext} + \mathsf{uip}$ with Theorem 4.5.2.

Bibliography

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

[ACS15] Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. Non-wellfounded trees in homotopy type theory. *Proceedings of TLCA 2015*, 38, 2015.

[Agd99] Programming Language group on Agda. *Agda*. `http://wiki.portal.chalmers.se/agda/`, 1999.

[Avr03] Arnon Avron. *Transitive Closure and the Mechanization of Mathematics*, pages 149–171. Springer Netherlands, Dordrecht, 2003.

[Bar91] Henk Barendregt. Introduction to generalized type systems. *Journal of functional programming*, 1(2):125–154, 1991.

[BB96] Franco Barbanera and Stefano Berardi. Proof-irrelevance out of excluded-middle and choice in the calculus of constructions. *Journal of Functional Programming*, 6(3):519–526, 1996.

[BC13] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development, Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[Bee85] Michael Beeson. *Foundations of Constructive Mathematics*. A Series of Modern Surveys in Mathematics. Springer, Berlin, 1985.

[Bet88] Ingemarie Bethke. *Notes on partial combinatory algebras*. PhD thesis, University of Amsterdam, 1988.

[Büc90] Julius Richard Büchi. *On a Decision Method in Restricted Second Order Arithmetic*, pages 425–435. Springer New York, New York, NY, 1990.

[Bur04] Wolfgang Burr. *The intuitionistic arithmetical hierarchy*, page 51–59. Lecture Notes in Logic. Cambridge University Press, 2004.

[Cai13] Andrés Caicedo. How is exponentiation defined in peano arithmetic? Mathematics Stack Exchange, 2013. URL: https://math.stackexchange.com/q/313049 (version: 2017-04-13).

[Can77] Georg Cantor. Ein beitrag zur mannigfaltigkeitslehre. *Journal für die reine und angewandte Mathematik*, 84:242–258, 1877.

[CH88] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2):95–120, 1988.

[Coq86] Thierry Coquand. *An analysis of Girard's paradox*. PhD thesis, INRIA, 1986.

[Coq89] Coq Team. *The Coq Proof Assistant*. `https://coq.inria.fr/`, 1989.

[CR12] Ray-Ming Chen and Michael Rathjen. Lifschitz realizability for intuitionistic zermelo–fraenkel set theory. *Archive for Mathematical Logic*, 51(7):789–818, 2012.

[Cro20] Laura Crosilla. Set Theory: Constructive and Intuitionistic ZF. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2020 edition, 2020.

[Cur80] Haskell Curry. Some philosophical aspects of combinatory logic. In Jon Barwise, Howard Jerome Keisler, and Kenneth Kunen, editors, *The Kleene Symposium*, volume 101 of *Studies in Logic and the Foundations of Mathematics*, pages 85–101. Elsevier, 1980.

[dG95] Philippe de Groote. *The Curry-Howard Isomorphism*. Academia, 1995.

[Dia75] Radu Diaconescu. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 51(1):176–178, 1975.

[dJ70] Dick de Jongh. The maximality of the intuitionistic predicate calculus with respect to heyting's arithmetic. *The Journal of Symbolic Logic*, 1970.

[dJS76] Dick de Jongh and Craig Smorynski. Kripke models and the intuitionistic theory of species. *Annals of Mathematical Logic*, 9(1):157, 1976.

[dJVV09] Dick de Jongh, Rineke Verbrugge, and Albert Visser. Intermediate logics and the de jongh property. *Logic Group Preprint Series*, 281:1–17, 2009.

[FŠ85] Harvey Friedman and Andrej Ščedrov. The lack of definable witnesses and provably recursive functions in intuitionistic set theories. *Advances in Mathematics*, 57(1):1–13, 1985.

[FŠ86] Harvey Friedman and Andrej Ščedrov. On the quantificational logic of intuitionistic set theory. *Mathematical proceedings of the Cambridge Philosophical Society*, 99(1):5–10, 1986.

[Geu94] Herman Geuvers. The calculus of constructions and higher order logic. In *The Curry-Howard isomophism*, pages 139–191. Katholieke Universiteit Leuven, 1994.

[Geu01] Herman Geuvers. Induction is not derivable in second order dependent type theory. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications*, pages 166–181, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Éditeur inconnu, 1972.

[Gir89] Jean-Yves Girard. *Proofs and types*, volume 7. Cambridge university press Cambridge, 1989.

[Hay80] Susumu Hayashi. Derived rules related to a constructive theory of metric spaces in intuitionistic higher order arithmetic without countable choice. *Annals of Mathematical Logic*, 19(1-2):33–65, 1980.

[HP17] Petr Hájek and Pavel Pudlák. *Metamathematics of first-order arithmetic*, volume 3. Cambridge University Press, 2017.

[HS98] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995)*, 36:83–111, 1998.

[Hur95] Antonius JC Hurkens. A simplification of girard's paradox. In *International Conference on Typed Lambda Calculi and Applications*, pages 266–278. Springer, 1995.

[Hyl88] J Martin E Hyland. A small complete category. *Annals of pure and applied logic*, 40(2):135–165, 1988.

[Kay91] Richard Kaye. *Models of Peano arithmetic*. Clarendon Press, 1991.

[Kle45] Stephen Cole Kleene. On the interpretation of intuitionistic number theory. *The journal of symbolic logic*, 10(4):109–124, 1945.

[Lei75] Daniel Leivant. *Absoluteness in intuitionistic logic*. PhD thesis, University of Amsterdam, 1975.

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

[LM91] Giuseppe Longo and Eugenio Moggi. Constructive natural deduction and its 'ω-set' interpretation. *Mathematical Structures in Computer Science*, 1(2):215–254, 1991.

[Mic13] Microsoft Research. *Lean Theorem Prover*. `https://leanprover.github.io/`, 2013.

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

[Pas20] Robert Passmann. De Jongh's Theorem for Intuitionistic Zermelo-Fraenkel Set Theory. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Pas22] Robert Passmann. The first-order logic of czf is intuitionistic first-order logic. *The Journal of Symbolic Logic*, page 1–23, 2022.

[PM15] Christine Paulin-Mohring. Introduction to the calculus of inductive constructions, 2015.

[Rat05] Michael Rathjen. The disjunction and related properties for constructive zermelo-fraenkel set theory. *The Journal of Symbolic Logic*, 70(4):1233–1254, 2005.

[Rat12] Michael Rathjen. From the weak to the strong existence property. *Annals of Pure and Applied Logic*, 163(10):1400–1418, 2012. Set Theory, Classical and Constructive – Invited papers from the meeting in Amsterdam, May 6–7, 2010.

[Reu99] Bernhard Reus. Realizability models for type theories. *Electronic Notes in Theoretical Computer Science*, 23(1):128–158, 1999. Tutorial Workshop on Realizability Semantics and Applications (associated to FLoC'99, the 1999 Federated Logic Conference).

[Smi88] Jan M. Smith. The independence of peano's fourth axiom from martin-löf's type theory without universes. *The Journal of Symbolic Logic*, 53(3):840–845, 1988.

[Str92] Thomas Streicher. Independence of the induction principle and the axiom of choice in the pure calculus of constructions. *Theoretical computer science*, 103(2):395–408, 1992.

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

[Swa14] Andrew W. Swan. Czf does not have the existence property. *Annals of Pure and Applied Logic*, 165(5):1115–1147, 2014.

[TvD98a] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics*, volume I. North-Holland Publishing Co., 1998.

[TvD98b] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics*, volume II. North-Holland Publishing Co., 1998.

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

[vdBDM07] Benno van den Berg and Federico De Marchi. Non-well-founded trees in categories. *Annals of Pure and Applied Logic*, 146(1):40–59, 2007.

[vdBvS18] Benno van den Berg and Lotte van Slooten. Arithmetical conservation results. *Indagationes Mathematicae*, 29:260–275, 2018.

[vO06] Jaap van Oosten. A general form of relative recursion. *Notre Dame Journal of Formal Logic*, 47(3):311–318, 2006.