On the treatment of predicative polymorphism in theories of explicit mathematics

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Mathis Kretz

2002

Leiter der Arbeit:

Prof. Dr. Gerhard Jäger,
Institut für Informatik und angewandte Mathematik

Contents

Introduction

Motivation

This thesis will study a way, in which predicative polymorphism can be treated in theories of explicit mathematics. The concept of polymorphism plays an important role in both computer science and mathematics. It is central to computer science, because many modern programming languages, especially in the functional and object oriented paradigms, exhibit some form of polymorphism in their type system. In programming languages polymorphism is used to enable data abstraction and factoring out of common behaviour across various components of a program. Such a mechanism is highly beneficial from an engineering point of view, as it lowers the cost of debugging, maintaining and modifying large software systems. In the object oriented paradigm, polymorphism is exploited not only on the implementation level of a software system, but also on the specification and design levels. Hence polymorphism is also reflected in object oriented modelling languages like, for example, UML. A good overview of the different flavours of polymorphism is given by Cardelli and Wegner [CW85].

In mathematics (and mathematical branches of computer science) polymorphism is mostly studied in the form of polymorphically typed λ-calculus. This extension of simply typed λ-calculus was introduced by Girard [GLT89], who termed it *System F*. Girard used the system to prove cut elimination for second order Peano arithmetic via a functional interpretation. However, the form of polymorphism used in System F is not without drawbacks. It is *impredicative* in the sense, that one may define types by referring to the collection of all types. Type variables appearing in a type expression σ may be instantiated with any type at all, so in particular with σ itself. This leads to the fact that systems based on impredicative polymorphism cannot have a straightforward set theoretic interpretation. That is to say, such a system cannot have a model, where terms are interpreted as set-theoretic functions and every type is interpreted as the set of all terms it contains. In this thesis, we will study two systems of *predicative* polymorphism, based on the work of Mitchell [Mit90, Mit96], where type variables range over a limited collection of types only. For such restricted forms of polymorphism, set-theoretic models are readily available.

We will show, how our two systems of predicative polymorphism may be embedded into two different theories of so-called explicit mathematics. Explicit mathematics was originally introduced by Feferman [Fef75, Fef79] as a formal framework for treating constructive mathematics. We will, however, be using a slight variation introduced by Jäger [Jäg88], which mainly differs from Feferman's original approach by the use of a naming relation on types. Explicit mathematics itself features untyped λ-abstraction, but not the typed analogue. It is therefore interesting in its own right, to study ways, in which such typing, particularly in the presence of polymorphism, may be simulated in an a priori type-free environment. Initial work in this direction was conducted by Feferman [Fef92, Fef90]. More recently Studer [Stu01] applied a similar method to predicative overloading. The main gain of an embedding of predicative polymorphism into explicit mathematics is, that the latter is well studied from a proof-theoretical perspective, mainly due to work by Feferman, Jäger and Strahm [Jäg88, FJ93, JS95] as well as Marzetta [Mar93].

Goals and scope

In this thesis, we shall essentially be doing the following three things:

1. Formally introduce the notion of predicative polymorphism. In doing so, we will consider a weaker and a slightly stronger variant.

2. Give an introduction to theories of explicit mathematics and find a particular theory, where the λ-abstraction mechanism has all the properties required for part 3.

3. Define and interpretation mapping of predicative polymorphism into explicit mathematics and show that the mapping constitutes a suitable embedding.

The third part will provide us with statements about the proof-theoretic strength of predicative polymorphism. In fact, we will obtain an exact correspondence in the case of the weaker variant and an upper bound for the strength of the stronger variant.

Acknowledgements

I am very grateful to Gerhard Jäger and Thomas Strahm, to the former for introducing me to theoretical computer science and logic and for expertly supervising my work on this thesis, to the latter for his constant readiness to help and for subtly pointing me in the right direction whenever I was lost. I would also like to thank all members of the research group for theoretical computer science and logic in Berne, especially Dieter Probst, Luca Alberucci, Geoffrey Ostrin and Marc Wirz, as well as Thomas Studer for many insightful discussions. Furthermore, my gratitude is due to my parents, Marion and Andreas Kretz-Lenz for the never-failing love and support they have given me over the years. Last, but by no means least, I would like to thank all my dear friends, (who are urged to stop reading after this section) for the great times we have together.

Chapter 1

The systems λ^p, λ^p_T and λ^p_{T+}

1.1 Overview

In the following we define three systems of polymorphically typed λ-calculus. First we define the base system λ^p of predicative polymorphism as given by Mitchell [Mit90, Mit96]. λ^p is a fragment of System F, as presented for example by Girard [GLT89]. It features a restricted form of polymorphism, which is achieved by splitting the types into two universes: the universe U_1 of "small" types and the universe U_2 of "large" types. The crucial feature of λ^p is, that variables in type expressions are taken to range over the small types only. We then extend the system λ^p to λ^p_T by adding some built-in constant types and constant terms, most notably a recursion scheme. The third system, called λ^p_{T+}, we obtain by adding an extra closure condition to the "large" types of λ^p_T. In the final section of this chapter, we prove some facts about the structure of the type universes in our calculi. These facts will become useful in a later chapter.

1.2 The λ^p-calculus

We first introduce the polymorphically typed λ^p-calculus or λ^p for short. To this end, we define the *preterms* of λ^p, as well as the *type expressions* of λ^p. After introducing the usual syntactic notions of *free variables* and *substitution*, we will provide a set of deduction rules, which govern the way, in which certain preterms are assigned types and become *well-typed* terms.

1.2.1 Preterms and type expressions

Definition 1.2.1 *We define the alphabet of λ^p to consist of the following symbols:*

1. *A countable set of individual variables denoted by $x, y, z, \ldots$,*

2. *a countable set of type variables denoted by $t, s, r, \ldots$,*

3. *the abstraction symbols* λ *and* Π,

4. *the universe symbols* U_1 *and* U_2,

5. *the equality symbol* $=$,

6. *the type arrow* $\rightarrow$,

7. *the strings* let *and* in,

8. *the typing symbol* $:$ *and*

9. *the delimiters* $.$, (*and*).

The set of individual variables and the set of type variables are taken to be disjoint. We speak simply of variables, when the distinction is irrelevant.

Definition 1.2.2 *We define the type expressions of* λ^p *to be those generated by the grammar*

$$\sigma ::= t \mid (\sigma \rightarrow \sigma) \mid \Pi t : U_1.\sigma,$$

where t *is a type variable. We shall be using the words "type" and "type expression" interchangeably in the context of* λ^p.

Definition 1.2.3 *We define the preterms of* λ^p *to be those generated by the grammar*

$$M ::= x \mid \lambda x : \sigma.M \mid MM \mid \lambda t : U_1.M \mid M\sigma \mid (\mathsf{let}\ x : \sigma = M\ \mathsf{in}\ M),$$

where x *is an individual variable,* t *a type variable and* σ *a type expression of* λ^p.

The inclusion of let in our calculus is somewhat unusual. As will become more apparent later, the intuitive reading of let is as an operator for explicit substitution. let will allow to substitute an arbitrary preterm for a variable in another preterm, without any universe constraints. For an example of this, see Remark 1.2.3.

1.2.2 Free variables and substitutions

Definition 1.2.4 *Let* σ *be a type expression of* λ^p. *We inductively define the set* $FV(\sigma)$ *of free type variables of* σ *as follows:*

1. $FV(\sigma) = \{t\}$, *if* σ *is the type variable* t.

2. $FV(\sigma) = FV(\tau) \cup FV(\xi)$, *if* σ *is the type expression* $\tau \rightarrow \xi$.

3. $FV(\sigma) = FV(\tau) \setminus \{t\}$, *if* σ *is the type expression* $\Pi t : U_1.\tau$.

We say that σ *is closed, if* $FV(\sigma) = \emptyset$.

Definition 1.2.5 *Let T be a preterm of λ^p. We inductively define the set $FV(T)$ of free variables of T as follows:*

1. $FV(T) = \{x\}$, *if T is a variable x.*
2. $FV(T) = (FV(M) \setminus \{x\}) \cup FV(\sigma)$, *if T is the preterm $\lambda x : \sigma.M$.*
3. $FV(T) = FV(M) \cup FV(N)$, *if T is the preterm MN.*
4. $FV(T) = FV(M) \setminus \{t\}$, *if T is the preterm $\lambda t : U_1.M$.*
5. $FV(T) = FV(M) \cup FV(\sigma)$, *if T is the preterm $M\sigma$.*
6. $FV(T) = FV(\sigma) \cup FV(N) \cup (FV(M) \setminus \{x\})$, *if T is the preterm* $(\mathsf{let}\ x : \sigma = N\ \mathsf{in}\ M)$.

We say, that T is closed if $FV(T) = \emptyset$.

The following definitions of substitution for type expressions and terms are somewhat lengthy. This is due to the fact, that we must rename bound variables, in the case, where unwanted binding of free variables would take place.

Definition 1.2.6 *Let A stand for a type expression or a preterm, v for a (type or individual) variable of λ^p and let σ and ξ be type expressions of λ^p. The type expression $[A/v]\,\sigma$, which results from substituting A for v in σ, we inductively define as follows:*

1. $[A/v]\,v :\equiv A$.
2. $[A/v]\,s :\equiv s$, *if s is a variable, distinct from v.*
3. $[A/v]\,(\sigma \to \xi) :\equiv ([A/v]\,\sigma) \to ([A/v]\,\xi)$.
4. $[A/v]\,(\Pi v : U_1.\sigma) :\equiv \Pi v : U_1.\sigma$.
5. $[A/v]\,(\Pi t : U_1.\sigma) :\equiv \Pi t : U_1.([A/v]\,\sigma)$, *if t is a type variable, distinct from v and $t \notin FV(A)$.*
6. $[A/v]\,(\Pi t : U_1.\sigma) :\equiv \Pi s : U_1.([A/v]\,([s/t]\,\sigma))$, *where s is a type variable, distinct from v, such that $s \notin FV(A) \cup FV(\sigma)$, if t is a type variable, distinct from v and $t \in FV(A)$*

Remark 1.2.1 *Note that, if v is an individual variable, then Definition 1.2.6 implies $[A/v]\,\sigma = \sigma$ for any type expression σ of λ^p.*

Definition 1.2.7 *Let A stand for a preterm or a type expression of λ^p, v stand for a (type or individual) variable of λ^p and let T, P and Q be preterms of λ^p. The preterm $[A/v]\,T$, which results from substituting A for v in T, we inductively define as follows:*

1. $[A/v]\,v :\equiv A$.

2. $[A/v]\, w :\equiv w$, *if* w *is a variable, distinct from* v.

3. $[A/v]\,(TC) :\equiv ([A/v]\, T)([A/v]\, C)$,*where* C *stands for a preterm or type expression.*

4. $[A/v]\,(\lambda v : \sigma.P) :\equiv \lambda v : \sigma.P$.

5. $[A/v]\,(\lambda x : \sigma.P) :\equiv \lambda x : ([A/v]\,\sigma).([A/v]\, P)$, *if* x *is a variable, distinct from* v *and* $x \notin FV(A)$.

6. $[A/v]\,(\lambda x : \sigma.P) :\equiv \lambda y : \sigma.([A/v]\,([y/x]\, P))$, *where* y *is a variable, distinct from* v, *such that* $y \notin FV(P) \cup FV(A)$, *if* x *is a variable, distinct from* v *and* $x \in FV(A)$.

7. $[A/v]\,(\mathsf{let}\ v : \sigma = P\ \mathsf{in}\ Q) :\equiv (\mathsf{let}\ v : [A/v]\,\sigma = [A/v]\, P\ \mathsf{in}\ Q)$.

8. $[A/v]\,(\mathsf{let}\ x : \sigma = P\ \mathsf{in}\ Q) :\equiv (\mathsf{let}\ x : [A/v]\,\sigma = ([A/v]\, P)\ \mathsf{in}\ ([A/v]\, Q))$, *if* x *is a variable, distinct from* v.

9. $[A/v]\,(\lambda v : U_1.P) :\equiv \lambda v : U_1.P$.

10. $[A/v]\,(\lambda t : U_1.P) :\equiv \lambda t : U_1.([A/v]\, P)$, *if* t *is a variable, distinct from* v *and* $t \notin FV(A)$.

11. $[A/v]\,(\lambda t : U_1.P) :\equiv \lambda s : U_1.([A/v]\,([s/t]\, P))$, *where* s *is a variable, distinct from* v, *such that* $s \notin FV(A) \cup FV(P)$, *if* t *is a variable, distinct from* v *and* $t \in FV(A)$.

Remark 1.2.2 *Note that, if* v *is a type variable, then for* $[A/v]\,(\lambda x : \sigma.P)$ *case 5 of Definition 1.2.7 applies. Similarly, if* v *is an individual variable, then case 10 applies for* $[A/v]\,(\lambda t : U_1.P)$.

1.2.3 The rules of λ^p

We now introduce the rules of λ^p. They are to be understood as a simultaneous definition of the universes U_1 and U_2, the well-typed terms of λ^p and the behaviour of the symbol $=$. All rules work with respect to a *context*, in which the free variables of a term are given a type. The first group of rules state, how such contexts are built. The second group of rules will be concerned with the actual typing of terms and the third group will provide us with a notion of when two typed terms are to be considered equal.

Context axioms and rules

Formally, a context Γ is a finite ordered sequence

$$\Gamma = (v_1, A_1), \ldots, (v_k, A_k)$$

of pairs (v_i, A_i), assigning to each variable v_i, where $1 \leq i \leq k$, a type in case v_i is an individual variable, or one of the universes U_1 or U_2 in case v_i is a type variable. We will write $v : A$ for a pair (v, A) belonging to Γ. Furthermore, we will write Γ_1, Γ_2 for the sequence which results from appending Γ_2 to the tail of Γ_1, where Γ_1 and Γ_2 are sequences. The following rules state, under what circumstances a sequence is a valid context:

(empty context) $$\emptyset \ \mathit{context}$$

(U_1 context) $$\frac{\Gamma \ \mathit{context}}{\Gamma, t : U_1 \ \mathit{context}} \ t \text{ not in } \Gamma$$

(U_i type context) $$\frac{\Gamma \rhd \sigma : U_i}{\Gamma, x : \sigma \ \mathit{context}} \ x \text{ not in } \Gamma$$

The only context axiom (empty context) states, that the empty sequence is a valid context. The rules (U_1 context) and (U_i type context) are used for introducing new typing assumptions about variables into a context. The side condition in both of rules ensures, that each variable is assigned at most one universe or type in a context.

The next group of rules serve to type preterms of λ^p. On one hand the type expressions need to be structured into the two universes U_1 and U_2. The universe rules take care of this aspect. On the other hand, we want certain terms to recieve types, which is the purpose of the term typing rules.

Universe rules

The following rules state, when a type expression belongs to U_1 and when it belongs U_2. Thus, given a type expression σ of λ^p, the judgement $\Gamma \rhd \sigma : U_i$ is to be read as "σ belongs to the universe U_i in context Γ".

($\rightarrow U_1$) $$\frac{\Gamma \rhd \tau : U_1 \quad \Gamma \rhd \sigma : U_1}{\Gamma \rhd \tau \rightarrow \sigma : U_1}$$

($U_1 \subseteq U_2$) $$\frac{\Gamma \rhd \tau : U_1}{\Gamma \rhd \tau : U_2}$$

($\Pi\ U_2$) $$\frac{\Gamma, t : U_1 \rhd \sigma : U_2}{\Gamma \rhd (\Pi t : U_1.\sigma) : U_2}$$

The rule ($\rightarrow U_1$) closes the universe U_1 under arrow types. ($U_1 \subseteq U_2$) states, that every type expression in U_1 is also belongs to U_2. Furthermore, the universe U_2 is closed under type abstraction by the rule ($\Pi\ U_2$).

Term typing rules

The term typing rules assign types to certain preterms. Thus, given a preterm T and a type σ of λ^p, the judgement $\Gamma \rhd T : \sigma$ is to be read as "T is of type σ in context Γ". The rules are as follows:

(var) $$\frac{\Gamma, x : A \ \textit{context}}{\Gamma, x : A \rhd x : A}$$

(add var) $$\frac{\Gamma \rhd A : B \quad \Gamma, x : C \ \textit{context}}{\Gamma, x : C \rhd A : B}$$

($\rightarrow$ Intro) $$\frac{\Gamma, x : \tau \rhd M : \tau' \quad \Gamma \rhd \tau : U_1 \quad \Gamma \rhd \tau' : U_1}{\Gamma \rhd (\lambda x : \tau . M) : \tau \rightarrow \tau'}$$

($\rightarrow$ Elim) $$\frac{\Gamma \rhd M : \tau \rightarrow \tau' \quad \Gamma \rhd N : \tau}{\Gamma \rhd MN : \tau'}$$

(Π Intro) $$\frac{\Gamma, t : U_1 \rhd M : \sigma}{\Gamma \rhd (\lambda t : U_1 . M) : \Pi t : U_1 . \sigma}$$

(Π Elim) $$\frac{\Gamma \rhd M : \Pi t : U_1 . \sigma \quad \Gamma \rhd \tau : U_1}{\Gamma \rhd M\tau : [\tau / t]\, \sigma}$$

(let) $$\frac{\Gamma \rhd \tau : U_1 \quad \Gamma, x : \sigma \rhd M : \tau \quad \Gamma \rhd N : \sigma}{\Gamma \rhd (\mathsf{let}\ x : \sigma = N\ \mathsf{in}\ M) : \tau}$$

In both (var) and (add var) the symbols A, B and C may stand for either types or universes. The rule (var) is the basic rule for using variable typing assumptions in proofs. The rule (add var) states, that if a typing judgement $A : B$ holds in a context Γ, then $A : B$ also holds in any context obtained by adding further assumptions to Γ. The rule ($\rightarrow$ Intro) is used to type λ-abstraction of an individual variable, but it is restricted to variables of a type, which is in U_1. Thus passing a term of a type, which is not in U_1, as an argument to another term is not allowed. This restriction is partly bypassed by the rule (let) and finally lifted entirely, when we introduce the system λ^p_{T+}. ($\rightarrow$ Elim) is the usual rule for typing application of two suitable terms. To type λ-abstraction of a type variable we have the rule (Π Intro). Correspondingly, the rule (Π Elim) is used for typing type application, but again this rule is restricted to types in U_1. Finally, (let) allows us to explicitely substitute a term of polymorphic type σ into a variable of type σ, where the type σ may also be one, which is not in U_1.

The equational rules of λ^p

The equational rules of λ^p fix the behaviour of the $=$ symbol as that of typed equality. Thus, given preterms T and S and a type expression σ of λ^p, the judgement $\Gamma \rhd T = S : \sigma$ is to be read as "T and S are equal and of type σ in the context Γ".

(add var$_=$) $$\frac{\Gamma, x : \tau \ \mathit{context} \quad \Gamma \rhd M = N : \sigma}{\Gamma, x : \tau \rhd M = N : \sigma}$$

(ref) $$\frac{\Gamma \rhd M : \sigma}{\Gamma \rhd M = M : \sigma}$$

(sym) $$\frac{\Gamma \rhd M = N : \sigma}{\Gamma \rhd N = M : \sigma}$$

(trans) $$\frac{\Gamma \rhd M = N : \sigma \quad \Gamma \rhd N = P : \sigma}{\Gamma \rhd M = P : \sigma}$$

(ξ) $$\frac{\Gamma, x : \sigma \rhd M = N : \tau}{\Gamma \rhd \lambda x : \sigma.M = \lambda x : \sigma.N : \sigma \to \tau}$$

(ν) $$\frac{\Gamma \rhd M_1 = M_2 : \sigma \to \tau \quad \Gamma \rhd N_1 = N_2 : \sigma}{\Gamma \rhd M_1 N_1 = M_2 N_2 : \tau}$$

(α) $$\frac{\Gamma \rhd \lambda x : \sigma.M : \sigma \to \tau \quad \Gamma \rhd \lambda y : \sigma. [y/x]\, M : \sigma \to \tau}{\Gamma \rhd \lambda x : \sigma.M = \lambda y : \sigma. [y/x]\, M : \sigma \to \tau} \; y \notin FV(M)$$

(β) $$\frac{\Gamma \rhd (\lambda x : \sigma.M)N : \tau \quad \Gamma \rhd [N/x]\, M : \tau}{\Gamma \rhd (\lambda x : \sigma.M)N = [N/x]\, M : \tau}$$

(η) $$\frac{\Gamma \rhd \lambda x : \sigma.(Mx) : \sigma \to \tau \quad \Gamma \rhd M : \sigma \to \tau}{\Gamma \rhd \lambda x : \sigma.(Mx) = M : \sigma \to \tau} \; x \notin FV(M)$$

(α_Π) $$\frac{\Gamma \rhd \lambda t : U_1.M : \Pi t : U_1.\sigma \quad \Gamma \rhd \lambda s : U_1. [s/t]\, M : \Pi t : U_1.\sigma}{\Gamma \rhd \lambda t : U_1.M = \lambda s : U_1. [s/t]\, M : \Pi t : U_1.\sigma}$$

(β_Π) $$\frac{\Gamma \rhd (\lambda t : U_1.M)\tau : [\tau/t]\, \sigma \quad \Gamma \rhd [\tau/t]\, M : [\tau/t]\, \sigma}{\Gamma \rhd (\lambda t : U_1.M)\tau = [\tau/t]\, M : [\tau/t]\, \sigma}$$

$$(\eta_\Pi) \qquad \frac{\Gamma \rhd \lambda t : U_1.(Mt) : \Pi t : U_1.\sigma \quad \Gamma \rhd M : \Pi t : U_1.\sigma}{\Gamma \rhd \lambda t : U_1.(Mt) = M : \Pi t : U_1.\sigma}\; t \notin FV(M)$$

$$(\xi_\Pi) \qquad \frac{\Gamma \rhd M = N : \sigma}{\Gamma \rhd \lambda t : U_1.M = \lambda t : U_1.N : \Pi t : U_1.\sigma}$$

$$(\nu_\Pi) \qquad \frac{\Gamma \rhd M = N : \Pi t : U_1.\sigma}{\Gamma \rhd M\tau = N\tau : [\tau/t]\,\sigma}$$

$$(\text{let}_=) \qquad \frac{\Gamma \rhd (\mathsf{let}\ x : \sigma = N\ \mathsf{in}\ M) : \tau \quad \Gamma \rhd [N/x]\,M : \tau}{\Gamma \rhd (\mathsf{let}\ x : \sigma = N\ \mathsf{in}\ M) = [N/x]\,M : \tau}$$

The rule (add var$_=$) is the equivalent to (add var) for the symbol $=$. (ref), (sym) and (trans) are the usual rules for making $=$ an equivalence relation. The rule (ξ) states, that λ-abstraction of an individual variable preserves equality. (ξ_Π) does the same for λ-abstraction of a type variable. Likewise, the rule (ν) states, that term application preserves equality and (ν_Π) does the same for type application. The (α) and (α_Π) rules are both instances of the usual α-conversion of λ-calculus, expressing, that terms, which differ only in the names of bound variables are considered equal. (α) treats the case of individual variable abstraction and (α_Π) treats the case of type variable abstraction. (β) and (β_Π) are also well known in λ-calculus. They state, that the effect of application is substitution of the argument into the abstracted variable. Again, we have a separate version for term and type application. (η) and (η_Π) make sure, that a term is equal to the same term wrapped in redundant λ-abstraction and application. The rule (let $_=$) defines the behaviour of the let-operator to be that of an explicit substitution.

Remark 1.2.3 *The purpose of the* let*-construct may require some illustration. Consider the following example: Suppose we have derived $\Gamma, x : \sigma \rhd M : \tau$ and $\Gamma \rhd N : \sigma$ in λ^p, but we cannot derive $\Gamma \rhd \sigma : U_1$. Moreover, suppose we want to substitute N for x in M. The natural way to do this, would be to build the preterm $T :\equiv (\lambda x : \sigma.M)N$ and reduce it to $[N/x]\,M$ using the rule (β). However, since σ is not in U_1, T cannot be typed in λ^p and thus we may not use the rule (β) after all. In such situations the* let*-construct proves to be helpful. We may instead build the preterm $S :\equiv (\mathsf{let}\ x : \sigma = N\ \mathsf{in}\ M)$ and conclude $\Gamma \rhd S : \tau$, using the rule (let). Furthermore, we may then reduce S to the desired $[N/x]\,M$ using the rule (let $_=$).*

Remark 1.2.4 *It is worth noting, that in order to receive System F, as used by Girard [GLT89], we merely have to add the inverse of the rule ($U_1 \subseteq U_2$), namely*

$$(U_2 \subseteq U_1) \qquad \frac{\Gamma \rhd \sigma : U_2}{\Gamma \rhd \sigma : U_1}$$

to the universe rules of λ^p. This amounts to abolishing the distinction between the two universes U_1 and U_2 in all of the rules of λ^p.

1.3 Extending λ^p to λ^p_T

In the next step, we extend the system λ^p by two built-in U_1-types *nat* and *bool* and the built-in terms 0 and *succ*, which stand for the natural number 0 and the successor function respectively. We also add the built-in terms $true$, $false$, which represent truth values and finally the symbols R and D, which represent a recursion operator and a case distinction operator respectively. This is done by first extending the alphabet, preterms and type expressions of λ^p, as well as the definitions for free variables and substitution. Then we extend the rules of λ^p by additional universe axioms and additional term typing axioms and rules. We also extend the equational rules of λ^p, but we do not need any new context rules. The new system, which results from these extensions to λ^p shall be named λ^p_T.

1.3.1 Preterms and type expressions of λ^p_T

Definition 1.3.1 *We define the alphabet of λ^p_T to be that of λ^p, extended by the following symbols:*

1. *The individual constants* 0*,* $true$ *and* $false$*,*
2. *the type constants nat and bool and*
3. *the operators succ, D and R.*

Definition 1.3.2 *The type expressions of λ^p_T are defined to be those generated by the grammar*

$$\sigma ::= nat \mid bool \mid t \mid (\sigma \to \sigma) \mid \Pi t : U_1.\sigma,$$

where t is a type variable. We shall be using the words "type" and "type expression" interchangeably in the context of λ^p_T.

Definition 1.3.3 *The preterms of λ^p_T are defined to be those generated by the grammar*

$$\begin{aligned} M ::= x \mid 0 \mid true \mid false \mid \lambda x : \sigma.M \mid MM \mid succ \mid RMMM \mid DMMM \\ \mid \lambda t : U_1.M \mid M\sigma \mid (\mathsf{let}\ x : \sigma = M\ \mathsf{in}\ M), \end{aligned}$$

where x is an individual variable, t a type variable and σ a type expression.

Note, that we may define *succ* to be a preterm on its own, since its type will later be fixed to $nat \to nat$. This cannot be done for the operators D and R, because their types will vary, depending on the types of the arguments, to which they are applied.

1.3.2 Free variables and substitution

Definition 1.3.4 *Let σ be a type expression of λ^p_T. To define the set $FV(\sigma)$ of free type variables of σ, we extend Definition 1.2.4 by the following case:*

4. *$FV(\sigma) = \emptyset$ if σ is the type expression bool or nat.*

Definition 1.3.5 *Let T be a preterm of λ^p_T. To define the set $FV(T)$ of free variables of T, we extend Definition 1.2.5 by the following cases:*

7. *$FV(T) = \emptyset$ if T is the preterm 0, true, $false$ or succ.*
8. *$FV(T) = FV(LMP)$ if T is the preterm $RLMP$.*
9. *$FV(T) = FV(LPB)$ if T is the preterm $DLPB$.*

Definition 1.3.6 *Let A stand for a type expression or preterm, v for a (type or individual) variable and let σ be a type expression of λ^p_T. To define the type expression $[A/v]\,\sigma$, which results from substituting A for v in σ, we extend Definition 1.2.6 by the following cases:*

7. *$[A/v]\,bool :\equiv bool$.*
8. *$[A/v]\,nat :\equiv nat$.*

Definition 1.3.7 *Let A stand for a preterm or type expression of λ^p_T and let v be a (type or individual) variable of λ^p_T. Furthermore let T, L, M, N and P be preterms of λ^p_T. To define the preterm $[A/v]\,T$, which results from substituting A for v in T, we extend Definition 1.2.7 by the following cases:*

12. *$[A/v]\,0 :\equiv 0$.*
13. *$[A/v]\,true :\equiv true$.*
14. *$[A/v]\,false :\equiv false$.*
15. *$[A/v]\,succ :\equiv succ$.*
16. *$[A/v]\,(RLMP) :\equiv R([A/v]\,L)([A/v]\,M)([A/v]\,P)$.*
17. *$[A/v]\,(DLMP) :\equiv D([A/v]\,L)([A/v]\,M)([A/v]\,P)$.*

1.3.3 Extending the rules of λ^p

Additional universe axioms

We add the two following axioms, stating that *nat* and *bool* are U_1-types:

$(nat\ U_1)$	$\emptyset \rhd nat : U_1$
$(bool\ U_1)$	$\emptyset \rhd bool : U_1$

Additional term typing axioms and rules

To the term typing rules of λ^p we add the following:

(0 *nat*) $$\emptyset \rhd 0 : nat$$

(succ) $$\emptyset \rhd succ : nat \to nat$$

(*true bool*) $$\emptyset \rhd true : bool$$

(*false bool*) $$\emptyset \rhd false : bool$$

(rec) $$\frac{\Gamma \rhd L : \sigma \quad \Gamma \rhd M : \sigma \to (nat \to \sigma) \quad \Gamma \rhd N : nat}{\Gamma \rhd RLMN : \sigma}$$

(case) $$\frac{\Gamma \rhd M : \sigma \quad \Gamma \rhd N : \sigma \quad \Gamma \rhd B : bool}{\Gamma \rhd DMNB : \sigma}$$

The axioms (0 *nat*), (succ), (*true bool*) and (*false bool*) make sure that the newly added constants recieve their intended types. The rules (rec) and (case) are used to type the recursion and case distinction operators respectively.

Additional equational rules

We add the following axioms to treat equality for the newly introduced operators:

($\text{case}_=$ *true*) $$\frac{\Gamma \rhd DMNtrue : \sigma \quad \Gamma \rhd M : \sigma}{\Gamma \rhd DMNtrue = M : \sigma}$$

($\text{case}_=$ *false*) $$\frac{\Gamma \rhd DMNfalse : \sigma \quad \Gamma \rhd N : \sigma}{\Gamma \rhd DMNfalse = N : \sigma}$$

($\text{rec}_=$ 0) $$\frac{\Gamma \rhd RLM0 : \sigma \quad \Gamma \rhd L : \sigma}{\Gamma \rhd RLM0 = L : \sigma}$$

($\text{rec}_=$ *succ*) $$\frac{\Gamma \rhd RLM(succN) : \sigma \quad \Gamma \rhd M(RLMN)N : \sigma}{\Gamma \rhd RLM(succN) = M(RLMN)N : \sigma}$$

The rules ($\text{case}_=$ *true*) and ($\text{case}_=$ *false*) state, that either the first or the second argument is returned by the D operator, depending on whether the third argument is $true$ or $false$. ($\text{rec}_=$ 0) and ($\text{rec}_=$ *succ*) are the usual recursion equations for the operator R.

Remark 1.3.1 *Simply ignoring all mechanisms of polymorphism, we can see that the simply typed λ-calculus, referred to in [GLT89] as Gödel's System T is clearly a subsystem of λ^p_T.*

1.4 Extending λ^p_T to λ^p_{T+}

In both λ^p and λ^p_T the U_2 types are not closed under $\to$, that is to say, there is no way of forming the type $\sigma \to \tau$ when σ or τ is not a U_1-type. This restriction is now lifted by extending the system λ^p_T by two additional rules, which will guarantee the new closure condition. Since we do not extend the language of λ^p_T itself, we do not need to extend the definitions of preterms and type expressions any further. Consequently, the definitions for free variables and substitution remain the same as in λ^p_T. The system, which results from adding the two extra rules to λ^p_T shall be named λ^p_{T+}.

1.4.1 Extending the rules of λ^p_T

Additional universe rule

The following rule serves to close the universe U_2 under $\to$:

$$(\to U_2) \qquad \frac{\Gamma \rhd \sigma : U_2 \quad \Gamma \rhd \tau : U_2}{\Gamma \rhd \sigma \to \tau : U_2}$$

It states that, if σ and τ are U_2-types, then so is the type $\sigma \to \tau$.

Additional term typing rule

The last rule represents the more general version of ($\to$ Intro), that is not restricted to U_1-types.

$$(\text{full} \to \text{Intro}) \qquad \frac{\Gamma, x : \sigma \rhd M : \tau}{\Gamma \rhd \lambda x : \sigma.M : \sigma \to \tau}$$

In λ^p_{T+} we may thus pass a term of a type, which is not in U_1 as an argument to another term.

Remark 1.4.1 *It is clear, that the rule ($\to$ Intro) of λ^p is rendered obsolete by adding (full $\to$ Intro). Also rendered obsolete are (let) and (let $_=$). This is to be understood in the following way: Assume λ^p_{T*} to be the system λ^p_{T+} without the rule (let) and the rule (let $_=$). We make the following definition in λ^p_{T*}:*

$$(\mathsf{let}\ x : \sigma = N\ \mathsf{in}\ M) :\equiv (\lambda x : \sigma.M)N$$

The rule (let) now turns out to be provable in λ^p_{T*}. To see this we need to show that if $\Gamma \rhd \sigma : U_2$, $\Gamma \rhd \tau : U_1$, $\Gamma, x : \sigma \rhd M : \tau$ and $\Gamma \rhd N : \sigma$, then

$$\Gamma \rhd (\mathsf{let}\ x : \sigma = N\ \mathsf{in}\ M) : \tau$$

Consider the following formal deduction in λ^p_{T*}:

1. *$\Gamma \rhd \sigma : U_2$ (Assumption)*

2. $\Gamma \rhd \tau : U_1$ *(Assumption)*

3. $\Gamma, x : \sigma \rhd M : \tau$ *(Assumption)*

4. $\Gamma \rhd N : \sigma$ *(Assumption)*

5. $\Gamma \rhd \sigma \to \tau$ *($\to U_2$)*

6. $\Gamma \rhd \lambda x : \sigma.M : \sigma \to \tau$ *(full $\to$ Intro)*

7. $\Gamma \rhd \underbrace{(\lambda x : \sigma.M)N}_{(\text{let } x:\sigma=N \text{ in } M)} : \tau$ *($\to$ Elim)*

Furthermore, the rule (let $_=$) is subsumed by the normal (β) rule of λ^p_{T*}. In this sense, it is not necessary to include (let) and (let $_=$) in λ^p_{T+}.

1.5 The structure of λ^p_{T+} types

As we have seen, the universe rules of λ^p_{T+} divide the types into the two universes U_1 and U_2. We now introduce inductive characterisations for these universes, both in λ^p_T and λ^p_{T+}, where we refer to U_1 types as *simple* types and U_2 types as *polymorphic* types. These characterisations will become useful in Chapter 3, where we will be mapping type judgements of the form $\Gamma \rhd T : \sigma$ to formulae of explicit mathematics. Throughout this section, we shall use the phrase "Γ is a context of λ^p_T" and "Γ is a context of λ^p_{T+}" to mean, that Γ *context* is derivable in λ^p_T and λ^p_{T+} respectively. Since λ^p_{T+} is an extension of λ^p_T, every context of λ^p_T is also a context of λ^p_{T+}. The converse cannot be expected to hold, since the context rule (U_i type context) depends on the closure conditions for the type universes.

Lemma 1.5.1 *Let σ be a type expression of λ^p_T (λ^p_{T+}). If $\Gamma \rhd \sigma : U_i$ is derivable in λ^p_T (λ^p_{T+}), where $i \in \{1, 2\}$, then Γ is a context of λ^p_T (λ^p_{T+}).*

Proof We prove this by an induction on the derivation of $\Gamma \rhd \sigma : U_i$. We thus need to consider all rules of λ^p_T (λ^p_{T+}), which may lead to a judgement of this form.

<u>Axioms (*nat* U_1) and (*bool* U_1):</u> $\emptyset$ *context* holds by the context axiom (empty context), so the claim holds for both universe axioms.

<u>Rules (var) and (add var):</u> In these cases, the claim follows directly from the assumptions.

<u>Rules ($\to U_1$), ($\to U_2$) and ($U_1 \subseteq U_2$):</u> In these cases, the claim follows trivially by the induction hypothesis.

<u>Rule ($\Pi\ U_2$):</u> Then $\sigma \equiv \Pi t : U_1.\xi$. So $\Gamma, t : U_1 \rhd \xi : U_2$ by assumption and thus $\Gamma, t : U_1$ *context* holds by the induction hypothesis. This, however, can only have been obtained by the rule (U_1 context) and therefore, by assumption of that rule Γ *context* holds.

This concludes the proof. $\square$

Lemma 1.5.2 *Let Γ be a context of λ^p_T (λ^p_{T+}). Then any initial segment of Γ is also a context of λ^p_T (λ^p_{T+}).*

Proof This is an induction on the length n of Γ.

$\underline{n = 0:}$ Then Γ is the empty sequence and any initial segment Γ' of Γ must also be the empty sequence. Thus the claim holds by the axiom (empty context).

$\underline{n \mapsto n+1:}$ Then Γ is of the form $\Delta, x : C$, where x is a variable and C stands for either a type expression or a universe. Let Γ' be an initial segment of Γ. If Γ' is Γ itself, then there is nothing to prove. Assume, therefore, that Γ' is already an initial segment of Δ. Since $\Delta, x : C$ *context* holds, this must have been concluded using the rule (U_1 context) or the rule (U_i type context). If (U_1 context) was applied, then Δ is a context of λ^p_T (λ^p_{T+}) by assumption and thus the claim holds by the induction hypothesis, since Δ has length n. Otherwise, if (U_i type context) was applied, then by assumption $\Delta \rhd \sigma : U_i$, where σ is a type expression and $i \in \{1, 2\}$. Therefore by Lemma 1.5.1, Δ is a context of λ^p_T (λ^p_{T+}) and thus the claim again follows by induction hypothesis, since again Δ has length n.

$\square$

Lemma 1.5.3 *Let Γ be a context of λ^p_T (λ^p_{T+}), such that $\Gamma \rhd \sigma : U_i$ is derivable in λ^p_T (λ^p_{T+}), where $i \in \{1, 2\}$. Furthermore, let Σ be a sequence, such that Γ, Σ is a context of λ^p_T (λ^p_{T+}). Then $\Gamma, \Sigma \rhd \sigma : U_i$ is also derivable in λ^p_T (λ^p_{T+}).*

Proof This is an easy induction on the length n of the sequence Σ.

$\underline{n = 0:}$ In this case Γ, Σ is Γ and the claim follows by assumption.

$\underline{n \mapsto n+1:}$ Let Σ be the sequence $\Sigma', x : C$ of length $n+1$, where x is a variable and C stands a universe or type expression. Therefore Σ' is a sequence of length n and, since Γ, Σ is a context of λ^p_T (λ^p_{T+}), then so is Γ, Σ' by Lemma 1.5.2. Thus, by the induction hypothesis $\Gamma, \Sigma' \rhd \sigma : U_i$. We may therefore apply the rule (add var) to get $\Gamma, \Sigma \rhd \sigma : U_i$.

$\square$

To make reasoning easier, we now introduce the notion of a *type variable context*. This reflects the fact, that judgements of the form $\sigma : U_i$, where σ is a type expression and $i \in \{1, 2\}$, depend only on the binding of type variables. Some of the following lemmata might also hold for contexts in general. However, since we do not need them in the general form, we prove only the restricted versions for the sake of simplicity.

Definition 1.5.1 *Let Γ be a context of λ^p_T (λ^p_{T+}). We define the sequence $\Gamma\lfloor_{type}$ to consist of exactly those elements of Γ, which are of the form $(t : U_1)$ for some type variable t of λ^p_{T+}, in the same order, in which they appear in Γ.*

Definition 1.5.2 *We call a context Γ of λ^p_T (λ^p_{T+}) a type variable context of, λ^p_T (λ^p_{T+}) if and only if all its elements are of the form $(t : U_1)$, where t is a type variable.*

Lemma 1.5.4 *Let Γ be a context of λ^p_T (λ^p_{T+}). Then $\Gamma\lfloor_{type}$ is a type variable context of λ^p_T (λ^p_{T+}).*

Proof We only need to show that $\Gamma\lfloor_{type}$ is a context of λ^p_T (λ^p_{T+}). The fact that it is a type variable context then follows trivially, by the definition of $\Gamma\lfloor_{type}$. The proof goes by induction on the derivation of Γ *context*. We thus only need to consider the context axioms and rules.

Axiom (empty context): Then $\Gamma = \emptyset$ and thus also $\Gamma\lfloor_{type} = \emptyset$, so $\Gamma\lfloor_{type}$ is a context of λ^p_T (λ^p_{T+}), again by the axiom (empty context).

Rule (U_1 context): So $\Gamma = \Gamma', t : U_1$ and by assumption Γ' *context* holds, so by the induction hypothesis $\Gamma'\lfloor_{type}$ is a type variable context. Since $\Gamma', t : U_1$ *context* holds by assumption, $t : U_1$ does not appear in Γ' and thus it does not appear in $\Gamma'\lfloor_{type}$ either. Therefore, again by the rule (U_1 context) $\Gamma'\lfloor_{type}, t : U_1$ *context*. Then the claim holds, since $\Gamma'\lfloor_{type}, t : U_1 = \Gamma\lfloor_{type}$.

Rule (U_i type context): Then by assumption $\Gamma' \rhd \sigma : U_1$, where $\Gamma = \Gamma', x : \sigma$. So by Lemma 1.5.1, we have Γ' *context*. Therefore, by induction hypothesis $\Gamma'\lfloor_{type}$ is a type variable context and $\Gamma'\lfloor_{type} = (\Gamma', x : \sigma)\lfloor_{type} = \Gamma\lfloor_{type}$, so $\Gamma\lfloor_{type}$ *context* holds.

Therefore, the claim holds for all contexts Γ of λ^p_T (λ^p_{T+}), which concludes the proof. $\square$

Lemma 1.5.5 *Let Γ be a context of λ^p_T (λ^p_{T+}) and σ a type expression of λ^p_T (λ^p_{T+}), such that $\Gamma \rhd \sigma : U_1$ is derivable in λ^p_T (λ^p_{T+}). Then $\Gamma\lfloor_{type} \rhd \sigma : U_1$ is also derivable in λ^p_T (λ^p_{T+}).*

Proof The proof is an induction on the derivation of $\Gamma \rhd \sigma : U_1$. We need to consider only those rules and axioms, which lead to a judgment of this form.

Axiom (*nat* U_1) and axiom (*bool* U_1): Then $\Gamma = \emptyset = \Gamma\lfloor_{type}$.

Rule (var): Then $\sigma \equiv t$ for some type variable t and $\Gamma = \Gamma', t : U_1$. By assumption $\Gamma', t : U_1$ is a context of λ^p_T (λ^p_{T+}) and $\Gamma\lfloor_{type} = \Gamma'\lfloor_{type}, t : U_1$. So applying the rule (var) again, we conclude $\Gamma\lfloor_{type} \rhd t : U_1$.

Rule (add var): Then $\Gamma = \Gamma', x : C$ and by assumption $\Gamma' \rhd \sigma : U_1$ and $\Gamma', x : C$ *context*. So by the induction hypothesis, $\Gamma'\lfloor_{type} \rhd \sigma : U_1$. We must distinguish the case, where x is an individual variable from the one, where it is a type variable.

Case 1) x is an individual variable: Then $\Gamma\lfloor_{type} = \Gamma'\lfloor_{type}$ and the claim holds trivially.

Case 2) x is a type variable: In this case, since $\Gamma', x : C$ *context* holds, $x : C$ does not appear in Γ' and thus $x : C$ does not appear in $\Gamma'\lfloor_{type}$ either. Therefore, $\Gamma'\lfloor_{type}, x : C$ is a context of λ^p_T (λ^p_{T+}) by rule (U_1 context) and indeed $\Gamma'\lfloor_{type}, x : C = \Gamma\lfloor_{type}$. Applying the rule (add var) again, we then also have $\Gamma\lfloor_{type} \rhd \sigma : U_1$.

Rule ($\rightarrow U_1$): Then $\sigma \equiv \tau \rightarrow \xi$ for some type expressions τ and ξ. Therefore, by assumption $\Gamma \rhd \tau : U_1$ and $\Gamma \rhd \xi : U_1$, so by induction hypothesis we have $\Gamma\lfloor_{type} \rhd \tau : U_1$ and $\Gamma\lfloor_{type} \rhd \xi : U_1$. Thus applying rule ($\rightarrow U_1$) again, we get $\Gamma\lfloor_{type} \rhd \sigma : U_1$.

Therefore, the claim holds in all cases and thus the proof is complete. □

Definition 1.5.3 *Let Γ_1 and Γ_2 be contexts of λ^p_T (λ^p_{T+}). We define $\Gamma_1 + \Gamma_2$ to be the sequence obtained by appending those judgements in Γ_2, which are not in Γ_1 to the end of Γ_1, in the same order, in which they appear in Γ_2.*

Lemma 1.5.6 *Let Γ_1 and Γ_2 be type variable contexts of λ^p_T (λ^p_{T+}). Then $\Gamma_1 + \Gamma_2$ is also a type variable context of λ^p_T (λ^p_{T+}).*

Proof The claim follows by iterated application of the rule (U_1 context). □

Lemma 1.5.7 *If Γ is a type variable context of λ^p_T (λ^p_{T+}), then any permutation of Γ is again a type variable context of λ^p_T (λ^p_{T+}).*

Proof The proof of this claim is immediate. A type variable context Γ is built up by applying instances of the rule (U_1 context) in a certain order. We may change this order arbitrarily to obtain any permutation of Γ. □

Lemma 1.5.8 *Let Γ be a type variable context of λ^p_T (λ^p_{T+}), such that $\Gamma \rhd \sigma : U_i$ is derivable in λ^p_T (λ^p_{T+}), where σ is a type expression and $i \in \{1,2\}$. Furthermore let t be a type variable, such that $t : U_i$ is not in Γ and Γ' be a sequence obtained by inserting the judgement $t : U_i$ at any position in Γ. Then Γ' is a type variable context of λ^p_T (λ^p_{T+}) and $\Gamma' \rhd \sigma : U_i$ is derivable in λ^p_T (λ^p_{T+}).*

Proof This claim can be shown by a trivial induction on the derivation of $\Gamma \rhd \sigma : U_i$. □

Lemma 1.5.9 *Let Γ be a type variable context of λ^p_T (λ^p_{T+}), such that $\Gamma \rhd \sigma : U_i$ is derivable in λ^p_T (λ^p_{T+}), where σ is a type expression and $i \in \{1,2\}$. Furthermore, let Γ' be any permutation of Γ. Then $\Gamma' \rhd \sigma : U_i$ is also derivable in λ^p_T (λ^p_{T+}).*

Proof Note, that by Lemma 1.5.7, Γ' is a type variable context. We prove the claim by induction on the derivation of $\Gamma \rhd \sigma : U_i$. We only need to be concerned with those rules, which lead to a judgement of this form.

Axioms (*bool* U_1) and (*nat* U_1): This case is trivial.

Rule (var): Then Γ is the type variable context $\Delta, t : U_i$. Let Γ' be a type variable context of λ^p_T (λ^p_{T+}), such that Γ' is a permutation of Γ. Therefore Γ' must be of the form $\Sigma, t : U_i, \Sigma'$ and by Lemma 1.5.2 $\Sigma, t : U_i$ is also a type variable context of λ^p_T (λ^p_{T+}). We may thus use the rule (var) to conclude $\Sigma, t : U_i \rhd t : U_i$, followed by an application of Lemma 1.5.3 to obtain $\Sigma, t : U_i, \Sigma' \rhd t : U_i$ and therefore $\Gamma' \rhd \sigma : U_i$.

Rule (add var): Then Γ is the type variable context $\Delta, s : U_j$, where s is a type variable and $j \in \{1, 2\}$. Let Γ' be a permutation of Γ. So Γ' has the form $\Sigma, s : U_j, \Sigma'$. Therefore, Σ, Σ' is a permutation of Δ. Since by assumption $\Delta \rhd \sigma : U_i$, we may use the induction hypothesis to obtain $\Sigma, \Sigma' \rhd \sigma : U_i$. Then, by Lemma 1.5.8 we have $\Sigma, s : U_j, \Sigma' \rhd \sigma : U_i$ and therefore $\Gamma' \rhd \sigma : U_i$.

Rules ($\rightarrow U_1$), ($\rightarrow U_2$) and ($U_1 \subseteq U_2$): In these cases, the claim follows immediately by applying the respective rule to the induction hypothesis.

Rule (Π U_2): Then σ is of the form $\Pi t : U_1.\tau$ for some type expression τ. Consider any permutation Γ' of Γ. Then $\Gamma', t : U_1$ is a permutation of $\Gamma, t : U_1$. By assumption, we have $\Gamma, t : U_1 \rhd \tau : U_2$, so by the induction hypothesis $\Gamma', t : U_1 \rhd \tau : U_2$. Thus, using the rule ($\Pi$ U_2) we get $\Gamma' \rhd \Pi t : U_1.\tau : U_2$ and therefore $\Gamma' \rhd \sigma : U_2$.

□

Definition 1.5.4 *Let σ be a type expression of λ^p_T (λ^p_{T+}). We call σ a simple type of λ^p_T (λ^p_{T+}), if and only if*

$$\Gamma \rhd \sigma : U_1$$

is derivable in λ^p_T (λ^p_{T+}) for some type variable context Γ of λ^p_T (λ^p_{T+}).

Remark 1.5.1 *By Lemma 1.5.4 and Lemma 1.5.5 it follows, that if Γ is a (not necessarily type variable) context of λ^p_T (λ^p_{T+}) and σ is a type expression of λ^p_T (λ^p_{T+}), such that $\Gamma \rhd \sigma : U_1$, then σ is a simple type of λ^p_T (λ^p_{T+}).*

Lemma 1.5.10 *The simple types of λ^p_T (λ^p_{T+}) can be characterised inductively by the following statements:*

1. *nat and bool are simple types of λ^p_T (λ^p_{T+}).*
2. *Each type variable t is a simple type of λ^p_T (λ^p_{T+}).*
3. *If σ and τ are simple types of λ^p_T (λ^p_{T+}), then so is $\sigma \rightarrow \tau$.*
4. *Nothing else is a simple type of λ^p_T (λ^p_{T+}).*

Proof

Statement 1: This follows trivially since $\emptyset \rhd bool : U_1$ and $\emptyset \rhd nat : U_1$ are axioms and $\emptyset$ is a type variable context of λ^p_{T+} vacuously.

Statement 2: Consider the following derivation in λ^p_T:

1. $\emptyset$ context (empty context)
2. $t : U_1$ context (U_1 context)
3. $t : U_1 \rhd t : U_1$ (var)

Therefore, t is a simple type of λ^p_T (λ^p_{T+}).

Statement 3: By assumption, we have $\Gamma_1 \rhd \sigma : U_1$ and $\Gamma_2 \rhd \tau : U_1$ for some type variable contexts Γ_1 and Γ_2 of λ^p_T (λ^p_{T+}). By Lemma 1.5.6 $\Gamma_1 + \Gamma_2$ and $\Gamma_2 + \Gamma_1$ are type variable contexts of λ^p_T (λ^p_{T+}). By Lemma 1.5.3 we also have $\Gamma_1 + \Gamma_2 \rhd \sigma : U_1$ and $\Gamma_2 + \Gamma_1 \rhd \tau : U_1$. Now, trivially $\Gamma_1 + \Gamma_2$ is a permutation of $\Gamma_2 + \Gamma_1$. Therefore, by Lemma 1.5.9 we conclude $\Gamma_1 + \Gamma_2 \rhd \tau : U_1$. By applying the rule ($\rightarrow U_1$), we get $\Gamma_1 + \Gamma_2 \rhd \sigma \rightarrow \tau : U_1$. Therefore, $\sigma \rightarrow \tau$ is a simple type of λ^p_T (λ^p_{T+}).

Statement 4: Assume $\Gamma \rhd \sigma : U_1$ holds for some type variable context Γ of λ^p_T (λ^p_{T+}) and type expression σ. Then, by inspection of the axioms and rules for universes, σ can only have one of the above forms.

□

Definition 1.5.5 *Let σ be a type expression of λ^p_T. We call σ a polymorphic type of λ^p_T, if and only if*

$$\Gamma \rhd \sigma : U_2$$

is derivable in λ^p_T for some type variable context Γ of λ^p_T.

Lemma 1.5.11 *The polymorphic types of λ^p_T can be characterised inductively by the following statements:*

1. *Every simple type of λ^p_{T+} is a polymorphic type of λ^p_T.*
2. *If σ is a polymorphic type of λ^p_T, then so is $\Pi t : U_1.\sigma$.*
3. *Nothing else is a polymorphic type of λ^p_T.*

Proof

Statement 1: If σ is a simple type of λ^p_{T+}, then $\Gamma \rhd \sigma : U_1$ for some type variable context Γ of λ^p_T. So by the rule ($U_1 \subseteq U_2$), we also have $\Gamma \rhd \sigma : U_2$ and therefore σ is a polymorphic type of λ^p_T.

<u>Statement 2:</u> If σ is a polymorphic type of λ^p_T, then $\Gamma \rhd \sigma : U_2$ for some type variable context Γ of λ^p_T. If $t : U_1$ is in Γ, then consider a permutation Γ' of Γ, such that Γ' is of the form $\Delta, t : U_1$. By Lemma 1.5.7 Γ' is also a type variable context of λ^p_T. Furthermore, by Lemma 1.5.9 we have $\Delta, t : U_1 \rhd \sigma : U_2$. Therefore, by the rule (Π U_2), $\Delta \rhd \Pi t : U_1.\sigma : U_2$. Thus, since Δ is a type variable context of λ^p_T, $\Pi t : U_1.\sigma$ is a polymorphic type of λ^p_T. On the other hand, if $t : U_1$ is not in Γ then consider the following derivation:

1. Γ context (Assumption)
2. $\Gamma \rhd \sigma : U_2$ (Assumption)
3. $\Gamma, t : U_1$ context (U_1 context)
4. $\Gamma, t : U_1 \rhd \sigma : U_2$ (add var)
5. $\Gamma \rhd \Pi t : U_1.\sigma : U_2$ (Π U_2)

So since Γ is a type variable context of λ^p_T, $\Pi t : U_1.\sigma$ is also a polymorphic type of λ^p_T.

<u>Statement 3:</u> Assume $\Gamma \rhd \sigma : U_2$ for some context Γ of λ^p_T. Then, by inspection of the universe rules of λ^p_T, it follows, that σ must have one of the above forms.

$\square$

Definition 1.5.6 *Let σ be a type expression of λ^p_{T+}. We call σ a polymorphic type of λ^p_{T+} if and only if*

$$\Gamma \rhd \sigma : U_2$$

is derivable in λ^p_{T+} for some type variable context Γ of λ^p_{T+}.

Lemma 1.5.12 *The polymorphic types of λ^p_{T+} can be characterised inductively by the following statements:*

1. *Every polymorphic type of λ^p_T is a polymorphic type of λ^p_{T+}.*
2. *If σ is a polymorphic type of λ^p_{T+}, then so is $\Pi t : U_1.\sigma$.*
3. *If σ and τ are polymorphic types of λ^p_{T+}, then so is $\sigma \to \tau$.*
4. *Nothing else is a polymorphic type of λ^p_{T+}.*

Proof

<u>Statement 1:</u> The statement holds trivially, since λ^p_{T+} is an extension of λ^p_T.

<u>Statement 2:</u> The proof of this statement is completely analogous to the one for statement 2 in Lemma 1.5.11.

Statement 3: By assumption, we have $\Gamma_1 \rhd \sigma : U_2$ and $\Gamma_2 \rhd \tau : U_2$ for some type variable contexts Γ_1 and Γ_2 of λ^p_{T+}. By Lemma 1.5.6 $\Gamma_1 + \Gamma_2$ and $\Gamma_2 + \Gamma_1$ are type variable contexts of λ^p_{T+}. By Lemma 1.5.3 we also have $\Gamma_1 + \Gamma_2 \rhd \sigma : U_2$ and $\Gamma_2 + \Gamma_1 \rhd \tau : U_2$. Now, trivially $\Gamma_1 + \Gamma_2$ is a permutation of $\Gamma_2 + \Gamma_1$. Therefore, by Lemma 1.5.9 we conclude $\Gamma_1 + \Gamma_2 \rhd \tau : U_2$. By applying the rule $(\rightarrow U_1)$, we get $\Gamma_1 + \Gamma_2 \rhd \sigma \rightarrow \tau : U_2$. Therefore, $\sigma \rightarrow \tau$ is a polymorphic type of λ^p_{T+}.

Statement 4: By inspection of the universe axioms and rules of λ^p_{T+}, this statement follows trivially.

$\square$

Chapter 2

Explicit mathematics: The theory EET and extensions

2.1 Overview

In this thesis, we will often be using theories of so called *explicit mathematics*. In the following, we will discuss the logical framework, that is commonly referred to as explicit mathematics. We will see, that it is not a single logical theory, but rather a collection of axioms, which may be customised into a particular theory, according to specific requirements. The introduction of explicit mathematics shall be undertaken in three steps. In the first step, we shall explain the underlying *logic of partial terms*. In the second step, we will introduce the first-order part of explicit mathematics, also known as *applicative theories*. The last step will contain the definition of the second-order part, that is to say a group of axioms for *types and names*, which will complete the introduction.

2.2 The logic of partial terms

The logic of partial terms (henceforth also called LPT) is essentially normal first-order predicate logic, extended by the concept of *definedness*, denoted by the relation symbol $\downarrow$. Given a term t, the formula $t\downarrow$ is intuitively read as either "t has a value" in mathematical contexts or "t terminates" in computer science contexts. We will now define both the syntax and the semantics of LPT and also quote the usual adequacy-theorem.

2.2.1 The syntax of LPT

A language $\mathcal{L}$ of LPT consists of the following:

Definition 2.2.1 *The alphabet of $\mathcal{L}$ consists of*

1. *A countable set $Var = \{a, b, c, x, y, z, \ldots\}$ of variables,*

2. *the logical symbols* $\neg$, $\vee$ *and* $\exists$,

3. *the unary symbol* $\downarrow$ *for definedness,*

4. *the binary symbol* $=$ *for equality,*

5. *for every natural number* n *a (possibly empty) set* Fun_n *of* n*-ary function symbols,*

6. *for every natural number* n *a (possibly empty) set* Rel_n *of* n*-ary relation symbols and*

7. *the auxiliary symbols* (*and*).

We shall be referring to the 0-ary function symbols of $\mathcal{L}$ as the *constant symbols* of $\mathcal{L}$. With these symbols, we now succesively define $\mathcal{L}$-terms, $\mathcal{L}$-atomic formulae and $\mathcal{L}$-formulae in the usual way.

Definition 2.2.2 *The* $\mathcal{L}$*-terms are inductively defined as follows:*

1. *Every variable and constant of* $\mathcal{L}$ *is an* $\mathcal{L}$*-term.*

2. *If* $t_1, \ldots, t_n$ *are* $\mathcal{L}$*-terms, and* f *is an* n*-ary function symbol of* $\mathcal{L}$ *such that* $n \geq 1$, *then* $\mathsf{f}(t_1, \ldots, t_n)$ *is also an* $\mathcal{L}$*-term.*

3. *Nothing else is an* $\mathcal{L}$*-term.*

Definition 2.2.3 *The* $\mathcal{L}$*-atomic formulae are exactly the expressions* $a\downarrow$, $a = b$ *as well as* $\mathsf{R}(t_1, \ldots, t_n)$, *where* $a, b, t_1, \ldots, t_n$ *are* $\mathcal{L}$*-terms and* R *is an* n*-ary relation symbol.*

Definition 2.2.4 *The* $\mathcal{L}$*-formulae are inductively defined as follows:*

1. *Every* $\mathcal{L}$*-atomic formula is an* $\mathcal{L}$*-formula.*

2. *If* A *is an* $\mathcal{L}$*-formula, then* $\neg A$ *is an* $\mathcal{L}$*-formula.*

3. *If* A *and* B *are* $\mathcal{L}$*-formulae, then* $(A \vee B)$ *is an* $\mathcal{L}$*-formula.*

4. *If* A *is an* $\mathcal{L}$*-formula and* x *is a variable of* $\mathcal{L}$, *then* $\exists x A$ *is an* $\mathcal{L}$*-formula.*

5. *Nothing else is an* $\mathcal{L}$*-formula.*

In case there is no danger of ambiguity, we shall merely be speaking of terms, atomic formulae and formulae instead of $\mathcal{L}$-terms, $\mathcal{L}$-atomic formulae and $\mathcal{L}$-formulae respectively. Outermost braces shall usually be omitted. We shall be employing vector notation for finite sequences of terms, writing the sequence $a_1, \ldots, a_n$ as $\vec{a}$. Given a term t we define the set $FV(t)$ of *free variables of* t in the usual inductive manner. This definition is extended as usual to the set $FV(A)$ of *free variables of a formula* A. We call t closed, if $FV(t) = \emptyset$ and A closed if $FV(A) = \emptyset$. Furthermore, given terms $\vec{a}$, we define $A[\vec{a}/\vec{x}]$ to be the formula, which results from substituting all free occurrences of the variables $\vec{x}$ in A by the terms $\vec{a}$ respectively, avoiding collisions by renaming bound variables. The term $t[\vec{a}/\vec{x}]$ is defined analogously.

Definition 2.2.5 *We define the following syntactic abbreviations*

1. $(A \wedge B) :\equiv \neg(\neg A \vee \neg B)$.
2. $(A \rightarrow B) :\equiv (\neg A \vee B)$.
3. $(A \leftrightarrow B) :\equiv (A \rightarrow B) \wedge (B \rightarrow A)$.
4. $\forall x A :\equiv \neg\exists x \neg A$.
5. $a \simeq b :\equiv (a\downarrow \vee b\downarrow \rightarrow a = b)$.
6. $(a \neq b) :\equiv a\downarrow \wedge b\downarrow \wedge \neg(a = b)$.

Furthermore, we make the convention, that $\neg$ *binds stronger than* $\vee$ *and* $\wedge$*, which in turn bind stronger than* $\rightarrow$ *and* $\leftrightarrow$ *.*

We will now list the axioms and deduction rules of LPT, which will supply us with a Hilbert-calculus and a notion of provability. The axioms and rules may be divided into four groups: Propositional axioms and rules, quantifier axioms and rules, definedness axioms and equality axioms.

I. Propositional axioms and rules These are the usual rules of any sound and complete Hilbert-calculus for propositional logic.

II. Quantifier axioms and rules For all formulae A and B, all terms a and all variables x, we have the axiom

(Q1) $(A[a/x] \wedge a\downarrow) \rightarrow \exists x A$

and the rule

$$(\exists) \qquad \frac{A \rightarrow B}{\exists x A \rightarrow B}\ x \notin FV(B)$$

III. Definedness axioms For every n-ary function symbol f and relation symbol R and for all terms $a, b, t_1, \ldots, t_n$, we have the axioms

(D1) $a\downarrow$, for all variables or constants a.

(D2) $\mathsf{f}(t_1, \ldots, t_n)\downarrow \rightarrow t_1\downarrow \wedge \ldots \wedge t_n\downarrow$.

(D3) $(a = b) \rightarrow a\downarrow \wedge b\downarrow$.

(D4) $\mathsf{R}(t_1, \ldots, t_n)\downarrow \rightarrow t_1\downarrow \wedge \ldots \wedge t_n\downarrow$.

IV. Equality axioms For every n-ary function symbol f and relation symbol R and for all terms $a, b, t_1, \ldots, t_n, s_1, \ldots, s_n$, we have the axioms

(E1) $(a = a)$.

(E2) $(a = b) \rightarrow (b = a)$.

(E3) $(a = b) \wedge (b = c) \rightarrow (a = c)$.

(E4) $\mathsf{R}(s_1, \ldots, s_n) \wedge (s_1 = t_1) \wedge \ldots \wedge (s_n = t_n) \rightarrow \mathsf{R}(t_1, \ldots, t_n)$.

(E5) $(s_1 = t_1) \wedge \ldots \wedge (s_n = t_n) \rightarrow \mathsf{f}(s_1, \ldots, s_n) \simeq \mathsf{f}(t_1, \ldots, t_n)$.

The axioms (D2) to (D4) are sometimes referred to as the *strictness axioms*. For any formula A, we write $\mathsf{LPT} \vdash A$ to express, that A is proveable in LPT, using the axioms and rules given under **I** to **IV**. We immediately obtain the duals of the quantifier axiom and rule for the universal quantifier in the form of the following easy lemma.

Lemma 2.2.1 *For all formulae A and B, all terms a and all variables x, we have*

(i) $\mathsf{LPT} \vdash \forall x A \wedge a\downarrow \rightarrow A[a/x]$.

(ii) If $x \notin FV(A)$ then $\mathsf{LPT} \vdash A \rightarrow B \Longrightarrow \mathsf{LPT} \vdash A \rightarrow \forall x B$.

Proof To prove (i), we note that by axiom (Q1) we have

$$\mathsf{LPT} \vdash \neg A[a/x] \wedge a\downarrow \rightarrow \exists x \neg A$$

We now use Definition 2.2.5 to recieve the following syntactic equivalences

$$\neg A[a/x] \wedge a\downarrow \rightarrow \exists x \neg A \equiv \neg(\neg A[a/x] \wedge a\downarrow) \vee \exists x \neg A \equiv A[a/x] \vee \neg a\downarrow \vee \exists x \neg A \equiv$$
$$A[a/x] \vee \neg(a\downarrow \wedge \neg\exists x \neg A) \equiv A[a/x] \vee \neg(a\downarrow \wedge \forall x A)$$

So, we also have $\mathsf{LPT} \vdash A[a/x] \vee \neg(a\downarrow \wedge \forall x A)$ and by the usual propositional rule $\mathsf{LPT} \vdash \neg(a\downarrow \wedge \forall x A) \vee A[a/x]$, which is again syntactically equivalent to $\mathsf{LPT} \vdash a\downarrow \wedge \forall x A \rightarrow A[a/x]$. This proves (i).

To prove (ii) we assume that $\mathsf{LPT} \vdash A \rightarrow B[a/x]$ and $x \notin FV(A)$. Via some propositional rules we obtain the contraposition, namely $\mathsf{LPT} \vdash \neg B[a/x] \rightarrow \neg A$ and since $x \notin FV(A)$, we may apply ($\exists$) to derive $\mathsf{LPT} \vdash \exists x \neg B \rightarrow \neg A$. Applying contraposition again, we get $\mathsf{LPT} \vdash A \rightarrow \neg\exists x \neg B$, which is syntactically equivalent to $\mathsf{LPT} \vdash A \rightarrow \forall x B$. This proves (ii) and concludes the proof. $\square$

2.2.2 The semantics of LPT

We now define a semantics for LPT. It differs from a semantics of normal predicate logic only in the interpretation of the function symbols. These are interpreted as *partial* functions, that is to say functions, which may be undefined on certain elements of their domain.

Definition 2.2.6 *We define a partial $\mathcal{L}$-structure to be a quintuple*

$$\mathcal{M} = (M, I_0, I_1, I_2, \ell)$$

with the following properties

1. *M is a non-empty set and ℓ an object, such that $\ell \notin M$. M is called the universe of $\mathcal{M}$.*

2. *I_0 is a function, mapping each n-ary relation symbol R of $\mathcal{L}$ to a function $I_0(\mathsf{R}) : M^n \to \{true, false\}$.*

3. *I_1 is a function, mapping each constant symbol c of $\mathcal{L}$ to an object $I_1(\mathsf{c}) \in M$.*

4. *I_2 is a function, mapping each n-ary function symbol f of $\mathcal{L}$ where $n \geq 1$ to a partial function $I_2(\mathsf{f})$ from M^n to M.*

We also write $|\mathcal{M}|$ for M, $\bowtie_{\mathcal{M}}$ for ℓ and $\mathsf{R}^{\mathcal{M}}$, $\mathsf{c}^{\mathcal{M}}$ and $\mathsf{f}^{\mathcal{M}}$ for $I_0(\mathsf{R})$, $I_1(\mathsf{c})$ and $I_2(\mathsf{f})$ respectively.

Definition 2.2.7 *Given an $\mathcal{L}$-structure $\mathcal{M}$, we define a valuation (in $\mathcal{M}$) to be a function $\alpha : Var \to |\mathcal{M}|$. Furthermore, if α is a valuation, $x \in Var$ and $m \in |\mathcal{M}|$, then $\alpha[x = m]$ is the valuation defined by*

$$\alpha[x = m](v) := \begin{cases} m, & \text{if } v = x \\ \alpha(v) & \text{otherwise.} \end{cases}$$

Definition 2.2.8 *Let $\mathcal{M}$ be a partial $\mathcal{L}$-structure and α a valuation in $\mathcal{M}$. We inductively define the value $\mathcal{M}_\alpha(t) \in |\mathcal{M}| \cup \{\bowtie_{\mathcal{M}}\}$ of a term t as*

1. *$\mathcal{M}_\alpha(t) := \alpha(t)$ if t is a variable symbol,*

2. *$\mathcal{M}_\alpha(t) := t^{\mathcal{M}}$ if t is a constant symbol,*

3. *$\mathcal{M}_\alpha(t) := \mathsf{f}^{\mathcal{M}}(\mathcal{M}_\alpha(t_1), \ldots, \mathcal{M}_\alpha(t_n))$ if $t \equiv \mathsf{f}(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, $\mathcal{M}_\alpha(t_1), \ldots, \mathcal{M}_\alpha(t_n) \in |\mathcal{M}|$ and $\mathsf{f}^{\mathcal{M}}$ is defined on $(\mathcal{M}_\alpha(t_1), \ldots, \mathcal{M}_\alpha(t_n))$, otherwise $\mathcal{M}_\alpha(t) :=\bowtie_{\mathcal{M}}$.*

Definition 2.2.9 *Let $\mathcal{M}$ be a partial $\mathcal{L}$-structure and α a valuation in $|\mathcal{M}|$. We inductively define the value $\mathcal{M}_\alpha(A) \in \{true, false\}$ of a formula A as follows:*

1. *If $A \equiv a\!\downarrow$ then*

$$\mathcal{M}_\alpha(A) := \begin{cases} true, & \text{if } \mathcal{M}_\alpha(a) \in |\mathcal{M}|, \\ false & \text{otherwise.} \end{cases}$$

2. *If $A \equiv (a = b)$ then*

$$\mathcal{M}_\alpha(A) := \begin{cases} true, & \text{if } \mathcal{M}_\alpha(a), \mathcal{M}_\alpha(b) \in |\mathcal{M}| \\ & \text{and } \mathcal{M}_\alpha(a) = \mathcal{M}_\alpha(b), \\ false & \text{otherwise.} \end{cases}$$

3. *If $A \equiv \mathsf{R}(t_1, \ldots, t_n)$, where R is an n-ary relation symbol and $t_1, \ldots, t_n$ are terms then*
$$\mathcal{M}_\alpha(A) := \begin{cases} true, & \text{if } \mathcal{M}_\alpha(t_1), \ldots, \mathcal{M}_\alpha(t_n) \in |\mathcal{M}| \\ & \text{and } \mathsf{R}^{\mathcal{M}}(\mathcal{M}_\alpha(t_1), \ldots, \mathcal{M}_\alpha(t_n)) = true, \\ false & \text{otherwise.} \end{cases}$$

4. *If $A \equiv \neg B$ for a formula B then*
$$\mathcal{M}_\alpha(A) := \begin{cases} true, & \text{if } \mathcal{M}_\alpha(B) = false \\ false & \text{otherwise.} \end{cases}$$

5. *If $A \equiv (B \vee C)$ then*
$$\mathcal{M}_\alpha(A) := \begin{cases} true, & \text{if } \mathcal{M}_\alpha(B) = true \text{ or } \mathcal{M}_\alpha(C) = true \\ false & \text{otherwise.} \end{cases}$$

6. *If $A \equiv \exists x B$ for a formula B then*
$$\mathcal{M}_\alpha(A) := \begin{cases} true, & \text{if } \mathcal{M}_{\alpha[x=m]}(B) = true \text{ for some } m \in |\mathcal{M}| \\ false & \text{otherwise.} \end{cases}$$

2.2.3 LPT is adequate with respect to $\mathcal{L}$-structures

Given an $\mathcal{L}$-structure $\mathcal{M}$ and a formula A, we say that A *is valid in* $\mathcal{M}$ and write $\mathcal{M} \models A$ if $\mathcal{M}_\alpha(A) = true$ for all valuations α in $\mathcal{M}$. Given a set of formulae Th, we say $\mathcal{M}$ *is a model of* Th and write $\mathcal{M} \models \mathsf{Th}$, if every formula in Th is valid in $\mathcal{M}$. If A is valid in all $\mathcal{L}$-structures, we say that A *is valid* and write $\mathsf{LPT} \models A$.

We may now state the adequacy theorem, which says that LPT is sound and complete with respect to the $\mathcal{L}$-structures. A proof of the theorem shall be omitted, as it is beyond the scope of this thesis.

Theorem 2.2.1 *For every $\mathcal{L}$-formula A, we have*
$$\mathsf{LPT} \vdash A \iff \mathsf{LPT} \models A.$$

This concludes our introduction of LPT. The next two sections will now be concerned with a number of useful theories and extensions of LPT, which together constitute explicit mathematics.

2.3 Applicative theories

Now we introduce a collection of theories, stated in the language $\mathcal{L}_1$ of the logic of partial terms. The theories deal with the application of terms on other terms in a partial setting and are therefore referred to as applicative theories. The intuition behind $\mathcal{L}_1$-terms is, that they represent mathematical operations or machine programs, which can be composed to form more complex operations or programs. We first define the language $\mathcal{L}_1$ and then state the axioms, which make up the central theory BON, as well as some further axioms, which may be added to extend the theory. Next, we will see that λ-abstraction is defineable in an extension of BON and that we may prove a recursion theorem.

2.3.1 The language $\mathcal{L}_1$

The language $\mathcal{L}_1$ consists of a number of constant symbols for building terms, as well as an application symbol. Here is the formal definition.

Definition 2.3.1 *The language $\mathcal{L}_1$ of* LPT *consists of the following:*

1. *The constant symbols* k, s, p, $\mathsf{p_0}$, $\mathsf{p_1}$, $\mathsf{0}$, $\mathsf{s_N}$, $\mathsf{p_N}$, $\mathsf{d_N}$, $\mathsf{r_N}$.

2. *The binary function symbol* $*$.

3. *The unary relation symbol* N.

The constant symbols of $\mathcal{L}_1$ will be read with the following intuitive interpretations: k and s act as the usual combinators of combinatory algebra, p, $\mathsf{p_0}$ and $\mathsf{p_1}$ represent pairing and projection, $\mathsf{0}$ will stand for the natural number 0, $\mathsf{s_N}$ and $\mathsf{p_N}$ for the numerical successor and predecessor, $\mathsf{d_N}$ for the numerical case distinction operator and $\mathsf{r_N}$ for the primitive recursion operator.

The function symbol $*$ is mostly written infix, that is to say, given $\mathcal{L}_1$-terms a and b, we write $a*b$ or ab instead of $*(a, b)$. Furthermore, we make the convention, that $*$ associates to the left, so the term $a_1a_2a_3 \dots a_n$ is read as $(\dots((a_1a_2)a_3)\dots a_n)$.

Definition 2.3.2 *Let a and b be terms and x a variable. We define the following syntactic abbreviations:*

1. $a \in \mathsf{N} :\equiv \mathsf{N}(a)$

2. $(\exists x \in \mathsf{N})A :\equiv \exists x(x \in \mathsf{N} \land A)$

3. $(\forall x \in \mathsf{N})A :\equiv \forall x(x \in \mathsf{N} \rightarrow A)$

4. $(a : \mathsf{N} \rightarrow \mathsf{N}) :\equiv (\forall x \in \mathsf{N})(ax \in \mathsf{N})$

5. $(a : \mathsf{N}^{n+1} \rightarrow \mathsf{N}) :\equiv (ax : \mathsf{N}^n \rightarrow \mathsf{N})$

6. $(a, b) :\equiv \mathsf{p}ab$

7. $a' :\equiv \mathsf{s_N} a$

8. $\top :\equiv \mathsf{0} = \mathsf{0}$.

9. $\mathsf{1} :\equiv \mathsf{0}'$.

Furthermore, let $a_1, \dots, a_{n+1}$ be terms. We then define n-Tuples inductively as $(a_1) := a_1$ and $(a_1, \dots, a_{n+1}) := ((a_1, \dots, a_n), a_{n+1})$.

2.3.2 The theory BON and some extensions

The theory BON is a set of axioms, which ensure, that the symbols of $\mathcal{L}_1$ behave according to their intuitive interpretation. The axioms are split into the following five groups:

I. Partial combinatory algebra

(BON1) $\mathsf{k}xy = x$.

(BON2) $\mathsf{s}xy\downarrow \land \mathsf{s}xyz \simeq (xz)(yz)$.

II. Pairing and projections

(BON3) $\mathsf{p_0}x\downarrow \land \mathsf{p_1}x\downarrow$.

(BON4) $\mathsf{p_0}(x, y) = x \land \mathsf{p_1}(x, y) = y$.

III. Natural Numbers

(BON5) $\mathsf{0} \in \mathsf{N} \land (\forall x \in \mathsf{N})(x' \in \mathsf{N})$.

(BON6) $(\forall x \in \mathsf{N})(x' \neq \mathsf{0} \land \mathsf{p_N}(x') = x)$.

(BON7) $(\forall x \in \mathsf{N})(x \neq \mathsf{0} \to \mathsf{p_N}x \in \mathsf{N} \land (\mathsf{p_N}x)' = x)$.

IV. Definition by numerical cases

(BON8) $u \in \mathsf{N} \land v \in \mathsf{N} \land u = v \to \mathsf{d_N}xyuv = x$.

(BON9) $u \in \mathsf{N} \land v \in \mathsf{N} \land u \neq v \to \mathsf{d_N}xyuv = y$.

V. Primitive recursion on N

(BON10) $(f : \mathsf{N} \to \mathsf{N}) \land (g : \mathsf{N}^3 \to \mathsf{N}) \to (\mathsf{r_N}fg : \mathsf{N}^2 \to \mathsf{N})$.

(BON11) $(f : \mathsf{N} \to \mathsf{N}) \land (g : \mathsf{N}^3 \to \mathsf{N}) \land x \in \mathsf{N} \land y \in \mathsf{N} \land h = \mathsf{r_N}fg$
$\to hx\mathsf{0} = fx \land hx(y') = gxy(hxy)$.

In this thesis, we will also be making use of two axioms, which may be added to BON. The first one states that the application of two terms is always defined. More formally:

(Tot) $$\forall x \forall y (xy\downarrow).$$

The second axiom, that we will be adding to BON states, that terms behave extensionally with respect to application. The formulation, we will use is the following:

$$(\mathsf{Ext}) \qquad \forall f \forall g \, [\forall x (fx \simeq gx) \;\rightarrow\; f = g] \, .$$

By BON+(Tot) we will denote the theory BON, extended by the axiom (Tot). Accordingly BON+(Tot)+(Ext) will denote BON+(Tot), extended by the axiom (Ext). Furthermore, if K stands for one of the theories BON, BON+(Tot) or BON+(Tot)+(Ext) and A is an $\mathcal{L}_1$-formula, then by $K \vdash A$ we mean, that A is proveable by the axioms and rules of LPT, together with the axioms of K.

We conclude this part by proving a theorem, which states that, if we are working in BON+(Tot), all $\mathcal{L}_1$-terms are defined and we may thus reason, without paying special attention to definedness.

Theorem 2.3.1 *For every $\mathcal{L}_1$-term t we have*

$$\mathsf{BON}{+}(\mathsf{Tot}) \vdash t{\downarrow}.$$

Proof We proceed by induction on the structure of t.

$\underline{t = x \text{ for some variable } x}$: The statement holds trivially, since $\mathsf{BON} \vdash x{\downarrow}$ for every variable, by axiom (D1).

$\underline{t = c \text{ for some constant } c}$: The statement holds trivially, since $\mathsf{BON} \vdash c{\downarrow}$ for every constant, by axiom (D1).

$\underline{t = mn \text{ for } \mathcal{L}_1\text{-terms } m \text{ and } n}$: By the induction hypothesis we have

$$(1) \qquad \mathsf{BON}{+}(\mathsf{Tot}) \vdash m{\downarrow},$$
$$(2) \qquad \mathsf{BON}{+}(\mathsf{Tot}) \vdash n{\downarrow}.$$

Furthermore, by the axiom (Tot), we have

$$(3) \qquad \mathsf{BON}{+}(\mathsf{Tot}) \vdash \forall x \forall y (xy){\downarrow}.$$

So using (1) and (2), we may apply the quantifier rule introduced in Lemma 2.2.1 twice to (3) and obtain

$$\mathsf{BON}{+}(\mathsf{Tot}) \vdash mn{\downarrow}.$$

Therefore, BON+(Tot) $\vdash t{\downarrow}$ holds for all $\mathcal{L}_1$-terms t. □

Remark 2.3.1 *By Theorem 2.3.1 it follows, that if we are working in a system containing* BON+(Tot), *we have the equivalence $s \simeq t \leftrightarrow s = t$ for all $\mathcal{L}_1$-terms s and t.*

2.3.3 λ-abstraction and the recursion theorem

Using the combinators k and s, we now define the λ-abstraction $\lambda x.t$ of an $\mathcal{L}_1$-term t. We then prove a number of lemmata, which together show us, that our definition of λ-abstraction behaves like the usual untyped λ-calculus, as long as we are reasoning in the system $\mathsf{BON}+(\mathsf{Tot})+(\mathsf{Ext})$. The first lemma will show us, that λ-abstraction of a variable causes that variable to become bound and that the application of an abstracted term $\lambda x.t$ to some other term s causes s to be substituted for x in t. This last property corresponds to the usual β-rule of untyped λ-calculus. The next lemma states, that two λ-abstracted terms, which differ only in their bound variables are provably equal. This fact corresponds to the usual α-rule of untyped λ-calculus. We then show, that if two terms are provably equal, then they stay equal, when the same variable is abstracted in both of them. The corresponding rule in untyped λ-calculus is usually referred to as ξ-conversion. The next lemma, that we then prove about λ-abstraction is a property, which corresponds to the usual η-rule in untyped λ-calculus. It states, that application is in a certain sense inverse to abstraction. The last property, that we show about λ-abstraction usually also holds in the untyped λ-calculus, namely that term substitution commutes with abstraction.

Using our definition of λ-abstraction, we then prove the existence of a *fixed point combinator* rec in $\mathsf{BON}+(\mathsf{Tot})$. A recursion theorem takes care of this, proceeding in the standard way of the untyped λ-calculus. To conclude this section, we prove two lemmata, which will become useful later. The first one ensures the existence of a boolean case distinction operator $\mathsf{d_B}$ in BON, which will be the counterpart of the D operator of λ^p_T. The second lemma shows, that there exists a recursor r, which will be used to model the R operator of λ^p_T.

Definition 2.3.3 *Let t be an $\mathcal{L}_1$-term and x a variable of $\mathcal{L}_1$. We define the term $\lambda x.t$ inductively as follows:*

1. $\lambda x.t :\equiv \mathsf{skk}$, *if* $t = x$.
2. $\lambda x.t :\equiv \mathsf{k}t$, *if* $x \notin FV(t)$.
3. $\lambda x.t :\equiv \mathsf{s}(\lambda x.m)(\lambda x.n)$ *if* $x \in FV(t)$, *where* $t = mn$.

Lemma 2.3.1 *Given a variable x and $\mathcal{L}_1$-terms t and a, the following statements hold*

1. $FV(\lambda x.t) = FV(t) \setminus \{x\}$.
2. $\mathsf{BON}+(\mathsf{Tot}) \vdash (\lambda x.t)a = t\,[a/x]$.
3. $\mathsf{BON}+(\mathsf{Tot}) \vdash (\lambda x.t)x = t$.

Proof

Claim 1.: We prove this by induction on the definition of $\lambda x.t$.

$\underline{t = x}$: Then $FV(t) = \{x\}$, so

$$FV(t) \setminus \{x\} = \emptyset = FV(\mathsf{skk}) = FV(\lambda x.x) = FV(\lambda x.t).$$

$\underline{t \text{ is a term and } x \notin FV(t)}$: Then

$$FV(t) \setminus \{x\} = FV(t) = FV(\mathsf{k}t) = FV(\lambda x.t).$$

$\underline{t = mn, \text{ where } m \text{ and } n \text{ are terms and } x \in FV(t)}$: Then

$$\begin{aligned} &FV(t) \setminus \{x\} = (FV(m) \cup FV(n)) \setminus \{x\} = (FV(m) \setminus \{x\}) \cup (FV(n) \setminus \{x\}) \\ &\overset{\text{ind. hyp.}}{=} FV((\lambda x.m)(\lambda x.n)) = FV(\mathsf{s}(\lambda x.m)(\lambda x.n)) = FV(\lambda x.mn) = FV(\lambda x.t). \end{aligned}$$

Therefore, the claim holds.

Claim 2.: We again prove this by induction on $\lambda x.t$, reasoning in $\mathsf{BON}+(\mathsf{Tot})$.

$\underline{t = x}$: Then we have

$$(\lambda x.t)a = (\lambda x.x)a = \mathsf{skk}a = \mathsf{k}a(\mathsf{k}a) = a = t\,[a/x]\,.$$

$\underline{t \text{ is a term and } x \notin FV(t)}$: Then

$$(\lambda x.t)a = \mathsf{k}ta = t = t\,[a/x]\,.$$

$\underline{t = mn, \text{ where } m \text{ and } n \text{ are terms and } x \in FV(t)}$: Then

$$\begin{aligned} &(\lambda x.t)a = \mathsf{s}(\lambda x.m)(\lambda x.n)a = (\lambda x.m)a(\lambda x.n)a \overset{\text{ind. hyp.}}{=} m\,[a/x]\,n\,[a/x] \\ &= mn\,[a/x] = t\,[a/x]\,. \end{aligned}$$

Therefore, the claim holds.

Claim 3.: This follows directly from claim 2.

□

Lemma 2.3.2 *Let t be an $\mathcal{L}_1$-term, x a variable and y a variable, such that $y \notin FV(t)$. Then*

$$\mathsf{BON}+(\mathsf{Tot}) \vdash \lambda x.t = \lambda y.(t\,[y/x]).$$

Proof The proof is an induction on the definition of $\lambda x.t$.

$\underline{t = x}$: Then $\lambda x.t \equiv \mathsf{skk}$. Therefore, $\mathsf{BON}+(\mathsf{Tot})$ proves $\lambda y.(t\,[y/x]) = \lambda y.y = \mathsf{skk} = \lambda x.t$.

$\underline{x \notin FV(t)}$: Then $\lambda x.t \equiv \mathsf{k}t$. Therefore, $\mathsf{BON}+(\mathsf{Tot})$ proves $\lambda y.(t\,[y/x]) = \lambda y.t$ and since $y \notin FV(t)$, also $\lambda y.t = \mathsf{k}t = \lambda x.t$.

$x \in FV(t)$, and $t = mn$, where m and n are terms: Then $\lambda x.t \equiv \mathsf{s}(\lambda x.m)(\lambda x.n)$ and by induction hypothesis $\mathsf{BON}+(\mathsf{Tot})$ proves

$$\mathsf{s}(\lambda x.m)(\lambda x.n) = \mathsf{s}(\lambda y.(m\,[y/x]))(\lambda y.(n\,[y/x])) = \lambda y.((m\,[y/x])(n\,[y/x])) = \lambda y.((mn)\,[y/x]) = \lambda y.(t\,[y/x]).$$

So the claim holds in all cases. □

Lemma 2.3.3 *Let s and t be $\mathcal{L}_1$-terms and x a variable. Then*

$$\mathsf{BON}+(\mathsf{Tot})+(\mathsf{Ext}) \vdash \forall x(t = s) \ \rightarrow\ \lambda x.t = \lambda x.s.$$

Proof We prove this by reasoning informally in $\mathsf{BON}+(\mathsf{Tot})+(\mathsf{Ext})$. Assume that $\forall x(t = s)$ holds. Since by Lemma 2.3.1 we have $(\lambda x.t)x = t$ and $(\lambda x.s)x = s$, we also get $\forall x\,((\lambda x.t)x = (\lambda x.s)x)$. Therefore applying the axiom (Ext), it follows, that $\lambda x.t = \lambda x.s$. □

Lemma 2.3.4 *Let t be an $\mathcal{L}_1$-term and x be a variable, such that $x \notin FV(t)$. Then*

$$\mathsf{BON}+(\mathsf{Tot})+(\mathsf{Ext}) \vdash \lambda x.(tx) = t.$$

Proof Let y be a variable. By Lemma 2.3.1 $\mathsf{BON}+(\mathsf{Tot})$ proves $(\lambda x.(tx))y = (tx)\,[y/x] = ty$ and so, by the quantifier rule obtained in Lemma 2.2.1, $\mathsf{BON}+(\mathsf{Tot}) \vdash \forall y((\lambda x.(tx))y = ty)$. Therefore, our claim holds by the axiom (Ext). □

Lemma 2.3.5 *Let t and a be $\mathcal{L}_2$-terms and x and y distinct variables such that $x \notin FV(a)$. Then*

$$\mathsf{BON}+(\mathsf{Tot}) \vdash (\lambda x.t)\,[a/y] = \lambda x.(t\,[a/y]).$$

Proof We proceed by induction on the definition of $\lambda x.t$.

$t = x$: Then $\mathsf{BON}+(\mathsf{Tot})$ proves

$$(\lambda x.t)\,[a/y] = (\mathsf{skk})\,[a/y] = \mathsf{skk} = \lambda x.t = \lambda x.(t\,[a/y]).$$

t is a term and $x \notin FV(t)$: Then $\mathsf{BON}+(\mathsf{Tot})$ proves

$$(\lambda x.t)\,[a/y] = (\mathsf{k}t)\,[a/y] = \mathsf{k}(t\,[a/y])$$

and since by assumption $x \notin FV(a)$, we also have

$$\lambda x.(t\,[a/y]) = \mathsf{k}(t\,[a/y]) = (\lambda x.t)\,[a/y]\,.$$

$t = mn$, where m and n are terms and $x \in FV(t)$: Then $\mathsf{BON}+(\mathsf{Tot})$ proves

$$(\lambda x.t)\,[a/y] = (\mathsf{s}(\lambda x.m)(\lambda x.n))\,[a/y] = \mathsf{s}((\lambda x.m)\,[a/y])((\lambda x.n)\,[a/y])$$

and with the induction hypothesis

$$\mathsf{s}((\lambda x.m)\,[a/y])((\lambda x.n)\,[a/y]) = \mathsf{s}(\lambda x.(m\,[a/y]))(\lambda x.(n\,[a/y])).$$

On the other hand, $\mathsf{BON}+(\mathsf{Tot})$ also proves

$$\begin{aligned}&\lambda x.(t\,[a/y]) = \lambda x.((mn)\,[a/y]) = \\ &\lambda x.((m\,[a/y])(n\,[a/y])) = \mathsf{s}(\lambda x.(m\,[a/y]))(\lambda x.(n\,[a/y])).\end{aligned}$$

Therefore the lemma holds in all cases. □

Theorem 2.3.2 *There exists a closed term* rec*, such that*

$$\mathsf{BON}+(\mathsf{Tot}) \vdash \mathsf{rec} f = f(\mathsf{rec} f).$$

for all terms f*.*

Proof Define $t := \lambda y.f(yy)$ and $\mathsf{rec} := \lambda f.tt$. Then, by Lemma 2.3.1 $\mathsf{BON}+(\mathsf{Tot})$ proves the following equalities:

$$\mathsf{rec} f = (\lambda y.f(yy))(\lambda y.f(yy)) = f((\lambda y.f(yy))(\lambda y.f(yy))) = f(\mathsf{rec} f).$$

This concludes the proof. □

Lemma 2.3.6 *There exists a closed* $\mathcal{L}_1$*-term* $\mathsf{d_B}$*, such that for all* $\mathcal{L}_1$*-terms* l *and* m*, the following statements hold:*

1. $\mathsf{BON}+(\mathsf{Tot}) \vdash \mathsf{d_B} lm\mathsf{1} = l$
2. $\mathsf{BON}+(\mathsf{Tot}) \vdash \mathsf{d_B} lm\mathsf{0} = m$

Proof Define $\mathsf{d_B} := \lambda l.\lambda m.\lambda b.\mathsf{d_N} lmb\mathsf{1}$. Therefore, we have

$$\mathsf{d_B} lm\mathsf{1} = \mathsf{d_N} lm\mathsf{11} = l,$$

which proves statement 1. Furthermore,

$$\mathsf{d_B} lm\mathsf{0} = \mathsf{d_N} lm\mathsf{01} = m,$$

so statement 2 also holds, concluding the proof. □

Lemma 2.3.7 *There exists a closed $\mathcal{L}_1$-term* r*, such that for all $\mathcal{L}_1$-terms l, m and n, the following statements hold:*

1. $\mathsf{BON}+(\mathsf{Tot}) \vdash \mathsf{r}lm0 = l$
2. $\mathsf{BON}+(\mathsf{Tot}) \vdash \mathsf{r}lm(n') = m(\mathsf{r}lmn)n$

Proof Define $f := \lambda r.\lambda l.\lambda m.\lambda n.\mathsf{d_N} l\,(m\,(rlm\,(\mathsf{p_N}n))\,(\mathsf{p_N}n))\,n\mathsf{0}$ and $\mathsf{r} := \mathsf{rec}f$. Therefore, we have

$$\mathsf{r}lm\mathsf{0} = (\mathsf{rec}f)\,lm\mathsf{0} = f\,(\mathsf{rec}f)\,lm\mathsf{0} = f\mathsf{r}lm\mathsf{0} = \mathsf{d_N}l\,(m\,(\mathsf{r}lm\,(\mathsf{p_N}\mathsf{0}))\,(\mathsf{p_N}\mathsf{0}))\,\mathsf{00} = l,$$

which proves 1. Furthermore, we have

$$\mathsf{r}lm\,(n') = (\mathsf{rec}f)\,lm\,(n') = f\,(\mathsf{rec}f)\,lm\,(n') = f\mathsf{r}lm\,(n') = \mathsf{d_N}l\,(m\,(\mathsf{r}lmn)\,n)\,(n')\,\mathsf{0}$$

and, since $\mathsf{BON} \vdash \neg(n' = \mathsf{0})$, we get

$$\mathsf{d_N}l\,(m\,(\mathsf{r}lmn)\,n)\,(n')\,\mathsf{0} = m\,(\mathsf{r}lmn)\,n,$$

which proves 2 and concludes the proof. □

2.4 Explicit mathematics

We are now ready to introduce explicit mathematics, which consists of adding a type structure to BON. Types are read as being collections of programs, which fulfil a certain specification or operations, which have certain properties. A special feature of explicit mathematics is, that types can themselves be represented by operations via a naming relation. Every type must have at least one operation, which is its name, but not all operations are names for types. In order to introduce explicit mathematics, we will first extend the language $\mathcal{L}_1$ to the language $\mathcal{L}_2$ by adding second order symbols. We then list the axioms of the base theory EET and provide two different induction schemes, by which EET may be extended.

2.4.1 The language $\mathcal{L}_2$

The language $\mathcal{L}_2$ is an extension of $\mathcal{L}_1$, although strictly speaking, it is no longer a language of LPT, since it contains second order constructs.

Definition 2.4.1 *The language $\mathcal{L}_2$ is defined by adding the following symbols to $\mathcal{L}_1$:*

1. *A countable set of type variables $U, V, W, X, Y, Z, \ldots$.*
2. *The binary relation symbols $\in$ and $\Re$.*

3. *A constant symbol* c_e *for every natural number* e.

$\mathcal{L}_2$*-individual terms* are now defined identically to the $\mathcal{L}_1$-terms, taking into account the extra individual constants c_e for all natural numbers e. We can define *atomic formulae* and *formulae* of $\mathcal{L}_2$ in the following manner.

Definition 2.4.2 *$\mathcal{L}_2$-atomic formulae are exactly the expressions of the form* $a\!\downarrow$, $(a = b)$, $\mathsf{N}(a)$, $(a \in X)$, $(X = Y)$ *and* $\Re(a, X)$, *where* a *and* b *are individual terms of* $\mathcal{L}_2$ *and* X *and* Y *are type variables.*

Definition 2.4.3 *$\mathcal{L}_2$-formulae are defined inductively as follows*

1. *Every $\mathcal{L}_2$-atomic formula is an $\mathcal{L}_2$-formula.*
2. *If A is an $\mathcal{L}_2$-formula, then so is $\neg A$.*
3. *If A and B are $\mathcal{L}_2$-formulae, then so is $(A \vee B)$.*
4. *If A is an $\mathcal{L}_2$-formula, x an individual variable and X a type variable, then $\exists x A$ and $\exists X A$ are also $\mathcal{L}_2$-formulae.*
5. *Nothing else is an $\mathcal{L}_2$-formula.*

In the theory EET, which we will shortly introduce, a special subset of the $\mathcal{L}_2$-formulae plays and important role, namely the *elementary* formulae. The next definition introduces this notion.

Definition 2.4.4 *An $\mathcal{L}_2$-formula is called stratified, if it does not contain the relation symbol $\Re$. A stratified formula, which does not contain bound type variables is called an elementary formula.*

In addition to the abbreviations defined in Definitions 2.2.5 and 2.3.2, we also state abbreviations, involving the newly introduced symbols.

Definition 2.4.5 *Let A be an $\mathcal{L}_2$-formula, a, b and $\vec{a} = a_1, \ldots, a_n$ $\mathcal{L}_2$-terms and $\vec{X} = x_1, \ldots, x_n$ type variables. We define the following syntactic abbreviations:*

1. $\forall X A :\equiv \neg\exists X \neg A$.
2. $(\exists x \in X)A :\equiv \exists x(x \in X \wedge A)$.
3. $(\forall x \in X)A :\equiv \forall x(x \in X \to A)$.
4. $\Re(\vec{a}, \vec{X}) :\equiv \Re(a_1, X_1) \wedge \ldots \wedge \Re(a_n, X_n)$.
5. $a \mathrel{\dot{\in}} b :\equiv \exists X(\Re(b, X) \wedge a \in X)$.

2.4.2 The theory EET

The theory EET consists of the axioms of BON, extended by further axioms to take care of the second order part of the language. The axioms, which are added can be divided into three groups. The first group, termed the *naming and extensionality* axioms ensures that every type has a name, that names uniquely refer to types and that types as collections are extensional. The second group makes up the *elementary comprehension* axioms. These axioms state, that in EET, one may form a type using comprehension restricted to an elementary formula A and that the constant c_e is a name of that type, where e is a Gödel number of A. The last group of axioms consists of further *strictness* axioms. That is, strictness is extended to hold also for the $\in$ and $\Re$ relations. We now list the axioms just described.

I. Naming and extensionality

(EET1) $\exists x \Re(x, X)$.

(EET2) $\Re(a, X) \wedge \Re(a, Y) \rightarrow X = Y$.

(EET3) $\forall z(z \in X \leftrightarrow z \in Y) \rightarrow X = Y$.

II. Elementary comprehension

In order to state these axioms correctly, we assume, that we have an arbitrary, but fixed scheme of assigning Gödel-numbers to $\mathcal{L}_2$-formulae at our disposal. Furthermore, we assume, that we have arbitrary, but fixed enumerations $v1, v2, v3, \ldots$ and $V1, V2, V3, \ldots$ for the individual variables and the type variables respectively. Let A be an $\mathcal{L}_2$-formula, in which only the individual variables $v1, v2, v3, \ldots, vn$ and only the type variables $V1, V2, V3, \ldots, Vm$ appear free. Moreover, let $\vec{a} = a_1, \ldots, a_m$ and $\vec{X} = X_1, \ldots, X_n$. We write $A[\vec{a}, \vec{X}]$ to denote the $\mathcal{L}_2$-formula, which is obtained by replacing v_i by a_i and V_j by X_j, where $1 \le i \le n$ and $1 \le j \le m$. Now let $A[x, \vec{y}, \vec{Z}]$ be an elementary $\mathcal{L}_2$-formula with Gödel-number e.

(EET4) $\exists X \forall x(x \in X \leftrightarrow A[x, \vec{u}, \vec{V}])$.

(EET5) $\Re(\vec{v}, \vec{V}) \wedge \forall x(x \in X \leftrightarrow A[x, \vec{u}, \vec{V}]) \rightarrow \Re(\mathsf{c}_e(\vec{u}, \vec{v}), X)$.

III. Strictness

(EET6) $a \in X \rightarrow a\downarrow$.

(EET7) $\Re(a, X) \rightarrow a\downarrow$.

Thus, by the theory EET we mean all the axioms of BON, along with the axioms (EET1) to (EET7), using LPT as the logic for the first-order part and classical logic with equality for the second-order part. Consequently, given an $\mathcal{L}_2$-formula A, by writing EET $\vdash A$ we mean, that A is provable in the framework just described. We will denote the use of additional axioms with EET in the usual manner, so for example EET+(Tot)+(Ext) will denote the theory obtained by adding the axioms (Tot) and (Ext) to EET.

Given an elementary formula $A[x, \vec{y}, \vec{Z}]$, we write $\{x : A[x, \vec{u}, \vec{V}]\}$ for the type which is formed by elementary comprehension with A and call this a *type expression*. Accordingly, we write $b \in \{x : A[x, \vec{u}, \vec{V}]\}$ for $A[b, \vec{u}, \vec{V}]$. Using this notation, we may now express the types

$$\begin{aligned} \mathsf{N} &:= \{x : \mathsf{N}(x)\}, \\ \mathsf{B} &:= \{x : x = \mathsf{0} \vee z = \mathsf{1}\}, \\ S \to T &:= \{f : (\forall x \in S)(fx \in T)\}, \end{aligned}$$

where S and T are type expressions. By axiom ($\mathsf{EET}5$), it follows immediately, that there exists a closed term nat, such that $\mathsf{EET} \vdash \Re(\mathsf{nat}, \mathsf{N})$. Accordingly, there exists a closed term bool, for which $\mathsf{EET} \vdash \Re(\mathsf{bool}, \mathsf{B})$. The next lemma shows, that there also exists a name for $S \to T$, which is uniform in the names of S and T.

Lemma 2.4.1 *Let a and b be $\mathcal{L}_2$-terms and A and B types of* EET. *There exists a closed $\mathcal{L}_2$-term* imp*, such that*

$$\mathsf{EET} \vdash \Re(a, A) \wedge \Re(b, B) \to \Re(\mathsf{imp}(a, b), A \to B).$$

Proof By definition we have $A \to B = \{f : (\forall x \in A)(fx \in B)\}$. Assume, that e be the Gödel number of the formula $(\forall x \in A)(fx \in B)$. Then, by elementary comprehension $\Re(\mathsf{c}_e(a, b), A \to B)$ and the lemma holds with $\mathsf{imp} := \mathsf{c}_e$. □

We now show that the $\dot{\in}$ relation, which was introduced as an abbreviation in Definition 2.4.5, behaves like the normal $\in$ relation with respect to imp, but works on names, rather than types.

Lemma 2.4.2 *Let a, b, f and x be $\mathcal{L}_2$-terms. Then we have*

$$\mathsf{EET} \vdash \Re(a) \wedge \Re(b) \wedge f \,\dot{\in}\, \mathsf{imp}(a, b) \wedge x \,\dot{\in}\, a \to fx \,\dot{\in}\, b.$$

Proof We prove this claim by reasoning in EET. From $\Re(a) \wedge \Re(b)$ we know, that $\exists X \Re(a, X)$ and $\exists X \Re(b, X)$. Thus, there exist types A and B, such that

(1) $$\Re(a, A),$$

(2) $$\Re(b, B).$$

Furthermore, from $f \,\dot{\in}\, \mathsf{imp}(a, b)$ we know, that $\exists X[\Re(\mathsf{imp}(a, b), X) \wedge f \in X]$. Therefore, with Lemma 2.4.1 and Axiom ($\mathsf{EET}2$), it follows that

(3) $$f \in A \to B.$$

From $x \,\dot{\in}\, a$ we get $\exists X[\Re(a, X) \wedge x \in X]$. So again by Axiom ($\mathsf{EET}2$), it follows that

(4) $$x \in A.$$

(3) and (4) together yield $fx \in B$, which in turn with (2) implies $fx \,\dot{\in}\, b$. □

2.4.3 Induction schemes for $\mathcal{L}_2$

We now introduce two induction axioms of different strength, which can be added to EET and quote a theorem, which states the proof-theoretic strength of the two resulting systems. The first induction scheme is called *type induction* and may be used to prove $(\forall x \in \mathsf{N})(x \in X)$ for some type X. The axiom is the following.

$$(\mathsf{T\text{-}I_N}) \qquad \mathsf{0} \in X \wedge (\forall x \in \mathsf{N})(x \in X \rightarrow x' \in X) \rightarrow (\forall x \in \mathsf{N})(x \in X).$$

The second induction scheme is termed *formula induction* and can be used to prove that $(\forall x \in \mathsf{N})A(x)$, where A is an arbitrary $\mathcal{L}_2$-formula. The axiom reads as follows.

$$(\mathsf{F\text{-}I_N}) \qquad A(\mathsf{0}) \wedge (\forall x \in \mathsf{N})(A(x) \rightarrow A(x')) \rightarrow (\forall x \in \mathsf{N})A(x).$$

We derive two other induction schemes in $\mathsf{EET}+(\mathsf{T\text{-}I_N})$, which will allow us to reason more comfortably later. The first one is formula induction restricted to elementary formulae. The second derived induction scheme, is a form of type induction, involving the $\dot{\in}$ relation on names of types.

Lemma 2.4.3 *Let A be an elementary $\mathcal{L}_2$-formula. Then*

$$\mathsf{EET}+(\mathsf{T\text{-}I_N}) \vdash [A(\mathsf{0}) \wedge (\forall n \in \mathsf{N})(A(n) \rightarrow A(n'))] \rightarrow (\forall n \in \mathsf{N})A(n).$$

Proof We may form the type

$$T_A := \{n : A(n)\}$$

in EET by elementary comprehension. Now instantiating the axiom scheme $(\mathsf{T} - \mathsf{I_N})$ with T_A, we get

$$\mathsf{EET}+(\mathsf{T\text{-}I_N}) \vdash [\mathsf{0} \in T_A \wedge (\forall n \in \mathsf{N})(n \in T_A \rightarrow n' \in T_A)] \rightarrow (\forall n \in \mathsf{N})(n \in T_A)$$

and so the lemma holds by definition of T_A. □

Lemma 2.4.4 *Let a and t be $\mathcal{L}_2$-terms. Then*

$$\begin{aligned}\mathsf{EET}+(\mathsf{T\text{-}I_N}) \vdash \Re(a) \rightarrow [(t\,[\mathsf{0}/x] \,\dot{\in}\, a \wedge (\forall n \in \mathsf{N})(t\,[n/x] \,\dot{\in}\, a \rightarrow t\,[n'/x] \,\dot{\in}\, a)) \\ \rightarrow (\forall n \in \mathsf{N})(t\,[n/x] \,\dot{\in}\, a)].\end{aligned}$$

Proof By Lemma 2.4.3 it suffices to show, that

$$\mathsf{EET}+(\mathsf{T\text{-}I_N}) \vdash \Re(a) \rightarrow (t(x) \,\dot{\in}\, a \leftrightarrow A(x))$$

for some elementary $\mathcal{L}_2$-formula A. We have $t(x) \,\dot{\in}\, a \equiv \exists X[\Re(a, X) \wedge t(x) \in X]$. So there exists a type T, such that $t(x) \,\dot{\in}\, a \leftrightarrow \Re(a, T) \wedge t(x) \in T$. Now, since by assumption we have $\Re(a)$ and axiom $(\mathsf{EET}2)$ holds, we also have $t(x) \,\dot{\in}\, a \leftrightarrow t(x) \in T$. We define $A(x) :\equiv t(x) \in T$. Thus A is an elementary formula and this concludes the proof. □

The last lemma, which we show here states, that the recursion operator introduced in Lemma 2.3.7 reflects the typing properties of the R operator of λ^p_T. We can do this using our alternative form of the type induction scheme.

Lemma 2.4.5 *Let a, l, m and n be $\mathcal{L}_1$-terms, such that*

$$\mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})+(\mathsf{Tot}) \vdash \Re(a) \wedge l \mathrel{\dot\in} a \wedge m \mathrel{\dot\in} \mathsf{imp}(a, \mathsf{imp}(\mathsf{nat}, a)) \rightarrow (\forall n \in \mathsf{N})(\mathsf{r}lmn \mathrel{\dot\in} a).$$

Proof To prove this claim we reason informally in $\mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})+(\mathsf{Tot})$. Let x be a variable such that $x \notin FV(lm)$. We define $t := \mathsf{r}lmx$ and aim to use Lemma 2.4.4. Therefore, we first show $t\,[\mathsf{0}/x] \mathrel{\dot\in} a$. We have the following equalities

$$t\,[\mathsf{0}/x] = \mathsf{r}lm\mathsf{0} = \mathsf{d_N}l(m(\mathsf{r}lm(\mathsf{p_N}\mathsf{0}))(\mathsf{p_N}\mathsf{0}))\mathsf{0}\mathsf{0} = l$$

and we have $l \mathrel{\dot\in} a$ by assumption, so $t\,[\mathsf{0}/x] \mathrel{\dot\in} a$ holds. We next show, that

$$(*) \qquad (\forall n \mathrel{\dot\in} \mathsf{N})(t\,[n/x] \mathrel{\dot\in} a \rightarrow t\,[\mathsf{s_N}n/x] \mathrel{\dot\in} a).$$

also holds. Again, we have the following equalities

$$t\,[\mathsf{s_N}n/x] = \mathsf{r}lm\mathsf{s_N}n = \mathsf{d_N}l(m(\mathsf{r}lmn)n)\mathsf{s_N}n\mathsf{0}.$$

Since $\neg(\mathsf{s_N}n = \mathsf{0})$, we must show that $m(\mathsf{r}lmn)n \mathrel{\dot\in} a$. By assumption we have $\mathsf{r}lmn \mathrel{\dot\in} a$ and $m \mathrel{\dot\in} \mathsf{imp}(a, \mathsf{imp}(\mathsf{nat}, a))$, so by Lemma 2.4.2 it follows, that $m(\mathsf{r}lmn) \mathrel{\dot\in} \mathsf{imp}(\mathsf{nat}, a)$ and thus, again by Lemma 2.4.2 we have $m(\mathsf{r}lmn)n \mathrel{\dot\in} a$, so $(*)$ also holds. Therefore, our claim holds by Lemma 2.4.4. □

The following theorem about $\mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})+(\mathsf{Tot})+(\mathsf{Ext})$ and $\mathsf{EET}+(\mathsf{F}\text{-}\mathsf{I_N})+(\mathsf{Tot})+(\mathsf{Ext})$ sets these theories in relation to systems of arithmetic and therefore determines their proof-theoretical strength. The theorem can be constructed from various results given in [Fef79], [Jäg88], [Mar93] and [JS95]. Its proof is not in the scope of this thesis.

Theorem 2.4.1 *We have the following proof-theoretical equivalences*

1. *The theories* $\mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})+(\mathsf{Tot})+(\mathsf{Ext})$ *and* PA.
2. *The theories* $\mathsf{EET}+(\mathsf{F}\text{-}\mathsf{I_N})+(\mathsf{Tot})+(\mathsf{Ext})$ *and* $\Pi^0_\infty{-}\mathsf{CA}$.

Chapter 3

The interpretation of λ^p_{T+} in explicit mathematics

3.1 Overview

We now show, that the systems λ^p_T and λ^p_{T+} can be embedded naturally into the theories $\mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})$ and $\mathsf{EET}+(\mathsf{F}\text{-}\mathsf{I_N})$ respectively. We first define an interpretation mapping, which assigns $\mathcal{L}_2$-terms to preterms of λ^p_{T+} and $\mathcal{L}_2$-formulae to type judgements of λ^p_{T+}. Then we prove some important properties of the interpretation mapping. Using these properties, we then prove the actual embedding theorems, embedding both the typing and equality rules of λ^p_T and λ^p_{T+}.

3.2 The interpretation mapping $[\![\cdot]\!]$

In the following, we successively define the interpretation mapping $[\![\cdot]\!]$, using the symbol ambiguously in the customary way. We first define, how variables of λ^p_{T+} are interpreted. Then, we move on to simple type expressions, followed by preterms. Ultimatively, we define, how entire type judgements and contexts are interpreted as formulae of explicit mathematics.

Definition 3.2.1 *We define $\hat{\cdot}$ to be an injective mapping of the variables of λ^p_T into the variables of the language $\mathcal{L}_2$. That is to say, if v and w are distinct variables of λ^p_T, then $\hat{v}$ and $\hat{w}$ are distinct variables of $\mathcal{L}_2$.*

Definition 3.2.2 *Given a simple type expression σ of λ^p_{T+}, we define the $\mathcal{L}_2$-term $[\![\sigma]\!]$ inductively as follows:*

1. *If σ is of the form nat, then $[\![\sigma]\!] := \mathsf{nat}$.*

2. *If σ is of the form bool, then $[\![\sigma]\!] := \mathsf{bool}$.*

3. *If σ is a type variable t, then $[\![\sigma]\!] := \hat{t}$.*

4. *If σ is of the form $\tau \to \xi$, where τ and ξ are simple type expressions of λ^p_{T+}, then* $[\![\sigma]\!] := \mathsf{imp}([\![\tau]\!], [\![\xi]\!])$.

Definition 3.2.3 *Given a preterm T of λ^p_{T+}, we define the $\mathcal{L}_2$-term $[\![T]\!]$ inductively as follows:*

1. *If T is a variable x, then $[\![T]\!] := \hat{x}$.*
2. *If T is 0, then $[\![T]\!] := \mathsf{0}$.*
3. *If T is true, then $[\![T]\!] := \mathsf{1}$.*
4. *If T is $false$, then $[\![T]\!] := \mathsf{0}$.*
5. *If T is of the form succ, then $[\![T]\!] := \mathsf{s_N}$.*
6. *If T is of the form $DLMB$, where L, M and B are preterms of λ^p_{T+}, then $[\![T]\!] := \mathsf{d_B}\,[\![L]\!]\,[\![M]\!]\,[\![B]\!]$.*
7. *If T is of the form $RLMN$, where L, M and N are preterms of λ^p_{T+}, then $[\![T]\!] := \mathsf{r}\,[\![L]\!]\,[\![M]\!]\,[\![N]\!]$.*
8. *If T is of the form $\lambda x:\sigma.M$, where M is a preterm of λ^p_{T+}, then $[\![T]\!] := \lambda\hat{x}.\,[\![M]\!]$.*
9. *If T is of the form MN, where M and N are preterms of λ^p_{T+}, then $[\![T]\!] := [\![M]\!]\,[\![N]\!]$.*
10. *If T is of the form $\lambda t:U_1.M$, where M is a preterm of λ^p_{T+}, then $[\![T]\!] := [\![M]\!]$.*
11. *If T is of the form $M\sigma$, where M is a preterm and σ a type expression of λ^p_{T+}, then $[\![T]\!] := [\![M]\!]$.*
12. *If T is of the form $(\mathsf{let}\ x:\sigma = M\ \mathsf{in}\ N)$, where M and N are preterms and σ a type expression of λ^p_{T+}, then $[\![T]\!] := (\lambda\hat{x}.\,[\![N]\!])\,[\![M]\!]$.*

In the next definition we will make crucial use of the inductive characterisations for the type universes, which were established by Lemmata 1.5.10, 1.5.11 and 1.5.12.

Definition 3.2.4 *Let T be a preterm and σ a type expression of λ^p_{T+}. We interpret judgements about T and σ as $\mathcal{L}_2$-formulae in the following manner:*

1. $[\![\sigma : U_1]\!] := \Re([\![\sigma]\!])$.
2. $[\![T:\sigma]\!] := [\![T]\!] \mathrel{\dot{\in}} [\![\sigma]\!]$, *if σ is a simple type expression of λ^p_{T+}.*
3. $[\![T : \Pi t : U_1.\sigma]\!] := \forall\hat{t}(\Re(\hat{t}) \;\to\; [\![T:\sigma]\!])$.
4. $[\![T:\sigma\to\tau]\!] := \forall\hat{s}([\![(s:\sigma)]\!] \;\to\; [\![(Ts:\tau)]\!])$, *where s is an individual variable of λ^p_{T+}, such that $s \notin FV(T)$, if σ or τ is not a simple type expression of λ^p_{T+}.*

Definition 3.2.5 *We define the interpretation of a context* Γ *of* λ^p_{T+} *inductively as follows:*

1. $[\![\emptyset]\!] := \top$.
2. $[\![\Gamma, t : U_1]\!] := [\![\Gamma]\!] \wedge [\![t : U_1]\!]$.
3. $[\![\Gamma, x : \sigma]\!] := [\![\Gamma]\!] \wedge [\![x : \sigma]\!]$.
4. $[\![\Gamma, t : U_2]\!] := [\![\Gamma]\!]$.

3.3 Properties of $[\![\cdot]\!]$

We now prove some important properties of the interpretation mapping $[\![\cdot]\!]$. First, we will see, that the separation of variables into type and individual variables is preserved by the interpretation. Then we prove the most important property for our application, namely that $[\![\cdot]\!]$ commutes with substitution of both type and individual variables. The last property, which we show, is that if the interpretations of two preterms of λ^p_{T+} are provably equal in EET, then they may replace each other in the interpretation of a type judgement.

Lemma 3.3.1 *Let* T *be a preterm,* t *a type variable,* σ *a simple type expression and* x *an individual variable of* λ^p_{T+}. *Then*

1. $\hat{t} \notin FV([\![T]\!])$,
2. $\hat{x} \notin FV([\![\sigma]\!])$.

Proof Both claims follow by trivial inductions and the fact that $\hat{\cdot}$ is injective. □

Lemma 3.3.2 *Let* σ *and* τ *be simple type expressions of* λ^p_{T+} *and* t *a type variable. Then*

$$\mathsf{EET} \vdash [\![[\tau/t]\,\sigma]\!] = [\![\sigma]\!]\left[[\![\tau]\!]/\hat{t}\right].$$

Proof We proceed by induction on the structure of σ.

$\underline{\sigma \equiv nat}$: Then the lemma trivially holds, since both nat and $[\![nat]\!]$ are closed.

$\underline{\sigma \equiv bool}$: Again the lemma trivially holds, since both $bool$ and $[\![bool]\!]$ are closed.

$\underline{\sigma \equiv v\text{, where } v \text{ is a type variable distinct from } t}$: Then EET proves $[\![[\tau/t]\,\sigma]\!] = [\![v]\!] = \hat{v}$ and by definition of $\hat{\cdot}$, we have $\hat{v} \neq \hat{t}$, so $[\![\sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] = \hat{v} = [\![[\tau/t]\,\sigma]\!]$.

$\underline{\sigma \equiv t}$: Then EET proves $[\![[\tau/t]\,\sigma]\!] = [\![\tau]\!]$ and since $[\![\sigma]\!] = \hat{t}$, also $[\![\sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] = [\![\tau]\!]$.

$\sigma \equiv \xi \to \gamma$, where ξ and γ are simple type expressions: Then EET proves

$$[\![[\tau/t]\,\sigma]\!] = [\![[\tau/t]\,(\xi \to \gamma)]\!] = [\![[\tau/t]\,\xi \to [\tau/t]\,\gamma]\!] = \mathsf{imp}([\![[\tau/t]\,\xi]\!]\,, [\![[\tau/t]\,\gamma]\!])$$

and by the induction hypothesis EET proves

$$\begin{aligned}\mathsf{imp}([\![[\tau/t]\,\xi]\!]\,, [\![[\tau/t]\,\gamma]\!]) &= \mathsf{imp}([\![\xi]\!]\left[[\![\tau]\!]/\hat{t}\right], [\![\gamma]\!]\left[[\![\tau]\!]/\hat{t}\right])\\ &= \mathsf{imp}([\![\xi]\!]\,, [\![\gamma]\!])\left[[\![\tau]\!]/\hat{t}\right] = [\![\sigma]\!]\left[[\![\tau]\!]/\hat{t}\right].\end{aligned}$$

This concludes the proof. □

Lemma 3.3.3 *Let σ be a (polymorphic) type expression of λ^p_{T+}, τ a simple type expression and t a type variable. Furthermore, let T be a preterm of λ^p_{T+}. Then*

$$\mathsf{EET} \vdash [\![T : \sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] \;\leftrightarrow\; [\![T : [\tau/t]\,\sigma]\!]\,.$$

Proof We proceed by induction on the structure of σ. We list the necessary equivalences. In the case of logical equivalences, we mean, that they are provable in EET.

σ is a simple type expression:

$$\begin{aligned}[\![T : \sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] &\equiv ([\![T]\!] \mathrel{\dot{\in}} [\![\sigma]\!])\left[[\![\tau]\!]/\hat{t}\right]\\ &\overset{\text{Lemma 3.3.1}}{\equiv} [\![T]\!] \mathrel{\dot{\in}} ([\![\sigma]\!]\left[[\![\tau]\!]/\hat{t}\right])\\ &\overset{\text{Lemma 3.3.2}}{\leftrightarrow} [\![T]\!] \mathrel{\dot{\in}} [\![[\tau/t]\,\sigma]\!] \equiv [\![T : [\tau/t]\,\sigma]\!]\end{aligned}$$

$\sigma \equiv \Pi t : U_1.\xi$ and ξ is a type expression:

$$[\![T : \sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] \equiv (\forall \hat{t}(\Re(\hat{t}) \;\to\; [\![T : \xi]\!]))\left[[\![\tau]\!]/\hat{t}\right] \equiv [\![T : \sigma]\!] \equiv [\![T : [\tau/t]\,\sigma]\!]$$

$\sigma \equiv \Pi s : U_1.\xi$, s is a type variable, distinct from t and ξ is a type expression:
There are two cases to consider.

Case 1) $s \notin FV(\tau)$:

$$\begin{aligned}[\![T : \sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] &\equiv \forall \hat{s}(\Re(\hat{s}) \;\to\; [\![T : \xi]\!])\left[[\![\tau]\!]/\hat{t}\right]\\ &\equiv \forall \hat{s}(\Re(\hat{s}) \;\to\; ([\![T : \xi]\!]\left[[\![\tau]\!]/\hat{t}\right]))\\ &\overset{\text{ind. hyp.}}{\leftrightarrow} \forall \hat{s}(\Re(\hat{s}) \;\to\; ([\![T : [\tau/t]\,\xi]\!])) \equiv [\![T : [\tau/t]\,\sigma]\!]\end{aligned}$$

Case 2) $s \in FV(\tau)$:

$$[\![T : \sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] \equiv \forall \hat{s}(\Re(\hat{s}) \;\to\; [\![T : \xi]\!])\left[[\![\tau]\!]/\hat{t}\right]$$

We choose a type variable r, distinct from s, such that $r \notin FV(\tau) \cup FV(\sigma)$. Then we have

$$\begin{aligned}
&\forall \hat{s}(\Re(\hat{s}) \rightarrow [\![T:\xi]\!])\left[[\![\tau]\!]/\hat{t}\right] \leftrightarrow \\
&\forall \hat{r}(\Re(\hat{r}) \rightarrow [\![T:\xi]\!]\,[\hat{r}/\hat{s}])\left[[\![\tau]\!]/\hat{t}\right] \overset{\text{ind. hyp.}}{\leftrightarrow} \\
&\forall \hat{r}(\Re(\hat{r}) \rightarrow [\![T:[r/s]\,\xi]\!])\left[[\![\tau]\!]/\hat{t}\right] \overset{\text{ind. hyp.}}{\leftrightarrow} \\
&\forall \hat{r}(\Re(\hat{r}) \rightarrow [\![T:[\tau/t]\,[r/s]\,\xi]\!]) \equiv \\
&[\![T:\Pi r:U_1.\,[\tau/t]\,[r/s]\,\xi]\!] \leftrightarrow \\
&[\![T:[\tau/t]\,(\Pi s:U_1.\xi)]\!] \equiv \\
&[\![T:[\tau/t]\,\sigma]\!]
\end{aligned}$$

<u>$\sigma \equiv \xi \rightarrow \eta$, where ξ or η is not a simple type expression:</u> Then

$$[\![T:\sigma]\!]\left[[\![\tau]\!]/\hat{t}\right] \equiv (\forall \hat{s}([\![s:\xi]\!] \rightarrow [\![Ts:\eta]\!]))\left[[\![\tau]\!]/\hat{t}\right],$$

where s is an individual variable of λ^p_{T+}, such that $s \notin FV(T)$. Now, since s is an individual variable and t a type variable of λ^p_{T+}, the variables $\hat{s}$ and $\hat{t}$ are distinct and by Lemma 3.3.1 $\hat{s} \notin FV([\![\sigma]\!])$. Therefore,

$$\begin{aligned}
&(\forall \hat{s}([\![s:\xi]\!] \rightarrow [\![Ts:\eta]\!]))\left[[\![\tau]\!]/\hat{t}\right] \equiv \\
&\forall \hat{s}(([\![s:\xi]\!]\left[[\![\tau]\!]/\hat{t}\right]) \rightarrow ([\![Ts:\eta]\!]\left[[\![\tau]\!]/\hat{t}\right])) \overset{\text{ind. hyp.}}{\leftrightarrow} \\
&\forall \hat{s}([\![s:[\tau/t]\,\xi]\!] \rightarrow [\![Ts:[\tau/t]\,\eta]\!]) \equiv \\
&[\![([\tau/t]\,\xi) \rightarrow ([\tau/t]\,\eta)]\!] \equiv \\
&[\![T:[\tau/t]\,\sigma]\!].
\end{aligned}$$

Hence, the claim holds for all type expressions σ. □

Lemma 3.3.4 *Let T and S be preterms and x an individual variable of λ^p_{T+}. Then*

$$\mathsf{EET}{+}(\mathsf{Tot}) \vdash [\![[S/x]\,T]\!] = [\![T]\!]\,[[\![S]\!]/\hat{x}].$$

Proof We prove this lemma by induction on the structure of T. We list the necessary equalities and mean, that they are provable by reasoning in $\mathsf{EET}{+}(\mathsf{Tot})$ and λ^p_{T+}. The axiom (Tot) is needed, since we require the properties asserted by Lemma 2.3.2 and 2.3.5 to prove this claim in the case, where T is a λ-abstraction over an individual variable.

<u>$T \equiv true$ or $T \equiv false$ or $T \equiv 0$ or $T \equiv succ$:</u>

$$[\![[S/x]\,T]\!] = [\![T]\!] = [\![T]\!]\,[[\![S]\!]/\hat{x}]$$

<u>$T \equiv x$:</u>

$$[\![[S/x]\,T]\!] = [\![S]\!] = \hat{x}\,[[\![S]\!]/\hat{x}] = [\![T]\!]\,[[\![S]\!]/\hat{x}]$$

$\underline{T \equiv y}$, where y is an individual variable, distinct from x:

$$[\![[S/x]\,T]\!] = [\![T]\!] = \hat{y}\,[[\![S]\!]\,/\hat{x}] = [\![T]\!]\,[[\![S]\!]\,/\hat{x}]$$

$\underline{T \equiv \lambda x : \sigma.M}$:

$$[\![[S/x]\,T]\!] = [\![T]\!] = (\lambda\hat{x}.\,[\![M]\!])\,[[\![S]\!]\,/\hat{x}] = [\![T]\!]\,[[\![S]\!]\,/\hat{x}]$$

$\underline{T \equiv \lambda y : \sigma.M}$, where y is an individual variable, distinct from x:
We must distinguish two cases.

Case 1) $y \notin FV(S)$:

$$[\![[S/x]\,T]\!] = [\![\lambda w : \sigma.\,[S/x]\,M]\!] = \lambda\hat{w}.\,[\![[S/x]\,M]\!] \overset{\text{ind. hyp.}}{=}$$
$$\lambda\hat{w}.([\![M]\!]\,[[\![S]\!]\,/\hat{x}]) \overset{\text{Lemma 2.3.5}}{=} (\lambda\hat{y}.\,[\![M]\!])\,[[\![S]\!]\,/\hat{x}] = [\![T]\!]\,[[\![S]\!]\,/\hat{x}]$$

Case 2) $y \in FV(S)$:

$$[\![[S/x]\,T]\!] = [\![\lambda z : \sigma.\,[S/x]\,[z/y]\,M]\!] = \lambda\hat{z}.\,[\![[S/x]\,[z/y]\,M]\!]$$

where z is an individual variable, distinct from both y and x, such that $z \notin FV(M) \cup FV(S)$. Therefore,

$$\lambda\hat{z}.\,[\![[S/x]\,[z/y]\,M]\!] \overset{\text{ind. hyp.}}{=} \lambda\hat{z}.([\![[z/y]\,M]\!]\,[[\![S]\!]\,/\hat{x}]) \overset{\text{ind. hyp.}}{=}$$
$$\lambda\hat{z}.(([\![M]\!]\,[\hat{z}/\hat{y}])\,[[\![S]\!]\,/\hat{x}]) \overset{\text{Lemma 2.3.5}}{=} (\lambda\hat{z}.([\![M]\!]\,[\hat{z}/\hat{y}]))\,[[\![S]\!]\,/\hat{x}] \overset{\text{Lemma 2.3.2}}{=}$$
$$(\lambda\hat{y}.\,[\![M]\!])\,[[\![S]\!]\,/\hat{x}] = [\![T]\!]\,[[\![S]\!]\,/\hat{x}]$$

$\underline{T \equiv MN}$:

$$[\![[S/x]\,T]\!] = [\![([S/x]\,M)([S/x]\,N)]\!] = [\![[S/x]\,M]\!]\,[\![[S/x]\,N]\!] \overset{\text{ind. hyp.}}{=}$$
$$([\![M]\!]\,[[\![S]\!]\,/\hat{x}])([\![N]\!]\,[[\![S]\!]\,/\hat{x}]) = ([\![M]\!]\,[\![N]\!])\,[[\![S]\!]\,/\hat{x}] = [\![T]\!]\,[[\![S]\!]\,/\hat{x}]$$

$\underline{T \equiv RLMN \text{ or } T \equiv DMNB}$: These cases are analogous to the case $T \equiv MN$. The operators *succ*, R and D are closed and map to closed terms under $[\![\cdot]\!]$.

$\underline{T \equiv \lambda t : U_1.M}$: Since t is a type variable and x an individual variable, they are distinct. We must thus distinguish between two cases:

Case 1) $t \notin FV(S)$:

$$[\![[S/x]\,T]\!] = [\![\lambda t : U_1.\,[S/x]\,M]\!] = [\![[S/x]\,M]\!] \overset{\text{ind. hyp.}}{=}$$
$$[\![M]\!]\,[[\![S]\!]\,/\hat{x}] = [\![T]\!]\,[[\![S]\!]\,/\hat{x}]$$

Case 2) $t \in FV(S)$: In this case we choose a type variable s of λ^p_{T+} in such a way, that $s \notin FV(S) \cup FV(M)$. Then we have

$$\begin{aligned}
&[\![[S/x]\, T]\!] = [\![\lambda s : U_1.([S/x]\,[s/t]\, M)]\!] = [\![[S/x]\,[s/t]\, M]\!] \overset{\text{ind. hyp.}}{=} \\
&[\![[s/t]\, M]\!]\, [[\![S]\!] / \hat{x}] \overset{\text{ind. hyp.}}{=} [\![M]\!]\, [\hat{s}/\hat{t}]\, [[\![S]\!] / \hat{x}] \overset{\text{Lemma 3.3.1}}{=} \\
&[\![M]\!]\, [[\![S]\!] / \hat{x}] = [\![T]\!]\, [[\![S]\!] / \hat{x}]
\end{aligned}$$

<u>$T \equiv M\sigma$:</u>

$$\begin{aligned}
&[\![[S/x]\, T]\!] = [\![([S/x]\, M)([S/x]\, \sigma)]\!] = [\![[S/x]\, M]\!] \overset{\text{ind. hyp.}}{=} \\
&[\![M]\!]\, [[\![S]\!] / \hat{x}] = [\![T]\!]\, [[\![S]\!] / \hat{x}]
\end{aligned}$$

<u>$T \equiv (\mathsf{let}\ x : \sigma = M\ \mathsf{in}\ N)$:</u>

$$\begin{aligned}
&[\![[S/x]\, T]\!] = [\![(\mathsf{let}\ x : ([S/x]\, \sigma) = [S/x]\, M\ \mathsf{in}\ N)]\!] = (\lambda \hat{x}.\, [\![N]\!])([\![[S/x]\, M]\!]) \overset{\text{ind. hyp.}}{=} \\
&(\lambda \hat{x}.\, [\![N]\!])([\![M]\!]\, [[\![S]\!] / \hat{x}]) = ((\lambda \hat{x}.\, [\![N]\!])\, [\![M]\!])\, [[\![S]\!] / \hat{x}] = [\![T]\!]\, [[\![S]\!] / \hat{x}]
\end{aligned}$$

<u>$T \equiv (\mathsf{let}\ y : \sigma = M\ \mathsf{in}\ N)$, where y is an individual variable, distinct from x:</u>

$$\begin{aligned}
&[\![[S/x]\, T]\!] = [\![(\mathsf{let}\ x : ([S/x]\, \sigma) = [S/x]\, M\ \mathsf{in}\ [S/x]\, N)]\!] = \\
&(\lambda \hat{y}.\, [\![[S/x]\, N]\!])([\![[S/x]\, M]\!]) \overset{\text{ind. hyp.}}{=} (\lambda \hat{y}.\, [\![N]\!]\, [[\![S]\!] / \hat{x}])([\![M]\!]\, [[\![S]\!] / \hat{x}]) \overset{\text{Lemma 2.3.5}}{=} \\
&((\lambda \hat{y}.\, [\![N]\!])\, [[\![S]\!] / \hat{x}])([\![M]\!]\, [[\![S]\!] / \hat{x}]) = ((\lambda \hat{y}.\, [\![N]\!])\, [\![M]\!])\, [[\![S]\!] / \hat{x}] = [\![T]\!]\, [[\![S]\!] / \hat{x}]
\end{aligned}$$

Hence, the claim holds for all preterms T. □

Lemma 3.3.5 *Let T and S be preterms, σ a type expression and x an individual variable of λ^p_{T+}. Then*

$$\mathsf{EET}{+}(\mathsf{Tot}) \vdash [\![T : \sigma]\!]\, [[\![S]\!] / \hat{x}] \leftrightarrow [\![[S/x]\, T : \sigma]\!]\,.$$

Proof We proceed by induction on the structure of σ. We list the necessary equivalences. In the case of logical equivalences, we mean, that they are provable in $\mathsf{EET}{+}(\mathsf{Tot})$. The axiom (Tot) is needed, since we make use of Lemma 3.3.4 to prove the claim in the case, where σ is a simple type expression.

<u>σ is a simple type expression:</u>

$$\begin{aligned}
&[\![T : \sigma]\!]\, [[\![S]\!] / \hat{x}] \equiv ([\![T]\!] \mathrel{\dot\in} [\![\sigma]\!])\, [[\![S]\!] / \hat{x}] \overset{\text{Lemma 3.3.1}}{\equiv} ([\![T]\!]\, [[\![S]\!] / \hat{x}]) \mathrel{\dot\in} [\![\sigma]\!] \overset{\text{Lemma 3.3.4}}{\leftrightarrow} \\
&[\![[S/x]\, T]\!] \mathrel{\dot\in} [\![\sigma]\!] \equiv [\![[S/x]\, T : \sigma]\!]
\end{aligned}$$

<u>$\sigma \equiv \Pi t : U_1.\xi$, where ξ is a type expression:</u>

$$\begin{aligned}
&[\![T : \sigma]\!]\, [[\![S]\!] / \hat{x}] \equiv \forall \hat{t}(\Re(\hat{t}) \rightarrow [\![T : \xi]\!])\, [[\![S]\!] / \hat{x}] \overset{\text{Lemma 3.3.1}}{\equiv} \\
&\forall \hat{t}(\Re(\hat{t}) \rightarrow [\![T : \xi]\!]\, [[\![S]\!] / \hat{x}]) \overset{\text{ind. hyp.}}{\leftrightarrow} \forall \hat{t}(\Re(\hat{t}) \rightarrow [\![[S/x]\, T : \xi]\!]) \equiv [\![[S/x]\, T : \sigma]\!]
\end{aligned}$$

$\sigma \equiv \tau \to \xi$, where τ or ξ is not a simple type expression:

$$[\![T : \sigma]\!]\,[[\![S]\!]\,/\hat{x}] \equiv \forall\hat{y}([\![y : \tau]\!] \;\to\; [\![Ty : \xi]\!])\,[[\![S]\!]\,/\hat{x}] \;\leftrightarrow$$
$$\forall\hat{z}([\![z : \tau]\!] \;\to\; [\![Tz : \xi]\!])\,[[\![S]\!]\,/\hat{x}]\,,$$

where y is an individual variable, such that $y \notin FV(T)$ and z is an individual variable, distinct from x, such that $z \notin FV(T)$. Then

$$\forall\hat{z}([\![z : \tau]\!] \;\to\; [\![Tz : \xi]\!])\,[[\![S]\!]\,/\hat{x}] \equiv \forall\hat{z}([\![z : \tau]\!]\,[[\![S]\!]\,/\hat{x}] \;\to\; [\![Tz : \xi]\!])\,[[\![S]\!]\,/\hat{x}])$$
$$\overset{\text{ind. hyp.}}{\leftrightarrow} \forall\hat{z}([\![z : \tau]\!] \;\to\; [\![([S/x]\,T)z : \xi]\!]) \;\leftrightarrow\; [\![[S/x]\,T : \sigma]\!]\,.$$

Hence, the claim holds for all type expressions σ. □

Lemma 3.3.6 *Let T and S be preterms and σ a type expression of λ^p_{T+}, such that*

$$\mathsf{EET} \vdash [\![S]\!] = [\![T]\!] \;\to\; ([\![S : \sigma]\!] \;\leftrightarrow\; [\![T : \sigma]\!])\,.$$

Proof We prove this lemma by induction on the structure of σ, reasoning in EET.

σ is a simple type expression: In this case,

$$[\![S : \sigma]\!] \equiv [\![S]\!] \;\dot{\in}\; [\![\sigma]\!]\,,$$
$$[\![T : \sigma]\!] \equiv [\![T]\!] \;\dot{\in}\; [\![\sigma]\!]\,.$$

Define $A(x) :\equiv x \,\dot{\in}\, [\![\sigma]\!]$. We now have $[\![T]\!] = [\![S]\!] \;\to\; (A([\![S]\!]) \;\leftrightarrow\; A([\![T]\!]))$ and therefore, by assumption $A([\![S]\!]) \;\leftrightarrow\; A([\![T]\!])$ which proves the claim.

$\sigma \equiv \Pi t : U_1.\tau$: In this case

$$[\![S : \sigma]\!] \equiv \forall\hat{t}(\Re(\hat{t}) \;\to\; [\![S : \tau]\!]),$$
$$[\![T : \sigma]\!] \equiv \forall\hat{t}(\Re(\hat{t}) \;\to\; [\![T : \tau]\!])$$

and by induction hypothesis $[\![S : \tau]\!] \;\leftrightarrow\; [\![T : \tau]\!]$, so the claim holds.

$\sigma \equiv \tau \to \xi$, where τ or ξ is not simple: In this case

$$[\![S : \sigma]\!] \equiv \forall\hat{x}([\![x : \tau]\!] \;\to\; [\![Sx : \xi]\!]),$$
$$[\![T : \sigma]\!] \equiv \forall\hat{y}([\![y : \tau]\!] \;\to\; [\![Ty : \xi]\!]),$$

where $x \notin FV(S)$ and $y \notin FV(T)$. We choose a variable $z \notin FV(S) \cup FV(T)$ and note that

$$\forall\hat{x}([\![x : \tau]\!] \;\to\; [\![Sx : \xi]\!]) \;\leftrightarrow\; \forall\hat{z}([\![z : \tau]\!] \;\to\; [\![Sz : \xi]\!]),$$
$$\forall\hat{y}([\![y : \tau]\!] \;\to\; [\![Ty : \xi]\!]) \;\leftrightarrow\; \forall\hat{z}([\![z : \tau]\!] \;\to\; [\![Tz : \xi]\!]).$$

Now since by assumption $[\![S]\!] = [\![T]\!]$, we also have $[\![Sz]\!] = [\![Tz]\!]$ and so by induction hypothesis $[\![Sz : \xi]\!] \;\leftrightarrow\; [\![Tz : \xi]\!]$. Therefore, our claim holds.

Thus, the claim holds for all type expressions σ, which concludes the proof. □

3.4 Embedding theorems

Finally, we show, that the interpretation mapping $[\![\cdot]\!]$ indeed defines an embedding of λ^p_T into EET+(T-$\mathsf{I_N}$)+(Tot) and of λ^p_{T+} into EET+(F-$\mathsf{I_N}$)+(Tot). For this purpose, we first show, that the interpretation of a simple type can always be proved to be a name. Then we show, that if a type judgement is derivable in λ^p_T, then its interpretation is provable in EET+(T-$\mathsf{I_N}$)+(Tot). Correspondingly, we show, that if a type judgement is derivable in λ^p_{T+}, then its interpretation is provable in EET+(F-$\mathsf{I_N}$)+(Tot).

Induction is required when proving, that the interpretation preserves typing of the R-operator. The difference in the power of the induction schemes arises from the fact, that in λ^p_{T+} the R-operator may also work on types, which are not simple. This is not possible in the case of λ^p_T, where all arrow-types are automatically simple.

The last theorem, which we prove states, that if we add the axioms (Ext) to the theories, the typed equality symbol of λ^p_{T+} corresponds to equality in explicit mathematics.

Theorem 3.4.1 *Let Γ be a context and σ a type expression of λ^p_{T+}, such that $\Gamma \rhd \sigma : U_1$. Then*

$$\mathsf{EET} \vdash [\![\Gamma]\!] \rightarrow \Re([\![\sigma]\!]).$$

Proof The claim follows immediately from Remark 1.5.1 and the definition of $[\![\cdot]\!]$. $\square$

Theorem 3.4.2 *Let Γ be a context, T a preterm and σ a type expression of λ^p_T, such that $\Gamma \rhd T : \sigma$. Then*

$$\mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})+(\mathsf{Tot}) \vdash [\![\Gamma]\!] \rightarrow [\![T : \sigma]\!].$$

Proof We prove this theorem by induction on the derivation of $\Gamma \rhd T : \sigma$, reasoning in EET+(T-$\mathsf{I_N}$)+(Tot). We check, that the claim holds for each axiom of λ^p_T and that the claim is preserved, whenever a term typing rule of λ^p_T is applied. Throughout the proof, we assume, that ξ, τ, and τ' denote type expressions and B, L, M and N denote preterms of λ^p_T.

Axiom (0 *nat*): In this case $\Gamma = \emptyset$, $T \equiv 0$ and $\sigma \equiv nat$. By axiom (BON5), we have $\mathsf{0} \in \mathsf{N}$. Since $\Re(\mathsf{nat}, \mathsf{N})$, we also have $\mathsf{0} \,\dot{\in}\, \mathsf{nat}$. Therefore, $[\![0 : nat]\!]$, so also $\top \rightarrow [\![0 : nat]\!]$ and thus $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Axiom (succ): Then $\Gamma = \emptyset$, $T \equiv succ$ and $\sigma \equiv nat \rightarrow nat$. By axiom ($\mathsf{BON}$5), we have $(\forall x \in \mathsf{N})(\mathsf{s_N} x \in \mathsf{N})$, which means $\mathsf{s_N} \in (\mathsf{N} \rightarrow \mathsf{N})$. Since $\Re(\mathsf{nat}, \mathsf{N})$, we have $\mathsf{s_N} \,\dot{\in}\, \mathsf{imp}(\mathsf{nat}, \mathsf{nat})$ by Lemma 2.4.1 and so $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Axiom (*true bool*): Then $\Gamma = \emptyset$, $T \equiv true$ and $\sigma \equiv bool$. We have $\mathsf{1} \in \{\mathsf{0}, \mathsf{1}\}$, so $\mathsf{1} \,\dot{\in}\, \mathsf{bool}$, so $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Axiom (*false bool*): Analogous to the case of Axiom (*true bool*), with $\mathsf{0}$ instead of $\mathsf{1}$.

Rule (var): In this case $T \equiv x$ and $\Gamma = \Gamma', x : \sigma$, for some context Γ'. Since, by assumption, Γ *context* holds, we have $[\![\Gamma]\!] \equiv [\![\Gamma']\!] \wedge [\![x : \sigma]\!]$. Therefore, $[\![\Gamma]\!] \rightarrow [\![x : \sigma]\!]$ and thus $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Rule (add var): Then $\Gamma = \Gamma', v : A$, for some context Γ' and by assumption Γ *context* and $\Gamma' \rhd T : \sigma$. Therefore, by the induction hypothesis $[\![\Gamma']\!] \rightarrow [\![T : \sigma]\!]$ and thus also $[\![\Gamma']\!] \wedge [\![v : A]\!] \rightarrow [\![T : \sigma]\!]$ so $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Rule ($\rightarrow$ Intro): In this case $T \equiv \lambda x : \tau.M$ and $\sigma \equiv \tau \rightarrow \tau'$. By assumption we have $\Gamma, x : \tau \rhd M : \tau'$, $\Gamma \rhd \tau : U_1$ and $\Gamma \rhd \tau' : U_1$. So by induction hypothesis

$$[\![\Gamma]\!] \wedge [\![x : \tau]\!] \rightarrow [\![M : \tau']\!]\,, \tag{1}$$
$$[\![\Gamma]\!] \rightarrow [\![\tau : U_1]\!]\,, \tag{2}$$
$$[\![\Gamma]\!] \rightarrow [\![\tau' : U_1]\!]\,. \tag{3}$$

From (1) we get

$$[\![\Gamma]\!] \rightarrow ([\![x : \tau]\!] \rightarrow [\![M : \tau']\!]). \tag{4}$$

Since by assumption $\Gamma \rhd \tau : U_1$ and $\Gamma \rhd \tau' : U_1$, we know by Remark 1.5.1, that τ and τ' are simple type expressions. Therefore, from (4) we get

$$[\![\Gamma]\!] \rightarrow ((\hat{x} \,\dot{\in}\, [\![\tau]\!] \rightarrow ([\![M]\!] \,\dot{\in}\, [\![\tau']\!]))).$$

Since $\Gamma, x : \tau$ is a context, we have $(x : \xi) \notin \Gamma$ for any ξ, so $\hat{x} \notin FV([\![\Gamma]\!])$. We may thus use the quantifier rule, obtained in Lemma 2.2.1 to conclude

$$[\![\Gamma]\!] \rightarrow \forall \hat{x}((\hat{x} \,\dot{\in}\, [\![\tau]\!]) \rightarrow ([\![M]\!] \,\dot{\in}\, [\![\tau']\!])).$$

Now, by Lemma 2.3.1 $[\![M]\!] = (\lambda \hat{x}.\, [\![M]\!])\hat{x}$, so

$$[\![\Gamma]\!] \rightarrow \forall \hat{x}((\hat{x} \,\dot{\in}\, [\![\tau]\!]) \rightarrow ((\lambda \hat{x}.\, [\![M]\!])\hat{x}) \,\dot{\in}\, [\![\tau']\!]). \tag{5}$$

But (5) means

$$[\![\Gamma]\!] \rightarrow (\lambda \hat{x}.\, [\![M]\!]) \,\dot{\in}\, \mathsf{imp}([\![\tau]\!]\,, [\![\tau']\!])$$

and thus $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Rule ($\rightarrow$ Elim): Then $T \equiv MN$ and by assumption $\Gamma \rhd M : \tau \rightarrow \sigma$ and $\Gamma \rhd N : \tau$. So, by induction hypothesis

$$[\![\Gamma]\!] \rightarrow [\![M : \tau \rightarrow \sigma]\!]\,, \tag{6}$$
$$[\![\Gamma]\!] \rightarrow [\![N : \tau]\!]\,. \tag{7}$$

From (6) and Lemma 1.5.11 we get, that $\tau \rightarrow \sigma$ is simple, so

$$[\![\Gamma]\!] \rightarrow [\![M]\!] \,\dot{\in}\, \mathsf{imp}([\![\tau]\!]\,, [\![\sigma]\!])$$

an thus, assuming that x is a variable, such that $x \notin FV(\llbracket M \rrbracket)$

$$\llbracket \Gamma \rrbracket \rightarrow \forall x((x \mathrel{\dot\in} \llbracket \tau \rrbracket) \rightarrow (\llbracket M \rrbracket\, x \mathrel{\dot\in} \llbracket \sigma \rrbracket)).$$

Therefore,

$$\llbracket \Gamma \rrbracket \rightarrow ((\llbracket N \rrbracket \mathrel{\dot\in} \llbracket \tau \rrbracket) \rightarrow (\llbracket M \rrbracket\, \llbracket N \rrbracket \mathrel{\dot\in} \llbracket \sigma \rrbracket))$$

and with (7)

$$\llbracket \Gamma \rrbracket \rightarrow \llbracket M \rrbracket\, \llbracket N \rrbracket \mathrel{\dot\in} \llbracket \sigma \rrbracket\,,$$

which means the same as $\llbracket \Gamma \rrbracket \rightarrow \llbracket T : \sigma \rrbracket$.

Rule (Π Intro): In this case $T \equiv \lambda t : U_1.M$ and $\sigma \equiv \Pi t : U_1.\tau$. By assumption therefore $\Gamma, t : U_1 \rhd M : \tau$ and by induction hypothesis

$$\llbracket \Gamma \rrbracket \wedge \llbracket t : U_1 \rrbracket \rightarrow \llbracket M : \tau \rrbracket\,.$$

Therefore, by Theorem 3.4.1

$$\llbracket \Gamma \rrbracket \wedge \Re(\hat{t}) \rightarrow \llbracket M : \tau \rrbracket\,.$$

Again, since $\Gamma, t : U_1$ is a context, $t : A \notin \Gamma$ for any type expression or universe symbol A. Thus $\hat{t} \notin FV(\llbracket \Gamma \rrbracket)$ and so we may use the quantifier rule we proved in Lemma 2.2.1 to obtain

$$\llbracket \Gamma \rrbracket \rightarrow \forall \hat{t}(\Re(\hat{t}) \rightarrow \llbracket M : \tau \rrbracket),$$

Therefore, since $\llbracket T \rrbracket \equiv \llbracket M \rrbracket$, by Lemma 3.3.6 we get

$$\llbracket \Gamma \rrbracket \rightarrow \forall \hat{t}(\Re(\hat{t}) \rightarrow \llbracket T : \tau \rrbracket)$$

and thus $\llbracket \Gamma \rrbracket \rightarrow \llbracket T : \sigma \rrbracket$.

Rule (Π Elim): Then $T \equiv M\xi$ and by assumption $\Gamma \rhd M : \Pi t : U_1.\sigma$ and $\Gamma \rhd \xi : U_1$. Therefore, by induction hypothesis

$$\llbracket \Gamma \rrbracket \rightarrow \forall \hat{t}(\Re(\hat{t}) \rightarrow \llbracket M : \sigma \rrbracket) \tag{8}$$

and by Theorem 3.4.1

$$\llbracket \Gamma \rrbracket \rightarrow \Re(\llbracket \xi \rrbracket). \tag{9}$$

From (8) we derive

$$\llbracket \Gamma \rrbracket \rightarrow (\Re(\llbracket \xi \rrbracket) \rightarrow \llbracket M : \sigma \rrbracket \left[\llbracket \xi \rrbracket / \hat{t}\right]).$$

Now with Lemma 3.3.3 we obtain

$$\llbracket \Gamma \rrbracket \rightarrow (\Re(\llbracket \xi \rrbracket) \rightarrow \llbracket M : [\xi/t]\, \sigma \rrbracket)$$

and using (9)

$$\llbracket \Gamma \rrbracket \rightarrow \llbracket M : [\xi/t]\, \sigma \rrbracket\,.$$

Therefore, since $\llbracket T \rrbracket \equiv \llbracket M \rrbracket$ we have $\llbracket T : [\xi/t]\, \sigma \rrbracket$ with Lemma 3.3.6.

Rule (let): Then $T \equiv (\mathsf{let}\ x : \xi = N\ \mathsf{in}\ M)$ and by assumption $\Gamma \rhd \sigma : U_1$, $\Gamma, x : \xi \rhd M : \sigma$ and $\Gamma \rhd N : \xi$. So, by induction hypothesis

$$\text{(10)} \qquad [\![\Gamma]\!] \wedge [\![x : \xi]\!] \rightarrow [\![M : \sigma]\!]\,,$$
$$\text{(11)} \qquad [\![\Gamma]\!] \rightarrow [\![N : \xi]\!]\,.$$

By (10) we obtain

$$[\![\Gamma]\!] \rightarrow ([\![x : \xi]\!] \rightarrow [\![M : \sigma]\!]).$$

Therefore, with the assumption that $\Gamma \rhd \sigma : U_1$ and Remark 1.5.1 we have

$$[\![\Gamma]\!] \rightarrow ([\![x : \xi]\!] \rightarrow [\![M]\!] \mathrel{\dot\in} [\![\sigma]\!])$$

and by Lemma 2.3.1

$$[\![\Gamma]\!] \rightarrow ([\![x : \xi]\!] \rightarrow (\lambda\hat{x}.\, [\![M]\!])\hat{x} \mathrel{\dot\in} [\![\sigma]\!]).$$

Since $\Gamma, x : \xi$ is a context, $x : \tau \notin \Gamma$ for any type expression τ and thus $\hat{x} \notin FV([\![\Gamma]\!])$. We may therefore use the quantifier rule we proved in Lemma 2.2.1 to obtain

$$[\![\Gamma]\!] \rightarrow \forall\hat{x}([\![x : \xi]\!] \rightarrow (\lambda\hat{x}.\, [\![M]\!])\hat{x} \mathrel{\dot\in} [\![\sigma]\!])$$

and so, by specialising with $[\![N]\!]$ and Lemma 3.3.1 we get

$$[\![\Gamma]\!] \rightarrow ([\![x : \xi]\!]\, [[\![N]\!] / \hat{x}] \rightarrow ((\lambda\hat{x}.\, [\![M]\!])\hat{x})\, [[\![N]\!] / \hat{x}] \mathrel{\dot\in} [\![\sigma]\!]).$$

By Lemma 3.3.5 it follows, that

$$[\![\Gamma]\!] \rightarrow ([\![N : \xi]\!] \rightarrow (\lambda\hat{x}.\, [\![M]\!])\, [\![N]\!] \mathrel{\dot\in} [\![\sigma]\!]\,.$$

Then by (11) we have

$$[\![\Gamma]\!] \rightarrow (\lambda\hat{x}.\, [\![M]\!])\, [\![N]\!] \mathrel{\dot\in} [\![\sigma]\!]$$

and so $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Rule (rec): Then $T \equiv RLMN$ and by assumption $\Gamma \rhd L : \sigma$, $\Gamma \rhd M : \sigma \rightarrow (nat \rightarrow \sigma)$, $\Gamma \rhd N : nat$, so by induction hypothesis

$$\text{(12)} \qquad [\![\Gamma]\!] \rightarrow [\![L : \sigma]\!]\,,$$
$$\text{(13)} \qquad [\![\Gamma]\!] \rightarrow [\![M : \sigma \rightarrow (nat \rightarrow \sigma)]\!]\,,$$
$$\text{(14)} \qquad [\![\Gamma]\!] \rightarrow [\![N : nat]\!]\,.$$

From $\Gamma \rhd M : \sigma \rightarrow (nat \rightarrow \sigma)$ it follows by Lemma 1.5.11, that $\Gamma \rhd \sigma : U_1$. Therefore, by Theorem 3.4.1,

$$\text{(15)} \qquad [\![\Gamma]\!] \rightarrow \Re([\![\sigma]\!]).$$

By (12), (13) and (14) and Remark 1.5.1 we then obtain

$$(16)\qquad [\![\Gamma]\!] \rightarrow [\![L]\!] \mathrel{\dot\in} [\![\sigma]\!]\,,$$
$$(17)\qquad [\![\Gamma]\!] \rightarrow [\![M]\!] \mathrel{\dot\in} \mathsf{imp}([\![\sigma]\!]\,, \mathsf{imp}(\mathsf{nat}, [\![\sigma]\!])),$$
$$(18)\qquad [\![\Gamma]\!] \rightarrow [\![N]\!] \mathrel{\dot\in} \mathsf{nat}.$$

Now using Lemma 2.4.5 on (15), (16) and (17), we conclude

$$[\![\Gamma]\!] \rightarrow \forall n(n \mathrel{\dot\in} \mathsf{N} \rightarrow (\mathsf{r}\,[\![L]\!]\,[\![M]\!]\,n \mathrel{\dot\in} [\![\sigma]\!])).$$

Specialising with $[\![N]\!]$, we get

$$[\![\Gamma]\!] \rightarrow ([\![N]\!] \mathrel{\dot\in} \mathsf{nat} \rightarrow \mathsf{r}\,[\![L]\!]\,[\![M]\!]\,[\![N]\!] \mathrel{\dot\in} [\![\sigma]\!]).$$

So by (18)

$$[\![\Gamma]\!] \rightarrow \mathsf{r}\,[\![L]\!]\,[\![M]\!]\,[\![N]\!] \mathrel{\dot\in} [\![\sigma]\!]\,,$$

which means $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Rule (case): In this case $T \equiv DMNB$ and by assumption $\Gamma \rhd M : \sigma$, $\Gamma \rhd N : \sigma$ and $\Gamma \rhd B : bool$, so by induction hypothesis

$$(19)\qquad [\![\Gamma]\!] \rightarrow [\![M : \sigma]\!]\,,$$
$$(20)\qquad [\![\Gamma]\!] \rightarrow [\![N : \sigma]\!]\,,$$
$$(21)\qquad [\![\Gamma]\!] \rightarrow [\![B]\!] \mathrel{\dot\in} \mathsf{bool}.$$

We have

$$[\![\Gamma]\!] \rightarrow [\![DMNB]\!] = \mathsf{d_B}\,[\![M]\!]\,[\![N]\!]\,[\![B]\!]\,.$$

By (21) we know, that either $[\![B]\!] = \mathsf{1}$ or $[\![B]\!] = \mathsf{0}$.

Case 1) $[\![B]\!] = \mathsf{1}$: Then $\mathsf{d_B}\,[\![M]\!]\,[\![N]\!]\,[\![B]\!] = [\![M]\!]$ and so $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$ by (19).

Case 2) $[\![B]\!] = \mathsf{0}$: Then $\mathsf{d_B}\,[\![M]\!]\,[\![N]\!]\,[\![B]\!] = [\![N]\!]$ and so $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$ by (20).

Therefore, our claim holds for all axioms and rules of λ^p_T, which concludes the proof. □

Theorem 3.4.3 *Let Γ be a context, T a preterm and σ a type expression of λ^p_{T+}, such that $\Gamma \rhd T : \sigma$. Then*

$$\mathsf{EET}{+}(\mathsf{F\text{-}I_N}){+}(\mathsf{Tot}) \vdash [\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]\,.$$

Proof As in the proof of Theorem 3.4.2, we also prove this claim by induction on the derivation of $\Gamma \rhd T : \sigma$. However, we need to reconsider only those rules, which contain arrow-types. The treatment of the other rules is literally identical to the one given in the proof of Theorem 3.4.2. All reasoning is done in $\mathsf{EET}{+}(\mathsf{F\text{-}I_N}){+}(\mathsf{Tot})$. Throughout the proof, we will assume that τ and τ' denote type expressions and L, M and N preterms of λ^p_{T+}.

Rule (full $\rightarrow$ Intro): In this case $T \equiv \lambda x : \tau.M$ and by assumption $\Gamma, x : \tau \rhd M : \tau'$

Case 1) Both τ, τ' are simple type expressions: Then the proof is the same as for the rule ($\rightarrow$ Intro) in Theorem 3.4.2.

Case 2) τ or τ' is not a simple type expression: Then by induction hypothesis we have

$$[\![\Gamma]\!] \wedge [\![x : \tau]\!] \rightarrow [\![M : \tau']\!],$$

which is equivalent to

$$[\![\Gamma]\!] \rightarrow ([\![x : \tau]\!] \rightarrow [\![M : \tau']\!]).$$

Now, by Lemma 2.3.1 $[\![M]\!] = (\lambda\hat{x}. [\![M]\!])\hat{x} = [\![(\lambda x : \tau.M)x]\!]$, so by Lemma 3.3.6

$$[\![\Gamma]\!] \rightarrow ([\![x : \tau]\!] \rightarrow [\![(\lambda x : \tau.M)x : \tau']\!]).$$

Since $\Gamma, x : \tau$ is a context, $(x : \gamma) \notin \Gamma$ for any type expression γ and so $\hat{x} \notin FV([\![\Gamma]\!])$. By the quantifier rule obtained in Lemma 2.2.1, we thus conclude

$$[\![\Gamma]\!] \rightarrow \forall\hat{x}([\![x : \tau]\!] \rightarrow [\![(\lambda x : \tau.M)x : \tau']\!]),$$

which means $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Rule ($\rightarrow$ Elim): In this case $T \equiv MN$ and by assumption $\Gamma \rhd M : \tau \rightarrow \tau'$ and $\Gamma \rhd N : \tau$.

Case 1) $\tau \rightarrow \tau'$ is a simple type expression: Then the proof is the same as the one for Rule ($\rightarrow$ Elim) in Theorem 3.4.2.

Case 2) $\tau \rightarrow \tau'$ is not a simple type expression: By induction hypothesis we have

$$(1) \qquad [\![\Gamma]\!] \rightarrow \forall\hat{s}([\![s : \tau]\!] \rightarrow [\![Ms : \tau']\!]),$$
$$(2) \qquad [\![\Gamma]\!] \rightarrow [\![N : \tau]\!],$$

where s is an individual variable of λ^p_{T+}, such that $s \notin FV(M)$. So specialising (1), we obtain

$$[\![\Gamma]\!] \rightarrow (([\![s : \tau]\!] [[\![N]\!] / \hat{s}]) \rightarrow ([\![Ms : \tau']\!] [[\![N]\!] / \hat{s}])).$$

Therefore, by Lemma 3.3.5 and the fact that $s \notin FV(M)$

$$[\![\Gamma]\!] \rightarrow ([\![N : \tau]\!] \rightarrow [\![MN : \tau']\!]),$$

so using (2), we get

$$[\![\Gamma]\!] \rightarrow [\![MN : \tau']\!]$$

and thus $[\![\Gamma]\!] \rightarrow [\![T : \sigma]\!]$.

Rule (rec): Then $T \equiv RLMN$ and by assumption $\Gamma \rhd L : \sigma$, $\Gamma \rhd M : \sigma \rightarrow (nat \rightarrow \sigma)$, $\Gamma \rhd N : nat$.

Case 1) σ is a simple type: Then the proof is the same as the one for Rule (rec) in Theorem 3.4.2.

Case 2) σ is not a simple type: Then by induction hypothesis we have

$$(3) \qquad [\![\Gamma]\!] \to [\![L : \sigma]\!]\,,$$
$$(4) \qquad [\![\Gamma]\!] \to \forall \hat{s}([\![s : \sigma]\!] \to \forall \hat{n}(\hat{n} \,\dot{\in}\, \mathsf{nat} \to [\![Msn : \sigma]\!])),$$
$$(5) \qquad [\![\Gamma]\!] \to [\![N]\!] \,\dot{\in}\, \mathsf{nat},$$

where s and n are individual variables of λ^p_{T+}, such that $s \notin FV(M)$ and $n \notin FV(Ms)$. Furthermore, we assume n to be chosen in a way, that $(n : \gamma) \notin \Gamma$ for any type expression γ and $n \notin FV(L)$. We aim to use the axiom (F-$\mathsf{I_N}$) and first show, that

$$(*) \qquad [\![\Gamma]\!] \to [\![RLMn : \sigma]\!]\,[\mathsf{0}/\hat{n}]$$

holds. By Lemma 3.3.5 we have

$$[\![RLMn : \sigma]\!]\,[\mathsf{0}/\hat{n}] \leftrightarrow [\![[0/n]\,RLMn : \sigma]\!]\,.$$

Furthermore, we have

$$[\![[0/n]\,RLMn]\!] = [\![RLM0]\!] = \mathsf{r}\,[\![L]\!]\,[\![M]\!]\,\mathsf{0} = [\![L]\!]\,.$$

So by Lemma 3.3.6 and (3) the formula $(*)$ holds. We now show, that

$$(**) \qquad [\![\Gamma]\!] \to (\forall \hat{n} \in \mathsf{N})(([\![RLMx : \sigma]\!]\,[\hat{n}/\hat{x}]) \to ([\![RLMx : \sigma]\!]\,[\hat{n}'/\hat{x}]))$$

also holds. Again by Lemma 3.3.5, we have

$$[\![RLMx : \sigma]\!]\,[\hat{n}'/\hat{x}] \leftrightarrow [\![[succn/x]\,RLMx : \sigma]\!]\,.$$

Now we may specialise (4) with $\mathsf{r}\,[\![L]\!]\,[\![M]\!]\,\hat{n}$ to get

$$[\![\Gamma]\!] \to ([\![s : \sigma]\!]\,[\mathsf{r}\,[\![L]\!]\,[\![M]\!]\,\hat{n}/\hat{s}] \to \forall \hat{m}(\hat{m} \,\dot{\in}\, \mathsf{nat} \to [\![Msm : \sigma]\!]\,[\mathsf{r}\,[\![L]\!]\,[\![M]\!]\,\hat{n}/\hat{s}])),$$

where m is a variable of λ^p_{T+}, not free in $RLMn$. This, by Lemma 3.3.5, yields

$$[\![\Gamma]\!] \to ([\![RLMn : \sigma]\!] \to \forall \hat{m}(\hat{m} \,\dot{\in}\, \mathsf{nat} \to [\![M(RLMn)m : \sigma]\!])).$$

We specialise the universally quantified part with $\hat{n}$, giving us

$$[\![\Gamma]\!] \to ([\![RLMn : \sigma]\!] \to (\hat{n} \,\dot{\in}\, \mathsf{nat} \to [\![M(RLMn)n : \sigma]\!])),$$

which implies

$$[\![\Gamma]\!] \to (\hat{n} \,\dot{\in}\, \mathsf{nat} \to ([\![RLMn : \sigma]\!] \to [\![M(RLMn)n : \sigma]\!]))$$

and since $\hat{n} \notin FV(\llbracket\Gamma\rrbracket)$, we may universally quantify this to

$$\llbracket\Gamma\rrbracket \rightarrow \forall\hat{n}(\hat{n} \,\dot{\in}\, \mathsf{nat} \rightarrow (\llbracket RLMn : \sigma\rrbracket \rightarrow \llbracket M(RLMn)n : \sigma\rrbracket)),$$

which implies

$$(6) \qquad \llbracket\Gamma\rrbracket \rightarrow (\forall\hat{n} \in \mathsf{N})(\llbracket RLMn : \sigma\rrbracket \rightarrow \llbracket M(RLMn)n : \sigma\rrbracket)).$$

Now consider the following equalities

$$\begin{aligned}\llbracket M(RLMn)n\rrbracket &= \llbracket M\rrbracket\,(\mathsf{r}\,\llbracket L\rrbracket\,\llbracket M\rrbracket\,\hat{n})\hat{n} = \mathsf{r}\,\llbracket L\rrbracket\,\llbracket M\rrbracket\,\hat{n}' = (\mathsf{r}\,\llbracket L\rrbracket\,\llbracket M\rrbracket\,\hat{x})\,[\hat{n}'/\hat{x}] \\ &= \llbracket [succn/x]\,RLMx\rrbracket\,.\end{aligned}$$

So from (6) we obtain with Lemma 3.3.6

$$\llbracket\Gamma\rrbracket \rightarrow (\forall\hat{n} \in \mathsf{N})(\llbracket [n/x]\,RLMx : \sigma\rrbracket \rightarrow \llbracket [succn/x]\,RLMx : \sigma\rrbracket)),$$

which is equivalent to $(**)$ by Lemma 3.3.5. Thus, by the axiom $(\mathsf{F}\text{-}\mathsf{I}_{\mathsf{N}})$ we may conclude

$$\llbracket\Gamma\rrbracket \rightarrow (\forall\hat{n} \in \mathsf{N})\,\llbracket RLMn : \sigma\rrbracket$$

and therefore, by (5) and Lemma 3.3.5

$$\llbracket\Gamma\rrbracket \rightarrow \llbracket RLMN : \sigma\rrbracket\,,$$

which means, that $\llbracket\Gamma\rrbracket \rightarrow \llbracket T : \sigma\rrbracket$ holds.

So the claim holds for all axioms and rules of λ^p_{T+} and our proof is complete. □

Theorem 3.4.4 *Let Γ be a context, T and S preterms and σ a type expression of λ^p_{T+}, such that $\Gamma \rhd T = S : \sigma$. Then*

$$\mathsf{EET}{+}(\mathsf{Tot}){+}(\mathsf{Ext}) \vdash \llbracket T\rrbracket = \llbracket S\rrbracket\,.$$

Proof We prove this claim by induction on the derivation of $\Gamma \rhd T = S : \sigma$. That is to say, we must check, that the claim is preserved, whenever an equational rule of λ^p_{T+} is applied. We take all reasoning to be done in the system $\mathsf{EET}{+}(\mathsf{Tot}){+}(\mathsf{Ext})$.

Rule (add var$_=$): So we have $\Gamma = \Gamma', x : \tau$ for some context Γ' and $\Gamma' \rhd T = S : \sigma$. Thus, by the induction hypothesis the claim follows at once.

Rule (ref): Then we have $T \equiv S$. This case follows trivially, by axiom (E1) of LPT.

Rule (sym): Then we have $\Gamma \rhd S = T : \sigma$ and by induction hypothesis $\llbracket S\rrbracket = \llbracket T\rrbracket$, so the claim follows trivially by axiom (E2) of LPT.

Rule (trans): So $\Gamma \rhd T = K : \sigma$ and $\Gamma \rhd K = S : \sigma$ and by the induction hypothesis

$$[\![T]\!] = [\![K]\!] \wedge [\![K]\!] = [\![S]\!] .$$

Then the claim follows trivially by axiom (E3) of LPT.

Rule (ξ): In this case $S \equiv \lambda x : \tau.M$ and $T \equiv \lambda x : \tau.N$. By assumption we have $\Gamma, x : \tau \rhd M = N : \xi$ and thus by the induction hypothesis $[\![M]\!] = [\![N]\!]$. It follows by Lemma 2.3.3 that $\lambda \hat{x}. [\![M]\!] = \lambda \hat{x}. [\![N]\!]$, which means $[\![S]\!] = [\![T]\!]$.

Rule (ν): Then $S \equiv M_1 N_1$ and $T \equiv M_2 N_2$. By assumption we have $\Gamma \rhd M_1 = M_2 : \tau \to \xi$ and $\Gamma \rhd N_1 = N_2 : \tau$. So by the induction hypothesis $[\![M_1]\!] = [\![M_2]\!]$ and $[\![N_1]\!] = [\![N_2]\!]$. So by axiom (E5) of LPT we get $[\![M_1]\!]\,[\![N_1]\!] = [\![M_2]\!]\,[\![N_2]\!]$, which means $[\![S]\!] = [\![T]\!]$.

Rule (α): In this case $T \equiv \lambda x : \sigma.M$ and $S \equiv \lambda y : \sigma.\,[y/x]\,M$. We have $[\![T]\!] = \lambda \hat{x}. [\![M]\!]$ and by Lemma 3.3.4 $[\![S]\!] = \lambda \hat{y}.([\![M]\!]\,[\hat{y}/\hat{x}])$. Therefore, by Lemma 2.3.2, it follows that $[\![T]\!] = [\![S]\!]$.

Rule (β): Then $T \equiv (\lambda x : \sigma.M)N$ and $S \equiv [N/x]\,M$. By Lemma 2.3.1 we have $[\![T]\!] = (\lambda \hat{x}. [\![M]\!])\,[\![N]\!] = [\![M]\!]\,[[\![N]\!]\,/\hat{x}]$ and by Lemma 3.3.4 $[\![S]\!] = [\![[N/x]\,M]\!] = [\![M]\!]\,[[\![N]\!]\,/\hat{x}]$. So we have $[\![T]\!] = [\![S]\!]$.

Rule (η): In this case $T \equiv \lambda x : \sigma.(Mx)$ and $S \equiv M$. By Lemma 2.3.4 we have $[\![T]\!] = \lambda \hat{x}. [\![Mx]\!] = \lambda \hat{x}.([\![M]\!]\,\hat{x}) = [\![M]\!]$. So $[\![S]\!] = [\![T]\!]$ holds.

Rule (α_Π): Then $S \equiv \lambda t : U_1.M$ and $T \equiv \lambda s : U_1.\,[s/t]\,M$. We have $[\![S]\!] = [\![\lambda t : U_1.M]\!] = [\![M]\!]$ and

$$[\![T]\!] = [\![\lambda s : U_1.\,[s/t]\,M]\!] = [\![[s/t]\,M]\!] \overset{\text{Lemma 3.3.4}}{=} [\![M]\!]\,\left[\hat{s}/\hat{t}\right] \overset{\text{Lemma 3.3.1}}{=} [\![M]\!] .$$

So $[\![S]\!] = [\![T]\!]$ holds.

Rule (β_Π): In this case $S \equiv (\lambda t : U_1.M)\tau$ and $T \equiv [\tau/t]\,M$. We have

$$[\![S]\!] = [\![(\lambda t : U_1.M)\tau]\!] = [\![\lambda t : U_1.M]\!] = [\![M]\!]$$

and by Lemma 3.3.1 $[\![T]\!] = [\![[\tau/t]\,M]\!] = [\![M]\!]$, so $[\![T]\!] = [\![S]\!]$ holds.

Rule (η_Π): Then $S \equiv \lambda t : U_1.Mt$ and $T \equiv M$. We have

$$[\![S]\!] = [\![\lambda t : U_1.Mt]\!] = [\![Mt]\!] = [\![M]\!] = [\![T]\!] ,$$

so the claim holds trivially.

Rule (ξ_Π): In this case $T \equiv \lambda t : U_1.M$ and $S \equiv \lambda t : U_1.N$ and by assumption $\Gamma \rhd M = N : \sigma$. So by the induction hypothesis $[\![M]\!] = [\![N]\!]$. Therefore, since $[\![T]\!] = [\![M]\!]$ and $[\![S]\!] = [\![N]\!]$, we trivially have $[\![T]\!] = [\![S]\!]$.

Rule (ν_Π): Then $S \equiv M\tau$ and $T \equiv N\tau$ and since $[\![S]\!] = [\![M]\!]$ and $[\![T]\!] = [\![N]\!]$, the claim follows directly from the induction hypothesis, as in the case of the rule (ξ_Π).

Rule (let $_=$): In this case $T \equiv (\mathsf{let}\ x : \tau = N\ \mathsf{in}\ M)$ and $S \equiv [N/x]\,M$. We have $[\![T]\!] = (\lambda\hat{x}.\,[\![M]\!])\,[\![N]\!] = [\![M]\!]\,[[\![N]\!]\,/\hat{x}]$ and by Lemma 3.3.4 $[\![S]\!] = [\![[N/x]\,M]\!] = [\![M]\!]\,[[\![N]\!]\,/\hat{x}]$. So $[\![T]\!] = [\![S]\!]$ holds.

Rule (case$_=$ $true$): Then $S \equiv DMNtrue$ and $T \equiv M$. By Lemma 2.3.6 we have $[\![S]\!] = \mathsf{d_B}\,[\![M]\!]\,[\![N]\!]\,\mathsf{1} = [\![M]\!]$, so $[\![S]\!] = [\![T]\!]$ holds.

Rule (case$_=$ $false$): This case also follows by Lemma 2.3.6, in a manner completely analogous to the case of rule (case$_=$ $true$).

Rule (rec$_=$ 0): Then $S \equiv RLM0$ and $T \equiv L$. Therefore, by Lemma 2.3.7

$$[\![S]\!] = \mathsf{r}\,[\![L]\!]\,[\![M]\!]\,\mathsf{0} = [\![L]\!]$$

and so $[\![S]\!] = [\![T]\!]$ holds.

Rule (rec$_=$ $succ$): In this case $S \equiv RLM(succN)$ and $T \equiv M(RLMN)N$. Again by Lemma 2.3.7 we have

$$[\![S]\!] = \mathsf{r}\,[\![L]\!]\,[\![M]\!]\,([\![N]\!]') = [\![M]\!]\,(\mathsf{r}\,[\![L]\!]\,[\![M]\!]\,[\![N]\!])\,[\![N]\!] = [\![T]\!]\,.$$

So $[\![S]\!] = [\![T]\!]$ also holds.

Therefore, the claim holds in all cases. □

Conclusions

Results

The results of this thesis may be stated in various ways. On one hand, we have demonstrated how predicative polymorphism can be simulated naturally in an untyped logical framework. On the other hand and perhaps more importantly, our results can also be used to determine the proof-theoretic strength of predicative polymorphism. In concluding, we elaborate slightly further on this aspect of our results. The notion of proof-theoretic strength, which is used when dealing with purely functional systems like λ^p_T and λ^p_{T+} is that of the *provably total functions*. By the provably total functions of λ^p_T and λ^p_{T+}, we mean those terms T for which we can derive $\emptyset \rhd T : nat \rightarrow nat$ in either λ^p_T or λ^p_{T+} respectively. Similarly, by the provably total functions of a theory A of explicit mathematics, we mean those closed terms f, for which we can prove $\forall x(x \,\dot{\in}\, \mathsf{nat} \rightarrow fx \,\dot{\in}\, \mathsf{nat})$ in A. In this way, provably total functions may be defined in most logical theories and it can be shown, that the concept of provably total functions usually coincides with other notions of proof-theoretic strength, where available. We will thus say, that a system of predicative polymorphism is proof-theoretically equivalent to a theory of explicit mathematics, if and only if there is a one-to-one correspondence of the provably total functions. The embedding theorems stated in this thesis automatically yield one direction of this correspondence.

Theorems 3.4.2 and 3.4.4 imply, that the system λ^p_T may be embedded into the theory $\mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})+(\mathsf{Tot})+(\mathsf{Ext})$ of explicit mathematics. Together with Remark 1.3.1 and Theorem 2.4.1, the situation can then be depicted as follows

$$\lambda^p_T \overset{\text{Theorems 3.4.2, 3.4.4}}{\rightsquigarrow} \mathsf{EET}+(\mathsf{T}\text{-}\mathsf{I_N})+(\mathsf{Tot})+(\mathsf{Ext}) \overset{\text{Theorem 2.4.1}}{\equiv} \mathsf{PA} \rightsquigarrow \text{System } T \overset{\text{Remark 1.3.1}}{\subseteq} \lambda^p_T,$$

where the second wavy arrow refers to a result by Gödel, well known as the *Dialectica interpretation*, which is described for example by Avigad and Feferman [AF98]. In sum, we may thus conclude that the system λ^p_T of predicative polymorphism is of the same proof-theoretic strength as PA. It is also well known, that PA and System T are proof-theoretically equivalent, which leads us to the conclusion that λ^p_T is conservative over System T. Therefore, from a computational point of view, nothing is gained from adding such a weak form of polymorphism to System T. However, from an engineering point of view there is certainly a gain. The polymorphism of λ^p_T is still useful for factoring out shared behaviour and thereby avoiding the duplication of "program code". To appreciate this, one may consider for example the identity function in both System T and λ^p_T. In the latter the term $\lambda t : U_1.\lambda x : t.x$ is well-typed and may be applied to any simple type σ to yield $\lambda x : \sigma.x$. We thus only need one definition of the identity function to cover all simple types. This is not the case in System T, where a separate identity $\lambda x : \sigma.x$ must be defined for every simple type σ.

The result obtained for λ^p_{T+} is weaker. Theorems 3.4.3 and 3.4.4 imply that the system λ^p_{T+} may be embedded into $\mathsf{EET}+(\mathsf{F\text{-}I_N})+(\mathsf{Tot})+(\mathsf{Ext})$. Thus, together with Theorem 2.4.1, we have the situation

$$\lambda^p_{T+} \overset{\text{Theorems 3.4.3, 3.4.4}}{\rightsquigarrow} \mathsf{EET}+(\mathsf{F\text{-}I_N})+(\mathsf{Tot})+(\mathsf{Ext}) \overset{\text{Theorem 2.4.1}}{\equiv} \mathsf{\Pi^0_\infty{-}CA}.$$

Therefore the proof-theoretic strength of $\mathsf{EET}+(\mathsf{F\text{-}I_N})+(\mathsf{Tot})+(\mathsf{Ext})$ and $\mathsf{\Pi^0_\infty{-}CA}$ provides an upper bound for the strength of λ^p_{T+}. In this case however, an exact correspondence does not follow immediately, since we do not have an interpretation result for $\mathsf{\Pi^0_\infty{-}CA}$, analogous to the one for System T.

Further work

From the results of this thesis, we obtain two interesting topics for further work. The first one addresses the problem of finding a lower bound for the strength of λ^p_{T+}. In fact, we may state it as the following conjecture.

Conjecture 3.4.1 λ^p_{T+} *is proof-theoretically equivalent to* $\mathsf{EET}+(\mathsf{F\text{-}I_N})+(\mathsf{Tot})+(\mathsf{Ext})$ *and* $\mathsf{\Pi^0_\infty{-}CA}$.

The most direct way to prove this conjecture would be to find a functional interpretation of $\mathsf{\Pi^0_\infty{-}CA}$ into a subsystem of λ^p_{T+}, analogous to the interpretation of PA into System T. In that case, the claim would hold by the same reasoning. However, such a functional interpretation would exceed the scope of this thesis.

The second topic adresses the use of the axiom (Tot) to prove the embedding results. We may state it as the following conjecture.

Conjecture 3.4.2 *The axiom* (Tot) *is not needed in the results of this thesis.*

A proof of this conjecture would most certainly go along the following lines: In λ^p_T and λ^p_{T+} we only deal with terms that are well-typed. That is to say, if a term T appears in a derivation of λ^p_T or λ^p_{T+}, then somewhere in that derivation we have $\Gamma \rhd T : \sigma$ for some context Γ and some type expression σ. By inspection of the definition of the interpretation mapping $[\![\cdot]\!]$, we may see that the judgement $T : \sigma$ is always interpreted as some formula of explicit mathematics, containing the term $[\![T]\!]$. By an inductive argument on the strictness axioms of EET, we may immediately conclude that $[\![T]\!]\downarrow$ must hold. Therefore, the interpretation of any term λ^p_T or λ^p_{T+} is always defined in explicit mathematics. Thus, for our application it should be sufficient to use the slightly different, partial definition for λ-abstraction in place of Definition 2.3.3 and prove the subsequent lemmata as well as the recursion-theorem under the assumption, that the term under consideration is defined.

Bibliography

[AF98] J. Avigad and S. Feferman. Gödel's functional ("dialectica") interpretation. In S. R. Buss, editor, *Handbook of Proof Theory*, chapter V, pages 337–405. Elsevier, 1998.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.

[Fef75] S. Feferman. A language and axioms for explicit mathematics. In A. Dold and B. Eckmann, editors, *Algebra and Logic*, volume 450 of *Lecture Notes in Mathematics*, pages 87–139. Springer, 1975.

[Fef79] S. Feferman. Constructive theories of functions and classes. In M. Boffa, D. van Dalen, and K. McAloon, editors, *Logic Colloquium '78*, Studies in Logic and the Foundations of Mathematics, pages 159–224. North-Holland, 1979.

[Fef90] S. Feferman. Polymorphic typed lambda-calculi in a type-free axiomatic framework. *Contemporary Mathematics*, 106, 1990.

[Fef92] S. Feferman. Logics for termination and correctness of functional programs. In Y. Moschovakis, editor, *Logic from Computer Science*, volume 21 of *Mathematical Sciences Research Institute Publications*. Springer, 1992.

[FJ93] S. Feferman and G. Jäger. Systems of explicit mathematics with non-constructive μ-operator. *Annals of Pure and Applied Logic*, 65:243–263, 1993.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[Jäg88] G. Jäger. Induction in the elementary theory of types and names. In E. Börger, H. Kleine Büning, and M. M. Richter, editors, *Computer Science Logic '87*, volume 329 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 1988.

[JS95] G. Jäger and T. Strahm. Totality in applicative theories. *Annals of Pure and Applied Logic*, 74, 1995.

[Mar93] M. Marzetta. *Predicative Theories of Types and Names*. PhD thesis, Institut für Informatik und angewandte Mathematik, University of Berne, 1993.

[Mit90] J. C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier, 1990.

[Mit96] J. C. Mitchell. *Foundations for Programming Languages.* Foundations of Computing series. MIT Press, 1996.

[Stu01] T. Studer. *Object-Oriented Programming in Explicit Mathematics: Towards the Mathematics of Objects.* PhD thesis, Institut für Informatik und angewandte Mathematik, University of Berne, 2001.