Inductive Definition in Type Theory

Paul Francis Mendler
Ph.D. Thesis

87-870
September 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

INDUCTIVE DEFINITION IN TYPE THEORY

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Paul Francis Mendler

January 1988

Inductive Definition in Type Theory

Paul Francis Mendler, Ph.D.

Cornell University 1988

Type theories can provide a foundational account of constructive mathematics, and for the computer scientist, they can also serve the dual roles of specification and programming languages. In the search for natural and expressive extensions to the NUPRL type theory, we are lead to consider forms of inductive and co-inductive definition.

We realize these notions through the addition of two new type constructors, denoted μ and ν. This represents a step towards a more expressive theory, without adopting a completely impredicative notion of type. With these constructors we can define all the common inductive data types, from natural numbers to infinite trees. Through the propositions-as-types principle, these type constructors yield inductively defined propositions. The induction principle associated with the μ types lets us define well-founded recursive functions, and dual principle for the ν types lets us inductively define their "infinite" elements. We present another induction principle for the μ types which takes advantage of the information hiding properties of

the $\{\_ \mid \_\}$ type, and can be used to define an unbounded search operator, or more generally, to compute not with elements of the μ type, but under the assumption of its inhabitation.

After presenting the proof rules for these new type constructors we give a semantic account, from which intuitionistic consistency is a consequence. First, we consider the the question of inductive types in the simpler setting of the second-order lambda calculus, where we prove a strong normalization property. We also consider typing terms in the presence of type constraints, and present a condition on the constraints (of polynomial complexity in the size of the constraints) for determining if the terms will be strongly normalizable or there will be a diverging typed term. Second, we develop a semantic account of the basic type theory, then relativize it to account for the impredicativity inherent in the definition of the new type constructors. We also show how this model can justify other impredicative type constructors, such as an impredicative type abstraction operation.

Biographical Sketch

Paul Francis "Nax" Mendler was born in Pittsburgh, Pennsylvania on November 29, 1960. Never one to burden others with his childhood memories, he entered Carnegie-Mellon University in the fall of 1978, and four years later graduated with a B.S. with University Honors in Applied Mathematics and, through some confusion, with the middle name "Fibonacci." His attachment to his Alma Mater is evident in the return of his first alumni mailing, indicating, "addressee deceased."

Nax entered the graduate program at Cornell University in the fall of 1982. On September 3, 1985 he donned an airconditioning repairman's uniform and intoned, "*andhasya dīpo vidyā*," and so entered the doctoral program, but because of an unchecked box on his "Examination for Admission to Doctoral Candidacy" form, was *not* awarded a Master of Science degree. He was, however, presented with a complementary egg salad sandwich.

After enjoying a full five years of residence in Ithaca, Nax is preparing to leave in September, 1987, to study at the University of Manchester, on a NATO Postdoctoral Fellowship in Science.

To my parents, Oliver and Blanche.

Acknowledgements

At Cornell, I've been most fortunate to work in the PRL group, a diverse and energetic collective, lead by Robert Constable. As my advisor, his inspirations and guidance were major influences upon me, and I owe him countless thanks for his help. I thank Prakash Panangaden for positive encouragement and discussions on topics murine and categorical. I also thank Anil Nerode for serving on my committee, and John Gilbert, who served for a period before going on leave.

Much thanks are due the PRL "boys," past and present: Stuart Allen (particularly for discussions about semantics), Dave Basin (for excursions in vegetaria), Joe Bates, Mark Bromley, Rance Cleaveland, Jim Cremer, Tim Griffin (for his implementation of inductive types), Bob Harper (for recent, electronic, discussions), Doug Howe, Todd Knoblock, Mike Manyin, Jim Sasaki, Michael Schwartzbach (who may or may not be a PRL boy) and Scott Smith. Working in this group has been an entirely positive experience. For various rewarding discussions, I am indebted to Val Breazu-Tannen, Furio Honsell, John Mitchell, Albert Meyer and Richard Shore.

I thank the National Science Foundation, IBM and Cornell University for financial support.

Without the ♡ and chi (not to be confused with χ) of Marsha Lawrence, I would not be alive today.

There is no wilder office environment than the one I now inhabit, thanks to the personages of Laurie Hendren, Ms. Aleta Ricciardi and Miss Anne Rogers. Special mention must be given to those friends up and down Upson's hallways: Vicki Feinberg, Mike Karasick, Haesun Park, Geri Pinkham and Carolyn Turbyfill. Thanks to Rob McCurley and Stuart Allen for help using LaTeX. Finally, while I don't have the space to thank them all individually, I owe all my friends a great deal, especially: Beth Byers, Sally Egan, Martha Hamblin, Paige Lawrence, Jacquie Lopez, Francesmary Modugno, Pat Ryan and Suzanne Sprunger. Finally, my thanks all those, though not mentioned, have helped me through it all.

Table of Contents

List of Figures

Chapter 1

Introduction

This thesis is about adding forms of inductive definition to the NUPRL logic, a constructive type theory, and constructing a semantic account that justifies the impredicativity inherent in this extension. In this chapter we motivate the use of type theory in the NUPRL system and the need for inductive types. We conclude with an outline of the remaining chapters.

1.1 Type theory and the NUPRL project

Type theories, since their initial presentation in Russell and Whitehead's *Principia Mathematica* [36], have been proposed as foundational theories for mathematics. More recently, Constable [10] and Bishop proposed the use of constructive theories as specification and programming languages, and as a foundation for computer science. The desire to formalize Bishop's constructive development of real analysis [4], influenced Martin-Löf in his initial work with type theories [33,32], later he became interested in their role as a programming language. In turn, the details of Martin-Löf's theories influenced Constable in the earlier PL/CV project [13,14] and the the ongoing NUPRL project [12], which is concerned with the development of a machine assisted environment for problem solving, using for its logical language such a constructive type theory.

The NUPRL logic is readily seen as a variation on the type theories of Martin-Löf, but the roots of these logics can be traced back to the founders of mathematical logic, notably Frege, Brouwer and Russell [28,18]. Its design was also influenced by the AUTOMATH project [7,8] and work of

Scott on constructive type theory [37]. Some features of the NUPRL logic are:

- A rich type structure, which makes it expressive enough to be able to formalize large amounts of constructive mathematics.
- Higher-order reasoning, through quantification over arbitrary types and a cumulative hierarchy of type universes.
- Open-endedness — built with the anticipation for possible extensions by new types and computations.

Key to the logic's relevance is the "propositions-as-types" principle, which provides a strong comprehension principle by embedding constructive logic into the type theory [17,29]. The logic allows proofs to be interpreted as programs [11], and thus certain styles of programming can be seen as arising from doing type theory.

We will not go into an extended discussion of the NUPRL system's problem solving environment, but we mention here that it supports linguistic objects such as definitions, theorems, proofs, computational expressions and libraries, and has a metalanguage (ML [23]) which allows one to write programs that can construct and transform proofs. The "proofs as programs" paradigm is manifested in an *extraction* facility, which can mechanically extract the computational content of proofs, and this can then be executed by the system's evaluation facility. Other research being conducted in machine-assisted formal theories include the Automath project [7,8], the LCF project [23], the Calculus of Constructions [16], PX [27] and the Logical Frameworks project [26].

1.2 Inductive definition

For anyone doing mathematics or programming, inductive definition needs no motivation; its natural expressiveness, elegance and computational efficiency motivate us to include forms of it into the NUPRL logic. In this thesis we address the problem of capturing forms of inductive definition in type theory, in the context of the NUPRL type theory.

We introduce two new type constructors, denoted μ and ν, which allow one to take the least and greatest solutions to suitable type equations, thus

defining general forms of inductive types and their duals, co-inductive, or lazy, types. We motivate these constructors though examples of their use in defining the natural numbers, lists, finite trees, well-founded trees, mutually defined data types, inductively defined predicates, parameterized inductive data types, streams and infinite trees. The induction principle associated with the μ types lets us define well-founded recursive functions, and dual principle for the ν types lets us inductively define their "infinite" elements.

While the type theoretic notion of function offered by the Π type constructor, and its independent "$\rightarrow$" form, requires totality, one can develop a notion of partial function through the use of the subtype constructor $\{\_|\ \_\}$. Thus, for a type ϕ that can be viewed as a predicate function on A, the elements of type $\{x\!:\!A|\ \phi(x)\} \rightarrow B$ may be considered to be partial functions with respect to $A \rightarrow B$. Given this framework, one wants convenient and expressive ways of defining ϕ and the "partial" function. Inductively defined types and functions defined from their induction principles are candidates for this situation. An example of this is the unbounded search operator discussed in 2.4, where we use a specific principle of inductive that takes into consideration the information hiding properties of the subtype constructor.

To justify the μ and ν type constructors, we give a lattice-theoretic semantics, for which the intuitionistic consistency of the proof theory is a consequence. (Intuitionistic consistency is the statement that *all* types are not inhabited, and under the propositions-as-types principle, it corresponds to propositional consistency — *False* is not provable.) We will also show how this model can justify other impredicative type constructors, such as an impredicative type abstraction operation.

1.3 Organization of the thesis

The remainder of the thesis is organized as follows. Chapter 2 begins with an introduction to type theory, and the basic theory we will be building upon, and then introduces the inductive type constructors μ and ν, giving examples of their use and proof rules for reasoning about them. We wish to prove a consistency result for this extension, so in chapter 3, we consider inductive types in the simpler setting of the second-order, polymorphic, lambda calculus [21,35,19], where we prove the strong normalization property for typed terms and also consider strong normalizability in the presence

of equational constraints between types, which allow for the typing of more terms. These arguments represent the core of the latter consistency arguments, without the additional concerns present in the latter arguments. In Chapter 4 we construct a lattice theoretic semantics for the basic type theory, based on the work of Allen [2] and Harper [25]. Chapter 5 extends this semantics to account for the inductive types through a Girard-like relativization [21]. Intuitionistic consistency of the proof theory with inductive types is a consequence of this semantic account. Chapter 6 draws conclusions from this work, and discusses further research.

Chapter 2

Inductive Types

In this chapter we begin with an outline of an intuitionistic theory of types, based on the NuPrl logic [12]. In section 2 we extend this basic theory with two new varieties of type constructors, denoted μ and ν, which allow one to take the least and greatest solutions to suitable type equations, thus defining general forms of inductive types and their duals, co-inductive, or lazy, types. We motivate these constructors though examples of their use in defining recursive data types and predicates. Finally, in sections 3 and 4, we present the proof rules needed to define the μ and ν type constructors and illustrate how well-founded recursive functions (in the μ case) and "infinite" objects (in the ν case) may be defined with the principles of induction associated with these types.

2.1 Introduction to type theory

The definition of a theory of types is done in three stages. First, the terms of the language are defined. This is straightforward: for instance, if b is a term and x a variable, then $\lambda x.b$ is a term where the x in front of the dot and all free occurrences of x in b become bound; and if B and C are terms then $(\Pi x : B)C$ is a term where the x in front of the colon and all free occurrences of x in C become bound. Second, an evaluation relation $\geq$ is defined on closed terms. Evaluation is a partial function on closed terms — a term can evaluate to at most one term. The reduction algorithm is often referred to as *lazy* or *head* reduction, and terms which evaluate to themselves are called *canonical.* We define evaluation inductively, by listing

the canonical terms and listing clauses for the other terms. For example, $\lambda x.b$ and $(\Pi x : B)C$ are taken to be a canonical terms, and we write the rule of β reduction as:

$$\frac{c \geq \lambda x.b \quad b[a/x] \geq e}{c(a) \geq e}.$$

An important computational property of a type will be that all its elements evaluate to some canonical form; from this we will be able to deduce that functions in Π types will be total, that elements in a Σ type will evaluate to pairs, and so on. In the third stage, we define types and their membership relations by way of a sequent style proof theory. We now give an outline of the notions of typehood and type membership, and discuss the rules of the proof theory, then introduce the principle of propositions-as-types, which will allow us to embed intuitionistic logic into type theory.

2.1.1 Typehood and membership

A type is a prescription for constructing equal members. In our theory, the types are defined inductively: there are atomic types, such as *void*, a type with no members, and the universe types U_1, $U_2, \ldots$; and there are type constructors, which let us build new types from old in a uniform fashion, such as Π, which takes a type B and a family of types $C(x)$ indexed by elements of B, and constructs their indexed product, denoted $(\Pi x : B)C$. Types are collected into the cumulative series of universe types U_1, $U_2, \ldots$, where they, too, have equality relations imposed upon them, based on their structure (rather than extensionally, on their membership relations).

In a theory of types one of the basic judgements is "terms a and b are equal elements of the type A," which we denote $a = b \in A$. One could also introduce the judgement "A and B are equal types," but it suffices to use the judgements $A = B \in U_k$. The judgement "a is an element of type A" is expressed by $a = a \in A$, and we often use the abbreviation $a \in A$ for this reflexive case. Terms in a type are called its *inhabitants*, and in the context of propositions-as-types, the question of type inhabitation is a fundamental one, because it corresponds to truth.

In defining a logic to reason about types and their elements, it is natural to extend judgements to express hypothetical assertions. We choose to do this in a sequent style [20], where the basic unit of inference is a hypothetical judgement called a *sequent*:

$$x_1 : A_1, \ldots, x_n : A_n \vdash b = b' \in B$$

where:

1. $0 \leq n$
2. $A_1, \ldots, A_n, b, b'$ and B are terms.
3. $x_1, \ldots, x_n$ are distinct variables
4. For $1 \leq i \leq n$, variables occurring free in A_i are among $x_1, \ldots, x_{i-1}$
5. the variables occurring free in b, b' and B are among $x_1, \ldots, x_n$.

The A_i's are called *hypotheses*, and $x_1 : A_1, \ldots, x_n : A_n$ is called a *context*. Let Γ range over contexts. We will formally define truth for sequents in chapter 4, but roughly speaking, a sequent is true if, for two vectors of terms that are equal in the context, the two instances of B are equal types, and the instances of b and b' are equal elements of that type.

A type, or type constructor, is typically presented in the proof theory by rules which prescribe how to form equal types (hence, reflexively, how to build types), how to construct (equal) members in the type, how members of the type are analyzed and how to compute with elements. Thus, rules defining a type fall into the categories of formation, introduction, elimination and computation.

We now illustrate the type theory and single out some important characteristics of it through examples. Appendix A contains a complete definition of the basic theory adopted in this thesis.

2.1.2 Example: the Π type constructor

For type $(\Pi x : B)C$, we can present simpler versions of the rules for the case when the family of types $C(x)$ is independent of its indexing by elements of B: as in the corresponding set theoretic case, the result is a function space from B to C, and for this case we abbreviate $(\Pi x : B)C$ by $B \to C$. The type formation rule, in its reflexive form, is:

$$\begin{array}{l} \Gamma \vdash B \to C \in U_j \\ \quad \Gamma \vdash B \in U_j \\ \quad \Gamma \vdash C \in U_j. \end{array}$$

We are introducing a convention for displaying sequent style rules called the *refinement* or *top-down* style, in which the subgoal sequents that can justify the goal sequent are written indented and below the goal. We can read this rule as "In the context Γ, for $A \rightarrow B$ to be a type of universe level j, it is sufficient that A and B are types of level j". The non-reflexive form of the rule is:

$$\begin{array}{l}\Gamma \vdash B \rightarrow C = B' \rightarrow C' \in U_j \\ \quad \Gamma \vdash B = B' \in U_j \\ \quad \Gamma \vdash C = C' \in U_j.\end{array}$$

As in this case, it is usually a trivial matter to infer the general form of the rule from its reflexive case, so we often state rules in their reflexive form.

Now that we know how to form types with the arrow constructor, we want to describe their elements. Informally, equal elements of $B \rightarrow C$ are normalizable terms which map equal elements of A to equal elements of B. One rule of introduction is:

$$\begin{array}{l}\Gamma \vdash \lambda x.c = \lambda x.c' \in B \rightarrow C \\ \quad \Gamma,\ x{:}B \vdash c = c' \in C \\ \quad \Gamma \vdash B \in U_j.\end{array}$$

The first subgoal asserts $c[b/x]$ and $c'[b'/x]$ will be equal elements of C for equal elements b and b' of B. The second subgoal is necessary to guarantee B is a type: recall that for the goal sequent to be true, instances of $B \rightarrow C$ must be equal types, and while the truth of first subgoal implies this for C, we need the second subgoal in order to conclude this for B. Subgoals such as the second are informally called *well-formedness* subgoals, and by convention are listed last. Type theory has a rich enough language that the well-formedness of expressions in it is, in general, undecidable, and thus our sequent calculus is designed to simultaneously prove the well-formedness and inhabitation of types. We have a second introduction rule for this type, referred to as the *extensionality* rule:

$$\begin{array}{l}\Gamma \vdash d = d' \in B \rightarrow C \\ \quad \Gamma,\ x{:}B \vdash d(x) = d'(x) \in C \\ \quad \Gamma \vdash d \in D \\ \quad \Gamma \vdash d' \in D' \\ \quad \Gamma \vdash B \in U_j.\end{array}$$

In contrast to the previous rule, here we do not require the terms being equated in $B \to C$ to display an outermost "λ," (for instance, one can imagine constants symbols in arrow types) but for technical reasons the second and third subgoals are necessary to ensure that d and d' are normalizable.

The second introduction rule has already hinted at how elements of $B \to C$ are used — by function application. The elimination rule is:

$$\begin{array}{l} \Gamma \vdash c(b) = c'(b') \in C \\ \quad \Gamma \vdash c = c' \in B \to C \\ \quad \Gamma \vdash b = b' \in B. \end{array}$$

This rule embodies the fact that equal elements of $B \to C$ map equal elements of B to equal elements of C, but it does not say *how* one actually computes with a $\lambda x.c$ term. This is prescribed in the computation rule:

$$\begin{array}{l} \Gamma \vdash \lambda x.c(b) = c[b/x] \in T \\ \quad \Gamma \vdash c[b/x] \in T. \end{array}$$

Thus, equality in a type is preserved under β reduction.

As an example of these rules, suppose $\Gamma \vdash B \in U_j$ and $\Gamma \vdash C \in U_j$. Then we have the following derivation, showing a typed version of η conversion, where for readability we omit well-formedness subgoals.

$\Gamma,\ x{:}B \to C \vdash x = \lambda y.x(y) \in B \to C$		*by extensionality*
1.	$\ldots z{:}B \vdash x(z) = (\lambda y.x(y))(z) \in C$	*by computation*
1.1	$\ldots \vdash x(z) = x(z) \in C$	*by elimination*
1.1.1	$\ldots \vdash x \in B \to C$	*by hypothesis*
1.1.2	$\ldots \vdash z \in B$	*by hypothesis*
2.	$\ldots \vdash x \in B \to C$	*by hypothesis*
3.	$\ldots \vdash \lambda y.x(y) \in B \to C$	*by introduction*
3.1	$\ldots y{:}B \vdash x(y) \in C$	*like 1.1*

(The rule of hypothesis lets us assert that the variable bound to a hypothesis does indeed inhabit that type:

$$\Gamma,\ x{:}A,\ \Gamma' \vdash x \in A.)$$

Now we briefly consider the proof rules for the general form of the Π type. Formation reflects the fact that the second component is a family of types:

$$\Gamma \vdash (\Pi x\!:\!B)C = (\Pi x\!:\!B')C' \in U_j$$
$$\Gamma \vdash B = B' \in U_j$$
$$\Gamma,\ x\!:\!B \vdash C = C' \in U_j.$$

Introduction, elimination and computation are as is the simpler arrow case, if we replace $B \rightarrow C$ by $(\Pi x : B)C$ — except that in the elimination rule, we must indicate the dependence of C on B:

$$\Gamma \vdash c(b) = c'(b') \in C[b/x]$$
$$\Gamma \vdash c = c' \in (\Pi x\!:\!B)C$$
$$\Gamma \vdash b = b' \in B.$$

While it seems appropriate in the case of a function space to represent a function by a lambda term, note that an indexed sum can also be represented by a lambda term that maps an index b to an element in $C[b/x]$.

2.1.3 Other type constructors

We briefly consider the other type constructors in the logic. In each case their proof rules can be grouped into the categories of formation, introduction, elimination and computation.

Analogous to Π, there is a Σ type constructor for forming indexed sums. Its formation rule is the same as Π's and we form canonical members by pairing an element b from B with an element c from $C[b/x]$, hence the introduction rule:

$$\Gamma \vdash \langle b,\ c\rangle = \langle b',\ c'\rangle \in (\Sigma x\!:\!B)C$$
$$\Gamma \vdash b = b' \in B$$
$$\Gamma \vdash c = c' \in C[b/x]$$
$$\Gamma,\ x\!:\!B \vdash C \in U_j.$$

The elimination form is called *spread* and it allows one to analyze pairs:

$$\Gamma \vdash \mathit{spread}(d;\ x,y.t) \in T[d/z]$$
$$\Gamma,\ x\!:\!B,\ y\!:\!C \vdash t \in T[\langle x,\ y\rangle/z]$$
$$\Gamma \vdash d \in (\Sigma x\!:\!B)C.$$

In the term $\mathit{spread}(d;\ x,y.t)$, the x and y in front of the dot and free occurrences of them in t become bound. The computation rule is:

$$\Gamma \vdash \mathit{spread}(\langle a,\ b\rangle;\ x,y.t) = t[a,b/x,y] \in T$$
$$\Gamma \vdash t[a,b/x,y] \in T.$$

As in the Π case, the Σ type has a useful characterization when C is independent of B, and that — as one can easily check against the rules — is as the cartesian product of B and C. For this case we abbreviate $(\Sigma x : B)C$ by $B \times C$.

Another important type construction is the formation of the disjoint union of two types: we denote this by $B + C$. The formation rule is simply:

$$\Gamma \vdash B + C = B' + C' \in U_j$$
$$\Gamma \vdash B = B' \in U_j$$
$$\Gamma \vdash C = C' \in U_j.$$

Elements of the union type are tagged by *inl* or *inr* to indicate from which component type they arise:

$$\Gamma \vdash \mathit{inl}(b) = \mathit{inl}(b') \in B + C$$
$$\Gamma \vdash b = b' \in B$$
$$\Gamma \vdash C \in U_j$$

$$\Gamma \vdash \mathit{inr}(c) = \mathit{inr}(c') \in B + C$$
$$\Gamma \vdash c = c' \in C$$
$$\Gamma \vdash B \in U_j.$$

The elimination rule analyzes the disjoint union element:

$$\Gamma \vdash \mathit{decide}(d;\ x.b;\ y.c) \in T[d/z]$$
$$\Gamma,\ x{:}B \vdash b \in T[\mathit{inl}(x)/z]$$
$$\Gamma,\ y{:}C \vdash c \in T[\mathit{inr}(y)/z]$$
$$\Gamma \vdash d \in B + C.$$

Decide's rules of computation distinguishs between tags:

$$\Gamma \vdash \mathit{decide}(\mathit{inl}(d);\ x.b;\ y.c) = b[d/x] \in T$$
$$\Gamma \vdash b[d/x] \in T$$

$$\Gamma \vdash \mathit{decide}(\mathit{inr}(d);\ x.b;\ y.c) = c[d/y] \in T$$
$$\Gamma \vdash c[d/y] \in T.$$

There are two further type constructors, I and $\{\_|\ \_\}$, which are best motivated after we introduce the principle of propositions-as-types.

2.1.4 The principal of propositions-as-types

Succinctly put, the propositions-as-types principle [17,29] is the identification of a proposition with the type of its justifications. An element of such a type encodes the computational content of a proof of that proposition, hence the aphorism: "TRVTH IS INHABITATION." This also justifies calling the types listed in the context "hypotheses," for to assume we are given a term in A, when A represents a proposition, is to assume A is true. We stop short of considering elements in a type as being actual *proofs* of a proposition, because the property of being a proof of a proposition should be decidable, and as already noted, membership is not.

Under this principle, we make a correspondence between propositional connectives and type constructors, arriving at a translation, $\mathcal{H}$, from propositions to types.

$$\begin{aligned}
\mathcal{H}(\mathit{False}) &\equiv \mathit{void} \\
\mathcal{H}(B \vee C) &\equiv \mathcal{H}(B) + \mathcal{H}(C) \\
\mathcal{H}(B \wedge C) &\equiv \mathcal{H}(B) \times \mathcal{H}(C) \\
\mathcal{H}(B \Rightarrow C) &\equiv \mathcal{H}(B) \rightarrow \mathcal{H}(C) \\
\mathcal{H}(\exists x \in B.C) &\equiv (\Sigma x{:}B)\mathcal{H}(C) \\
\mathcal{H}(\forall x \in B.C) &\equiv (\Pi x{:}B)\mathcal{H}(C) \\
\mathcal{H}(a = b \in B) &\equiv I(a, b, B)
\end{aligned}$$

As in the usual intuitionistic reading of negation, we regard $\neg A$ as an abbreviation of $A \Rightarrow \mathit{False}$.

In the last clause of the above definition, we used a new type to reflect the judgement $a = b \in B$ into a type. Its formation rule is:

$$\begin{array}{l}
\Gamma \vdash I(a, b, B) \in U_j \\
\quad \Gamma \vdash B \in U_j \\
\quad \Gamma \vdash a \in B \\
\quad \Gamma \vdash b \in B.
\end{array}$$

Novices often confuse this type or the judgement $a \in B$ for the $\in$ predicate of set theory. Firstly, $a \in B$ is a judgement about types and their elements and not a formula *in* type theory — it corresponds to the judgement "A *is true.*" Secondly, it is more illuminating to think of type theory as a many-sorted logic (sorted by types) and $I(a, b, B)$ as the formula $a =_B b$ — the equality predicate for sort B; for this formula to be well-formed it

is natural to require a and b to be terms of type B. Thus, one can not encode the set-theoretic proposition $a \in b$ as $I(a, a, b)$ because, if it is a well-formed type, it necessarily will be inhabited.

The introduction rule for an I type states that it is inhabited by the atom *true* when it represents a true judgment:

$$\frac{\Gamma \vdash a = b \in B}{\Gamma \vdash \mathit{true} \in I(a, b, B)}.$$

The elimination rule shows this is the only situation when the I type is inhabited:

$$\frac{\Gamma \vdash t \in I(a, b, B)}{\Gamma \vdash a = b \in B}.$$

Our final type constructor is unusual in that it does not prescribe how to form new objects. Rather, it allows us to hide or trivialize the computational content of a type. In the reflexive case, the formation rule is unremarkable:

$$\frac{\Gamma \vdash B \in U_j \qquad \Gamma,\ x{:}B \vdash C \in U_j}{\Gamma \vdash \{x{:}B \mid C\} \in U_j},$$

but the general rule displays a unique property: subtypes are equated when their second components are merely co-inhabited:

$$\frac{\Gamma \vdash B = B' \in U_j \qquad \Gamma,\ x{:}B \vdash C \in U_j \qquad \Gamma,\ x{:}B \vdash C' \in U_j \qquad \Gamma,\ x{:}B,\ y{:}C \vdash t' \in C' \qquad \Gamma,\ x{:}B,\ y{:}C' \vdash t \in C}{\Gamma \vdash \{x{:}B \mid C\} = \{x{:}B' \mid C'\} \in U_j}.$$

(The choice of this courser equality is unrelated to the type's other properties; we could have adopted the same notion of type equality given to Π and Σ types.) Equality in this type is the restriction of B's equality to elements b for which $C[b/x]$ is inhabited:

$$\frac{\Gamma \vdash b = b' \in B \qquad \Gamma \vdash c \in C[b/x] \qquad \Gamma,\ x{:}B \vdash C \in U_j}{\Gamma \vdash b = b' \in \{x{:}B \mid C\}}.$$

Note that c is not a component of the element of $\{x : B|\ C\}$ — this is the point at which information is suppressed. The elimination rule lets us assume $C(x)$ is inhabited, but note that its inhabitant y can not appear in the conclusion:

$$\begin{array}{l}\Gamma \vdash t = t' \in T \\ \quad \Gamma,\ y{:}C[b/x],\ w{:}I(b,b,B) \vdash t = t' \in T \\ \quad \Gamma \vdash b \in \{x{:}B|\ C\} \\ \quad \Gamma,\ x{:}B \vdash C \in U_j.\end{array}$$

In a derivation, subtype elimination and introduction rules often work in concert, in their suppression of information. The following example shows that inhabitants of $\{x : A|\ B\}$ also inhabit $\{x : A|\ B + C\}$: note how z is used to inhabit the union type, but does not appear in the goal sequent. (As usual, we suppress well-formedness subgoals.)

$y{:}\{x{:}A|\ B\} \vdash y \in \{x{:}A|\ B + C\}$		*by elim*
1.	$\ldots z{:}B[y/x],\ w{:}I(y,y,A) \vdash y \in \{x{:}A|\ B + C\}$	*by intro*
1.1	$\ldots \vdash y \in A$	*by I elim*
1.1.1	$\ldots \vdash w \in I(y,y,A)$	*by hyp*
1.2	$\ldots \vdash inl(z) \in B[y/x] + C[y/x]$	*by intro*
1.2.1	$\ldots \vdash z \in B[y/x]$	*by hyp*
2.	$\ldots \vdash y \in \{x{:}A|\ B\}$	*by hyp*

If we are to take the propositions-as-types principle to heart, then the form of the rules appears to put the cart before the horse, because a sequent doesn't assert a type is inhabited and so a proposition is true, so much as it asserts (in the reflexive case) that a particular element inhabits a type and so there is a proof of a proposition with that as its computational content. But this presentational route is only taken to simplify the formalization of the theory: an examination of the proof rules will reveal that the inhabiting term or terms can be mechanically generated in a bottom-up fashion. In the NuPrl system, this procedure is referred to as *extraction*: one proves a proposition in top-down fashion and an inhabiting term is synthesized bottom-up (although one can imagine more elaborate paradigms, so long as the completed proofs are valid). These extracted terms are often large (searching for a "looping combinator," Howe [30] extracted a term of over forty pages) so a usable computer implementation of the logic must certainly allow for their suppression in display.

When the rules are displayed in this style, the propositions-as-types principle leaps off the screen. For instance, the introduction and elimination rules for arrow are now presented as:

$$\frac{\Gamma \vdash B \to C}{\Gamma,\ B \vdash C} \qquad \textit{and} \qquad \frac{\Gamma \vdash C}{\Gamma \vdash B \to C \quad \Gamma \vdash B},$$

which are the rules for $\Rightarrow$ introduction and elimination. Similarly, the introduction and elimination rules for the general Π type, cartesian product, the general Σ type and the disjoint union type can be presented as in figure 2.1. In further agreement with the translation $\mathcal{H}$, we see these are the introduction and elimination rules for $\forall$, $\wedge$, $\exists$ and $\vee$, respectively. How are the other rules to be read in this light? The formation rules assert a distinct judgement, "A is a well-formed proposition," and the computation rules assert equivalences among proof expressions.

This completes our overview of the basic theory of types. One more important property of type theories we note here is extensibility: their design allows for possible extension by new atomic types, type constructors or elements. In the next section we take advantage of this by extending the basic theory with new type constructors.

2.2 Examples of inductive types

In this section we introduce two new type constructors, written μ and ν, to allow us to solve certain type equations. Both of the μ and ν constructors are given in a simple and a parameterized version. We motivate these types through a series of examples. Our syntax is $(\mu{:}U_j)B$ and $(\nu x{:}U_j)B$, for simple inductive types, where the μ type is meant to denote the least solution to $x \equiv B(x)$ in U_j, and the ν type, the greatest solution. For the parameterized inductive types, our syntax is $(\mu x : C \to U_j)B@c$ and $(\nu x{:}C \to U_j)B@c$, where the μ type is defined as being a type $x(c)$ where x is the least solution to $x \equiv B(x)$ in $C \to U_j$, and the ν type is the greatest solution of this.

With simple μ types, one can represent basic inductively defined types, such as the natural numbers, lists, binary trees and well-founded trees:

$$\frac{\Gamma \vdash (\Pi x{:}B)C \qquad \Gamma \vdash C[b/x]}{\Gamma,\ x{:}B \vdash C \qquad \Gamma \vdash (\Pi x{:}B)C \qquad \Gamma \vdash b \in B}$$

$$\frac{\Gamma \vdash B \times C}{\Gamma \vdash B \qquad \Gamma \vdash C} \qquad \frac{\Gamma \vdash T}{\Gamma,\ B,\ C \vdash T \qquad \Gamma \vdash B \times C}$$

$$\frac{\Gamma \vdash (\Sigma x{:}B)C}{\Gamma \vdash C[b/x] \qquad \Gamma \vdash b \in B} \qquad \frac{\Gamma \vdash T}{\Gamma,\ x{:}B,\ C \vdash T \qquad \Gamma \vdash (\Sigma x{:}B)C}$$

$$\frac{\Gamma \vdash B + C}{\Gamma \vdash B} \qquad \frac{\Gamma \vdash T}{\Gamma,\ B \vdash T \qquad \Gamma,\ C \vdash T \qquad \Gamma \vdash B + C}$$

$$\frac{\Gamma \vdash B + C}{\Gamma \vdash C}$$

Figure 2.1: *Proof rules with suppressed inhabiting terms*

$$
\begin{array}{rcl}
N & \equiv & (\mu N{:}U_1)1 + N \\
List & \equiv & (\mu L{:}U_1)1 + (A \times L) \\
Tree & \equiv & (\mu T{:}U_1)A + (B \times T \times T) \\
Wtree & \equiv & (\mu W{:}U_1)(\Sigma x{:}A)B \multimap W.
\end{array}
$$

Each type provides an inductive principle for computing with its elements. With the simple ν types, one can represent "infinite" objects, such as streams, types of finite and infinite trees, and types of just infinite trees:

$$
\begin{array}{rcl}
Stream & \equiv & (\nu S{:}U_1)A \times S \\
FItree & \equiv & (\nu T{:}U_1)A + (B \times T \times T) \\
Itree & \equiv & (\nu T{:}U_1)B \times T \times T.
\end{array}
$$

Dual to the μ case, each ν type provides an inductive principle for defining elements of it.

The parameterized versions of the μ and ν types allow us to recursively define predicates, under the propositions-as-types principle, as well as to create more complex data types. For example, let f be an arbitrary operation on N; $\phi(n)$ is inhabited when f has a root greater or equal to n:

$$\phi(n) \equiv (\mu D{:}N \to U_1)\lambda x.I(f(x), 0, N) + D(x+1)@n.$$

We will return to this type later in the chapter in defining an unbounded search operation. One common example of data types defined with a parameterized μ type is that of mutually defined data types. Suppose we wish to inductively define T_1 and T_2, where:

$$
\begin{array}{rcl}
T_1 & \equiv & A(T_1, T_2) \\
T_2 & \equiv & B(T_1, T_2).
\end{array}
$$

It is easy to define a type Two with canonical elements $\bot$ and $\top$ and the elimination form $case(a; b; c)$, where:

$$\frac{a \geq \bot \quad b \geq e}{case(a; b; c) \geq e} \quad \text{and} \quad \frac{a \geq \top \quad c \geq e}{case(a; b; c) \geq e}.$$

With this simple type, we can define T_1 or T_2 by:

$$(\mu T : Two \to U_1)\lambda x.case(x; A(T(\bot), T(\top)); B(T(\bot), T(\top)))@b,$$

with b being $\bot$ or $\top$, respectively. With the parameterized ν type, we can do similar things, only obtaining the greatest solutions. For example, suppose we wish to define a type that asserts proposition p is common knowledge — p is true and for the collection of individuals I, it is common knowledge that everyone knows p. Let P be the type of propositions, $\mathit{true} \in P \rightarrow U_1$ and $K \in (I \times P) \rightarrow P$. Then the common knowledge of p can be expressed by the type:

$$(\nu C : P \rightarrow U_1)\lambda q.\mathit{true}(q) \times (\Pi i : I)C(K(i,q))@p.$$

We continue now by presenting the proof rules for these inductive types.

2.3 Proof rules for simple types

The formation rule for simple μ type prescribes how such a type may be constructed:

$$\begin{array}{l}\Gamma \vdash (\mu x : U_j)B \in U_j \\ \quad \Gamma,\ x : U_j \vdash B \in U_j \\ \quad \Gamma,\ x : U_j,\ y : U_j, x \subseteq y \vdash t \in B \subseteq B[y/x].\end{array}$$

$(\mu x : U_j)B$ is defined as being the inductive solution to $x \equiv B(x)$ in U_j. The first subgoal asserts that the body of the μ type must be an operation on universe level j. Up until now, all type constructors were predicative in nature: they defined new types in a universe level without needing to quantify over the types of that level, but this quantification is implicit in inductive types, and so we must come to terms with it in some way. Most of the effort invested in chapter 5 will be in justifying this impredicativity. Moreover, the second subgoal is asserting that the body is a *monotonic* operation on U_j. In chapters 4 and 5, we will model type universes by complete lattices, and this subgoal ensures the body of the inductive type represents a monotonic operation on such a complete lattice. Thus, we may take least and greatest fixed points of this operation to be the meanings of μ and ν types, respectively. The formation rule for the simple ν types is identical to the last rule:

$$\begin{array}{l}\Gamma \vdash (\nu x : U_j)B \in U_j \\ \quad \Gamma,\ x : U_j \vdash B \in U_j \\ \quad \Gamma,\ x : U_j,\ y : U_j, x \subseteq y \vdash t \in B \subseteq B[y/x].\end{array}$$

The introduction rule for the simple μ type prescribes how we may construct equal elements in that type:

$$\begin{array}{l}\Gamma \vdash b = b' \in (\mu x{:}U_j)B \\ \quad \Gamma \vdash b = b' \in B[(\mu x{:}U_j)B/x] \\ \quad \Gamma \vdash (\mu x{:}U_j)B \in U_j.\end{array}$$

This is the essence of an inductive type: to construct equal elements in $(\mu x{:}U_j)B$, one constructs equal elements in the "unrolling" of the type. As an example, the following are elements of type *List* (where $\bot \in 1$).

$$\begin{array}{rcl} nil & \equiv & inl(\bot) \\ [a_1] & \equiv & inr(\langle a_1,\ nil\rangle) \\ [a_2, a_1] & \equiv & inr(\langle a_2,\ [a_1]\rangle)\end{array}$$

The corresponding rule for the simple ν type is its elimination rule, which prescribes how elements of the type are used:

$$\begin{array}{l}\Gamma \vdash out(b) = out(b') \in B[(\nu x{:}U_j)B/x] \\ \quad \Gamma \vdash b = b' \in (\nu x{:}U_j)B.\end{array}$$

Thus, *out* is a projection function that maps an element of the ν type into its "unrolling."

The elimination rule for simple μ type prescribes the use of its elements: how we may compute with them. It embodies the induction principle for these types.

$$\begin{array}{l}\Gamma \vdash \mu\_ind(b;\ z, y.d) \in D[b/y] \\ \quad \Gamma,\ x{:}U_j,\ x \subseteq (\mu x{:}U_j)B,\ z{:}(\Pi y{:}x)D,\ y{:}B \vdash d \in D \\ \quad \Gamma \vdash b \in (\mu x{:}U_j)B\end{array}$$

Under the propositions-as-types principle, we can read this as a proof by induction on the inductive definition of a type: to conclude $D(b)$ is true, show that by assuming $D(y)$ for y in a fixed subset x of $(\mu x : U_j)B$, we can infer $D(y)$ for y in $B(x)$. As an example of the μ_ind form defining a recursive operation on a data type, take T to be the type of binary trees with natural numbers at the leaves: $T \equiv (\mu T{:}U_1)N + (T \times T)$. Then the following sums the values in the leaves of $t \in T$:

$$\mu\_ind(t; sum, t.decide(t; lf.lf; u.spread(u; l, r.sum(l) + sum(r)))).$$

The corresponding rule for the simple ν type is the introduction rule, which asserts an inductive principle for defining elements of the type:

$$\begin{array}{l}\Gamma \vdash \nu\_ind(d;\ z, y.b) \in (\nu x{:}U_j)B \\ \quad \Gamma,\ x{:}U_j,\ (\nu x{:}U_j)B \subseteq x,\ z{:}D \to x,\ y{:}D \vdash b \in B \\ \quad \Gamma \vdash d \in D \\ \quad \Gamma \vdash (\nu x{:}U_j)B \in U_j.\end{array}$$

Let S be the type of streams of natural numbers: $S \equiv (\nu x{:}U_1)N \times x$. Then the following is a stream of increasing numbers, starting at n:

$$\nu\_ind(n; z, y.\langle y,\ z(y+1)\rangle),$$

and the following takes streams s and t, and interleaves their elements to form a third stream, by outjecting s into the number n and the stream u, then forming the stream where n is followed by the interleaving of t and u.

$$\nu\_ind(\langle s,\ t\rangle; z, y.spread(y; s, t.spread(out(s); n, u.\langle n,\ z(\langle t,\ u\rangle)\rangle))))$$

Lastly, we have the following rules for computing with *ind* forms, which show the μ_ind and ν_ind forms to be fixed point combinators.

$$\begin{array}{l}\Gamma \vdash \mu\_ind(b;\ z, y.d) = d[\lambda y.\mu\_ind(y;\ z, y.d), b/z, y] \in T \\ \quad \Gamma \vdash d[\lambda y.\mu\_ind(y;\ z, y.d), b/z, y] \in T\end{array}$$

$$\begin{array}{l}\Gamma \vdash out(\nu\_ind(b;\ z, y.d)) = d[\lambda y.\nu\_ind(y;\ z, y.d), b/z, y] \in T \\ \quad \Gamma \vdash d[\lambda y.\nu\_ind(y;\ z, y.d), b/z, y] \in T\end{array}$$

2.4 Proof rules for parameterized types

In this section, we will use the following abbreviations.

$$\begin{array}{rclcrcl} \mu@c & \equiv & (\mu x{:}C \to U_j)B@c & & \mu & \equiv & \lambda w.\mu@w \\ \nu@c & \equiv & (\nu x{:}C \to U_j)B@c & & \nu & \equiv & \lambda w.\nu@w \\ A \subseteq_C B & \equiv & (\Pi w{:}C)A(w) \subseteq B(w) & & & & \end{array}$$

The proof rules for the parameterized versions of the μ and ν types generalize the simple rules in a straightforward manner — here we are defining a family of types, indexed by a type C, instead of a single type. The formation rule for the parameterized μ type is:

$$\Gamma \vdash \mu@c \in U_j$$
$$\Gamma,\ x{:}C \to U_j \vdash B \in C \to U_j$$
$$\Gamma,\ x{:}C \to U_j,\ y{:}C \to U_j,\ x \subseteq_C y \vdash t \in B \subseteq_C B[y/x]$$
$$\Gamma \vdash C \in U_j$$
$$\Gamma \vdash c \in C.$$

$\mu@c$ is defined as being the type $x(c)$ where x is the inductive solution to $x \equiv B(x)$ in type $C \to U_j$. As before, in order for this equation to have a solution it must be well-formed and monotonic, and that is ensured by the given subgoals. The formation rule for the parameterized ν type is identical:

$$\Gamma \vdash \nu@c \in U_j$$
$$\Gamma,\ x{:}C \to U_j \vdash B \in C \to U_j$$
$$\Gamma,\ x{:}C \to U_j,\ y{:}C \to U_j,\ x \subseteq_C y \vdash t \in B \subseteq_C B[y/x]$$
$$\Gamma \vdash C \in U_j$$
$$\Gamma \vdash c \in C.$$

The introduction rule for the parameterized μ type prescribes how to construct equal elements in that type:

$$\Gamma \vdash b = b' \in \mu@c$$
$$\Gamma \vdash b = b' \in B[\mu/x](c)$$
$$\Gamma \vdash \mu@c \in U_j.$$

As in the simple case, to construct equal elements in the inductive type, one constructs equal elements in the "unrolling" of that type. The only complication here is that $B[\mu/x]$ is a family of types, so we evaluate it for parameter value c. The corresponding rule for the parameterized ν type is its elimination rule:

$$\Gamma \vdash \mathit{out}(b) = \mathit{out}(b') \in B[\nu/x](c)$$
$$\Gamma \vdash b = b' \in \nu@c.$$

As before, *out* maps an element of the ν type into its "unrolling."

We give two versions of the elimination rule for parameterized μ types. The first is a generalization of the simple case's elimination rule:

$$\Gamma \vdash \mu\_\mathit{ind}(c, b;\ z, w, y.d) \in T[c, b/w, y]$$
$$\Gamma,\ x{:}C \to U_j,\ x \subseteq_C \mu,\ z{:}(\Pi w{:}C)(\Pi y{:}x(w))T,\ w{:}C,\ y{:}B(w)$$
$$\vdash d \in T$$
$$\Gamma \vdash b \in \mu@c.$$

The corresponding rule for the parameterized ν type is the introduction rule, which asserts an inductive principle for defining elements in that type:

$$\begin{array}{l}\Gamma \vdash \nu\_ind(c;\ z, y.d) \in \nu@c \\ \quad \Gamma,\ x{:}C \rightarrow U_j,\ \nu \subseteq_C x,\ z{:}(\Pi y{:}C)x(y),\ y{:}C \vdash d \in B \\ \quad \Gamma \vdash \nu@c \in U_j.\end{array}$$

We have another version of an elimination rule, which differs from the previous rule in that elements of the μ type are not involved in the computation: the rule tells us not so much how to compute *with elements* of the inductive type, but how to compute, knowing that the inductive type *is inhabited.* Note that this rule makes crucial use of the information hiding capabilities of the $\{\_ \mid \_\}$ type and that in the first subgoal, the element of type $B(w)$ can not appear in term d.

$$\begin{array}{l}\Gamma \vdash \mu\_ind(c;\ z, w.d) \in T[c/w] \\ \quad \Gamma,\ x{:}C \rightarrow U_j,\ x \subseteq_C \mu,\ z{:}(\Pi w{:}\{w{:}C \mid x(w)\})T,\ w{:}C,\ B(w) \\ \qquad \vdash d \in T \\ \quad \Gamma \vdash b \in \mu@c\end{array}$$

With such a rule we can prove an unbounded search property, and extract a "μ" operator from the proof. For example, let $\phi(n)$ be defined as earlier in the chapter:

$$\phi(n) \equiv (\mu D{:}N \rightarrow U_1)\lambda x.I(f(x), 0, N) + D(x+1)@n.$$

Recall that $\phi(n)$ is inhabited when f has a root greater than or equal to n. We will show that knowing, for some n, $\phi(n)$ to be inhabited, we can compute a root of f, without actually needing n or the inhabitant of $\phi(n)$ in the computation. The problem statement is: (in program extraction style, we suppress the display of the extraction term)

$$f{:}\{f{:}N \rightarrow N \mid (\Sigma n{:}N)\phi(n)\} \vdash (\Sigma n{:}N)I(f(n), 0, N).$$

Let $P \equiv (\Sigma n : N)I(f(n), 0, N)$. We proceed by eliminating on the hypothesis and applying the second induction rule. The induction subgoal reads:

$$\ldots z{:}(\Pi n{:}\{n{:}N \mid x(n)\})P,\ n{:}N,\ I(f(n), 0, N) + x(n+1) \vdash P.$$

We can prove this by cases on whether of not n is a root of f: if $f(n)$ is 0, the solution is immediate; otherwise we claim $z(n+1) \in P$, which follows from hypotheses $I(f(n), 0, N) + x(n+1)$ and $\neg I(f(n), 0, N)$. For the last subgoal of the induction rule we prove $\phi(0)$ is inhabited using assumption $(\Sigma n : N)\phi(n)$ and the fact that $(\Pi n : N)\phi(n) \rightarrow \phi(0)$ is inhabited, which follows by induction on n. What is the term inhabiting type P? It is a recursive procedure that, starting at $n = 0$, tests to see if n is a root of f and returns n, if that is the case; otherwise it recurses with $n + 1$ — in short, an unbounded search.

Finally, we have the following rules for computing with *ind* forms, which show the μ_ind and ν_ind forms to be fixed point combinators.

$$\begin{array}{l}\Gamma \vdash \mu\_ind(b; c;\ z, w, y.d) = \\ \quad d[\lambda w.\lambda y.\mu\_ind(w; y;\ z, w, y.d), b, c/z, w, y] \in T \\ \quad \Gamma \vdash d[\lambda w.\lambda y.\mu\_ind(w; y;\ z, w, y.d), b, c/z, w, y] \in T\end{array}$$

$$\begin{array}{l}\Gamma \vdash \mu\_ind(b;\ z, w.d) = d[\lambda w.\mu\_ind(w;\ z, y.d), b, c/z, w, y] \in T \\ \quad \Gamma \vdash d[\lambda w.\lambda y.\mu\_ind(w; y;\ z, w, y.d), b, c/z, w, y] \in T\end{array}$$

$$\begin{array}{l}\Gamma \vdash out(\nu\_ind(b;\ z, y.d)) = d[\lambda y.\nu\_ind(y;\ z, y.d), b/z, y] \in T \\ \quad \Gamma \vdash d[\lambda y.\nu\_ind(y;\ z, y.d), b/z, y] \in T\end{array}$$

Summary

We have introduced type constructors into a constructive type theory that let use solve monotonic type operations for their least and greatest solutions, giving us inductive and co-inductive (lazy) types. This gives us types which naturally represent recursive data types and recursively defined predicates, without introducing diverging or partially defined elements. We will eventually arrive at a semantics for this theory in chapter 5, but first we examine the issue in the simpler setting of the second-order lambda calculus, then build a semantics for the basic theory in chapter 4, which we finally extend to inductive types.

Chapter 3

Inductive Types and Type Constraints in Second-Order Lambda Calculus

Before attempting to verify the consistency of the extension of the basic type theory by inductive types, it is advisable to consider the same question in the simpler setting of the second-order lambda calculus. This will allow us to separate the core of the consistency argument from the additional concerns of the type theory. Thus, in this chapter we consider the problem of extending the second-order lambda calculus with recursive types in ways so as to maintain its strong normalizability property, which is the property corresponding to intuitionistic consistency in the type theory case. We do this extension in two ways. In section 1, type constructors μ and ν are added, which give the least and greatest solutions to positively defined type expressions. While in section 2, we consider typing terms in the presence of equational type constraints. In both cases, the method of proof employed is based upon Girard's *candidat de réductibilité* method.

The chapter is organized as follows. In section 1, we give an extension to the second-order lambda calculus [21,35,19,6], which permits the definition of least and greatest solutions to positively defined type expressions using type constructors μ and ν, respectively. With μ, one can define inductive types such as the natural numbers, constructive ordinals, lists and trees, and there are induction combinators available for each type; with ν, "lazy" types such as streams and potentially infinite trees can be defined. The focus here

is not on model theory, as in [15,3,5], but on normalization properties: the central result is a proof of strong normalizability, using Girard's *candidat de réductibilité* method [22,21], which we extend to exploit the fact that the collection of ground sets forms a complete lattice and that inductive types can be viewed as least and greatest fixed points of monotonic (but not necessarily continuous) operations on it.

Using the structures built in section 1, we proceed, in section 2, to consider typing terms in the presence of equational type constraints [5], which allow the typing of more terms. The problem can be stated as follows.

> *Given a collection of equational constraints $\tau_i = T_i$ for i in a fixed index set I, type constants τ_i and closed type expressions T_i, which we close under reflexivity, symmetry, transitivity and substitution, and the additional typing rule:*
>
> $$\frac{a : A \quad A = B}{a : B},$$
>
> *when are the resulting typed terms strongly normalizable?*

We give a decidable (in the case of a finite I) necessary and sufficient condition for this. Thus, we see, for example, that:

$$\begin{aligned} \tau_0 &= \tau_1 \to \tau_0 \\ \tau_1 &= \tau_0 \to \tau_1 \end{aligned}$$

will yield only strongly normalizable typed terms, while:

$$\begin{aligned} \tau_0 &= \tau_1 \to \tau_0 \\ \tau_1 &= \tau_2 \to \tau_1 \\ \tau_2 &= \tau_0 \to \tau_2 \end{aligned}$$

will allow a diverging term to be typed.

3.1 Inductive types

In this section we will prove the strong normalizability of terms in a second order lambda calculus extended with inductive types. First, the type expressions, terms and reduction are defined. The rest of the section consists of the proof of strong normalization, which we now outline.

1. Let $\top$ be the set of strongly normalizable untyped terms. A collection of sets of untyped terms, Ξ, is defined, so that $\xi \in \Xi \Rightarrow \xi \subseteq \top$.

2. In definition 3.1, operations on Ξ corresponding to the type constructors $\rightarrow, \Delta, \mu$ and ν are defined.

3. These operations are employed to extend an environment ρ, a function from type variables to Ξ, to $[\![\_]\!]\rho$, a function from type expressions to Ξ, by structural induction on type expressions.

4. We define the untyped terms $\hat{a}$ that are instances of a typed term a, with respect to a given environment, and note that the untyped term corresponding to a is an instance of a. Truth for judgements is defined.

5. In lemma 3.5, by an induction on the derivation of judgements, all judgements are shown to be true.

6. In lemma 3.6, we note that a typed term is strongly normalizable, if the corresponding untyped term is strong normalizable. Now our conclusion, theorem 3.1, follows easily: by 5, if $a : A$, then $\models a : A$; and by 4, this implies $\hat{a} \in [\![A]\!]\rho$, where we take $\hat{a}$ to be the untyped term corresponding to a. By 1 and 3 this implies that $\hat{a} \in \top$. Thus, all typed terms are strongly normalizable.

3.1.1 Type expressions, terms and reduction

Assume a denumerably infinite list of type variables $V_1, V_2, V_3, \ldots$, and let X, Y and Z range over them. Inductively define the type expressions as follows.

- X is a type expression.
- If A and B are type expressions, then so are $A \rightarrow B$ and $\Delta X.A$
- If X occurs positively in type expression A, then $\mu X.A$ and $\nu X.A$ are type expressions.

"X occurs positively in A," $Pos(A, X)$, iff each free occurrence of X in A is on the left-hand side of an even number of $\rightarrow$'s; similarly $Neg(A, X)$ iff

each free occurrence is on the left-hand side of an odd number of $\rightarrow$'s. Let A, B and C range over type expressions. Let $FV(A)$ denote the set of type variables occurring free in A. As one might expect, in the standard encodings [34]:

$$\begin{array}{lcl} A \times B & \equiv & \Delta Z.(A \rightarrow B \rightarrow Z) \rightarrow Z \\ A + B & \equiv & \Delta Z.(A \rightarrow Z) \rightarrow (B \rightarrow Z) \rightarrow Z \\ \Sigma Y.A & \equiv & \Delta Z.(\Delta Y.A \rightarrow Z) \rightarrow Z, \end{array}$$

X occurs positively in them iff X occurs positively in A and B. The positivity requirement will ensure the monotonicity of certain complete lattice operations, and hence the existence of least and greatest fixed points for them, which will be the meanings given to the inductive types.

Define the untyped terms to be the set of untyped lambda terms with possible occurrences of the constants R, in, Ω and out. Let a and b range over untyped terms. Untyped terms of the following syntactic forms are called *critical* and we let t range over them.

$$\lambda x.a, \quad R\,a, \quad R, \quad in\,a, \quad in, \quad \Omega\,a\,b, \quad \Omega\,a, \quad \Omega, \quad out$$

Typed terms are defined in figure 3.1. Let a and b range over typed terms, also, letting the context determine which is meant. As usual, we identify type expressions or terms which are alphabetic variants in their bound variables.

The reductions for typed terms are as follows.

1. $(\lambda x^A.a)\,b \longmapsto a[b/x^A]$
2. $\lambda x^A.a\,x^A \longmapsto a$
3. $(\Lambda X.a)\,B \longmapsto a[B/X]$
4. $\Lambda X.a\,X \longmapsto a$
5. $R^\mu B\,a\,(in^\mu b) \longmapsto a\,\mu(\lambda x^\mu.x^\mu)(R^\mu B\ a)\,b$
6. $out^\nu(\Omega^\nu B\,a\,b) \longmapsto a\,\nu(\lambda x^\nu.x^\nu)(\Omega^\nu B\ a)\,b$

Reduction 2 requires x to not occur free in a, and reduction 4 requires X to not occur free in a. Untyped terms have the analogous reductions (3 and 4 being inapplicable). Define $a > b$ to mean a reduces to b in one step, $a >^* b$ to mean a reduces to b, and let $\top$ be the set of strongly normalizable untyped terms. Reduction is Church-Rosser [39].

For each type $\mu \equiv \mu X.A$, there are constants:

$$in^{\mu} : A[\mu/X] \to \mu$$
$$R^{\mu} : \Delta Y.(\Delta X.(X \to \mu) \to (X \to Y) \to A \to Y) \to \mu \to Y$$

and for each type $\nu \equiv \nu X.A$, there are constants:

$$out^{\nu} : \nu \to A[\nu/X]$$
$$\Omega^{\nu} : \Delta Y.(\Delta X.(\nu \to X) \to (Y \to X) \to Y \to A) \to Y \to \nu$$

where Y does not occur free in A and is distinct from X. Complete the definition with the following.

$$x^A : A$$

$$\frac{b : B}{\lambda x^A.b : A \to B} \qquad \frac{b : B}{\Lambda X.b : \Delta X.B}\ *$$

$$\frac{c : A \to B \quad a : A}{c\,a : B} \qquad \frac{b : \Delta X.B}{b\,A : B[A/X]}$$

$*$ *X is not free in A for any x^A free in b.*

Figure 3.1: *Definition of typed terms*

3.1.2 Strong normalization

Define the following predicates for $\xi \subseteq \top$, and the collection of the so-called ground sets, Ξ:

$$\begin{aligned} G_1(\xi) &\equiv \forall a, b.\ (a \in \xi \wedge a >^* b) \Rightarrow b \in \xi \\ G_2(\xi) &\equiv \forall a \in \top.\ (\forall t.\ a >^* t \Rightarrow t \in \xi) \Rightarrow a \in \xi \\ \Xi &\equiv \{\xi \subseteq \top \mid G_1(\xi) \wedge G_2(\xi)\}. \end{aligned}$$

Some basic notions about partial orderings will help in analyzing Ξ. The pair $\langle S, \sqsubseteq \rangle$ is a *partial ordering* iff $\sqsubseteq$ is a binary relation over set S that is reflexive, anti-symmetric and transitive. For a given partial ordering $\langle S, \sqsubseteq \rangle$:

- $x \in S$ is an *upper bound* of $T \subseteq S$ iff $y \sqsubseteq x$ for all $y \in T$.

- $x \in S$ is a *lower bound* of $T \subseteq S$ iff $x \sqsubseteq y$ for all $y \in T$.
- Upper bound x is the *least upper bound* of $T \subseteq S$ iff $x \sqsubseteq y$ for all upper bounds y of T.
- Lower bound x is the *greatest lower bound* of $T \subseteq S$ iff $y \sqsubseteq x$ for all lower bounds y of T.
- S is a *complete lattice* iff the least upper and greatest lower bounds of every subset $T \subseteq S$ exist.
- A function $f \in S_1 \rightarrow S_2$, where $\langle S_1, \sqsubseteq_1 \rangle$ and $\langle S_2, \sqsubseteq_2 \rangle$ are partial orderings, is *monotonic* iff $x \sqsubseteq_1 y \Rightarrow f(x) \sqsubseteq_2 f(y)$ and is *anti-monotonic* iff $x \sqsubseteq_1 y \Rightarrow f(y) \sqsubseteq_2 f(x)$.
- Given the set A, the function space $A \rightarrow S$ may be partially ordered, the so-called *point-wise* ordering, by the relation $\sqsubseteq'$, where $f \sqsubseteq' g \equiv \forall a \in A.\ f(a) \sqsubseteq g(a)$.
- $T \subseteq S$ is a *chain* iff $x, y \in T \Rightarrow x \sqsubseteq y \vee y \sqsubseteq x$.

Lemma 3.1 *Ξ is a complete lattice under $\subseteq$.*

Proof. It suffices to show every subset $T \subseteq \Xi$ has a greatest lower bound. But this is trivial, since properties G_1 and G_2 are closed under intersection: the greatest lower bound is $\cap T$.

We note here the least element of Ξ, denoted $\perp$, is $\{c \in \top \mid \neg\exists t.\ c >^* t\}$. We can also note that the least upper bound of a chain is its union. The only non-trivial fact to verify is that G_2 holds for ξ, the union of chain $T \subseteq \Xi$. Suppose $a \in \top$; it can reduce to only a finite number of distinct terms, by König's Lemma. So, if all the critical terms a reduces to are in ξ, they must all be in some $\xi' \in T$, hence $a \in \xi'$, by $G_2(\xi')$, and so $a \in \xi$. Conclude $G_2(\xi)$.

□

Let $\Xi \stackrel{\text{mon}}{\rightarrow} \Xi$ be the monotonic operations in $\Xi \rightarrow \Xi$. Give these function spaces the usual point-wise orderings.

Definition 3.1 *The operations given below are well-defined.*

$$\begin{aligned}
\rightarrow &\in \Xi\times\Xi\rightarrow\Xi, \quad \xi_1\rightarrow\xi_2 \equiv \{c \in \top \mid \forall a.\, a \in \xi_1 \Rightarrow c\,a \in \xi_2\} \\
\boldsymbol{\Delta} &\in (\Xi\rightarrow\Xi)\rightarrow\Xi, \quad \boldsymbol{\Delta}(f) \equiv \bigcap\{f(\xi) \mid \xi \in \Xi\} \\
\boldsymbol{\mu} &\in (\Xi \stackrel{\text{mon}}{\rightarrow} \Xi)\rightarrow\Xi, \quad \boldsymbol{\mu}(f) \equiv \text{lfp } \lambda\xi.\{c \in \top \mid \forall a.c >^* in\,a \Rightarrow a \in f(\xi)\} \\
\boldsymbol{\nu} &\in (\Xi \stackrel{\text{mon}}{\rightarrow} \Xi)\rightarrow\Xi, \quad \boldsymbol{\nu}(f) \equiv \text{gfp of } \lambda\xi.\{c \in \top \mid out\,c \in f(\xi)\}
\end{aligned}$$

Moreover, $\rightarrow$ is anti-monotonic in its first and monotonic in its second argument, and $\boldsymbol{\Delta}$, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are all monotonic.

□

Well-formedness proof. To show $G_1(\xi_1\rightarrow\xi_2)$, suppose $c \in \xi_1\rightarrow\xi_2$ and $c >^* c'$. Then $c' \in \top$ and given $a \in \xi_1$, it suffices to show $c'\,a \in \xi_2$. But $c\,a >^* c'\,a$ so $c'\,a \in \xi_2$ by $G_1(\xi_2)$. To show $G_2(\xi_1\rightarrow\xi_2)$, suppose $c \in \top$ and if $c >^* t$, then $t \in \xi_1\rightarrow\xi_2$. Given $a \in \xi_1$, it suffices to show $c\,a \in \xi_2$. Note $c\,a \in \top$: an infinite reduction of it must begin:

$$c\,a >^* c'\,a' > b > \cdots \tag{3.1}$$

where $c >^* c'$, $a >^* a'$ and $c'\,a'$ is a redex with contractum b, so c' is critical. But then $c' \in \xi_1\rightarrow\xi_2$ by assumption and $a' \in \xi_1$ by $G_1(\xi_1)$, hence $c'\,a' \in \xi_2$ by definition, so $c'\,a' \in \top$; thus (3.1) is finite, and so $c\,a \in \top$. Now suppose $c\,a >^*$ critical t'. The reduction either looks like (3.1), so there is a $c'\,a' \in \xi_2$ where $c'\,a' >^* t'$ and so $t' \in \xi_2$ by $G_1(\xi_2)$; or like $c\,a >^* c'\,a' = t'$, where $c >^* c'$, and $a >^* a'$. But if t' is critical, so is c'. Then by the same argument we find $t' \in \xi_2$. Conclude $c\,a \in \xi_2$ by $G_2(\xi_2)$ and thus $G_2(\xi_1\rightarrow\xi_2)$.

Given $f \in \Xi\rightarrow\Xi$, $\boldsymbol{\Delta}(f) \in \Xi$ since Ξ is closed under intersection.

Given $f \in \Xi \stackrel{\text{mon}}{\rightarrow} \Xi$, to guarantee the existence of a least fixed point [41], it suffices to show the function (call it g) we are trying to take the least fixed point of is in $\Xi \stackrel{\text{mon}}{\rightarrow} \Xi$. Fix $\xi \in \Xi$. To show $G_1(g(\xi))$, suppose $c \in g(\xi)$ and $c >^* c'$. If $c' >^* in\,a$ then $c >^* in\,a$ and so $a \in f(\xi)$. Conclude $G_1(g(\xi))$. To show $G_2(g(\xi))$, suppose $c \in \top$ and if $c >^* t$ then $t \in g(\xi)$. So if $c >^* in\,a$ then $in\,a \in g(\xi)$, but that implies $a \in f(\xi)$. Conclude $G_2(g(\xi))$, and so $g \in \Xi\rightarrow\Xi$. The monotonicity of g follows easily from f's monotonicity.

Given $f \in \Xi \stackrel{\text{mon}}{\rightarrow} \Xi$, again it suffices to show that the function we are taking the greatest fixed point of (g) is in $\Xi \stackrel{\text{mon}}{\rightarrow} \Xi$. Fix $\xi \in \Xi$. To show $G_1(g(\xi))$, suppose $c \in g(\xi)$ and $c >^* c'$. Then $out\,c >^* out\,c'$, so $out\,c' \in f(\xi)$ by $G_1(f(\xi))$. Conclude $G_1(g(\xi))$. To show $G_2(g(\xi))$, suppose $c \in \top$ and if $c >^* t$ then $t \in g(\xi)$. It suffices to show $out\,c \in f(\xi)$. First, $out\,c \in \top$: an infinite reduction of it must begin:

$$out\, c >^* out\,(\Omega\, a\, b) > a\,(\lambda x.x)(\Omega\, a)\, b > \cdots \tag{3.2}$$

where $c >^* \Omega\, a\, b$, so $\Omega\, a\, b \in g(\xi)$ by assumption, thus $out\,(\Omega\, a\, b) \in f(\xi)$ and so (3.2) is finite. Second, suppose $out\, c >^*$ critical t'. The reduction sequence must be like (3.2), so $out\,(\Omega\, a\, b) >^* t'$, thus $t' \in f(\xi)$ by $G_1(f(\xi))$. Therefore, by $G_2(f(\xi))$, $out\, c \in f(\xi)$. Conclude $G_2(g(\xi))$, and so $g \in \Xi \to \Xi$. The monotonicity of g again follows from f's monotonicity.

The monotonicity and anti-monotonicity properties of all four operations follow easily from their definitions, if, for the last two, we note that the greatest and least fixed point operations are themselves monotonic.
□

Define *environments* to be mappings ρ, from type variables to Ξ. If $\xi \in \Xi$, then let $\rho[\xi/X]$ be the environment where:

$$\rho[\xi/X](Y) \equiv \begin{cases} \xi & \text{If } X \text{ is } Y \\ \rho(Y) & \text{otherwise.} \end{cases}$$

Definition 3.2 *For a given environment ρ, we define its extension, $[\![\_]\!]\rho$, to type expressions, by structural induction:*

$$\begin{array}{rcl} [\![X]\!]\rho & \equiv & \rho(X) \\ [\![A \to B]\!]\rho & \equiv & [\![A]\!]\rho \to [\![B]\!]\rho \\ [\![\Delta X.A]\!]\rho & \equiv & \boldsymbol{\Delta}(\boldsymbol{\lambda}\xi.[\![A]\!]\rho[\xi/X]) \\ [\![\mu X.A]\!]\rho & \equiv & \boldsymbol{\mu}(\boldsymbol{\lambda}\xi.[\![A]\!]\rho[\xi/X]) \\ [\![\nu X.A]\!]\rho & \equiv & \boldsymbol{\nu}(\boldsymbol{\lambda}\xi.[\![A]\!]\rho[\xi/X]) \end{array}$$

□
The following lemma verifies the well-formedness of this definition.

Lemma 3.2 *For type expressions A, and environments ρ and ρ',*

1. *If $Y \notin FV(A)$, then $[\![A]\!]\rho = [\![A[Y/X]]\!]\rho[\rho(X)/Y]$.*
2. *If $\forall X \in FV(A).\ \rho(X) = \rho'(X)$, then $[\![A]\!]\rho = [\![A]\!]\rho'$.*
3. *$[\![A]\!]\rho \in \Xi$.*
4. *$Pos(A, X) \Rightarrow \boldsymbol{\lambda}\xi.[\![A]\!]\rho[\xi/X]$ is a monotonic operation on Ξ.*
5. *$Neg(A, X) \Rightarrow \boldsymbol{\lambda}\xi.[\![A]\!]\rho[\xi/X]$ is an anti-monotonic operation on Ξ.*

Proof. Induct on A. *1-3*: follow from induction and definition 3.1. We need *1* and *2* here to show $[\![\_]\!]\rho$ respects α-equivalence among types, e.g. that $[\![\Delta X.B]\!]\rho = [\![\Delta Y.B[Y/X]]\!]\rho$, when Y does not occur free in B.

4: suppose $Pos(A, X)$. The base case is trivial. If A is $B \to C$, fix $\xi_1, \xi_2 \in \Xi$ such that $\xi_1 \subseteq \xi_2$. By induction, $[\![B]\!]\rho[\xi_1/X] \supseteq [\![B]\!]\rho[\xi_2/X]$ and $[\![C]\!]\rho[\xi_1/X] \subseteq [\![C]\!]\rho[\xi_2/X]$, so by the monotonicity/anti-monotonicity of $\to$, $[\![B \to C]\!]\rho[\xi_1/X] \subseteq [\![B \to C]\!]\rho[\xi_2/X]$. Conclude $\lambda\xi.[\![B \to C]\!]\rho[\xi/X]$ is monotonic. If A is $\Delta Y.B$, fix $\xi, \xi_1, \xi_2 \in \Xi$ such that $\xi_1 \subseteq \xi_2$. By induction (or identity, if X is Y), $[\![B]\!]\rho[\xi_1/X][\xi/Y] \subseteq [\![B]\!]\rho[\xi_2/X][\xi/Y]$, so $\lambda\xi.[\![B]\!]\rho[\xi_1/X][\xi/Y] \sqsubseteq \lambda\xi.[\![B]\!]\rho[\xi_2/X][\xi/Y]$. Thus, we have the containment $[\![\Delta Y.B]\!]\rho[\xi_1/X] \subseteq [\![\Delta Y.B]\!]\rho[\xi_2/X]$ by the monotonicity of $\boldsymbol{\Delta}$. Conclude $\lambda\xi.[\![\Delta Y.B]\!]\rho[\xi/X]$ is monotonic. The same argument proves the remaining cases, and case *5* is exactly like *4*.

□

$\boldsymbol{\lambda\xi}.[\![A]\!]\rho[\xi/X]$ may be monotonic without being continuous: consider $\boldsymbol{\lambda\xi}.[\![\mathbf{1}+X+(\boldsymbol{N} \to X)]\!]\rho[\xi/X]$, where $\boldsymbol{N} \equiv \mu Y.\mathbf{1}+Y$ and $\mathbf{1} \equiv \Delta Z.Z \to Z$.

We now define a notion of truth for judgements. Fix environment ρ; an untyped term $\hat{a}$ is an *instance* of typed term a, with respect to ρ, if:

$$\hat{a} = a[b_1, \ldots, b_n/x_1^{B_1}, \ldots, x_n^{B_n}],$$

with $n \geq 0$, and $b_i \in [\![B_i]\!]\rho$, for $1 \leq i \leq n$. In particular, when $n = 0$, we see the untyped term corresponding to a is an instance of a. Define truth, for judgements, by:

$$\models a : A \quad \equiv \quad \forall\rho.\, \forall\hat{a}.\, \hat{a} \in [\![A]\!]\rho.$$

Here are two standard technical lemmas and a corollary we will need in lemma 3.5.

Lemma 3.3 *For $\xi \in \Xi$, $a[b/x] \in \xi$ and $b \in \top$ imply $(\lambda x.a)b \in \xi$.*

Proof. $\lambda x.a \in \top$. There are two ways it might have an infinite reduction sequence. Either: $\lambda x.a > \lambda x.a' > \lambda x.a'' > \cdots$, but then: $a[b/x] > a'[b/x] > a''[b/x] > \cdots$, which contradicts $a[b/x] \in \top$. The other way is: $\lambda x.a >^* \lambda x.a'x > a' > a'' \cdots$, where x is not free in a'. But then: $a[b/x] >^* a'b > a''b > \cdots$, and again we contradict $a[b/x] \in \top$.

$(\lambda x.a)b \in \top$. There are two ways it might have an infinite reduction sequence. Either: $(\lambda x.a)b >^* (\lambda x.a')b' > a'[b'/x] > \cdots$, where $a >^* a'$ and

$b >^* b'$. But $a[b/x] >^* a'[b'/x]$ and so this reduction converges. The other way is: $(\lambda x.a)b >^* (\lambda x.a'x)b' > a'b' > \cdots$, where $a >^* a'$ and $b >^* b'$, and x does not occur free in a'. But $a[b/x] >^* a'b >^* a'b'$, thus this reduction converges, too.

Now suppose $(\lambda x.a)b >^* t$. The reduction most be like one of the two reductions sequences given in the previous step. In either case, we see $a[b/x] >^* t$, thus $t \in \xi$ by $G_1(\xi)$, and so $(\lambda x.a)b \in \xi$ by $G_2(\xi)$.
□

Lemma 3.4 $[\![A[B/X]]\!]\rho = [\![A]\!]\rho[\xi/X]$, *where* $\xi \equiv [\![B]\!]\rho$.

Proof. By a straightforward induction on A.
□

Corollary 3.1 *For* $\mu \equiv \mu X.A$ *and* $\nu \equiv \nu X.A$,

$$\begin{array}{lcl} [\![\mu]\!]\rho & = & \{c \in \top \mid \forall a.\ c >^* in\ a \Rightarrow a \in [\![A[\mu/X]]\!]\rho\} \\ [\![\nu]\!]\rho & = & \{c \in \top \mid out\ c \in [\![A[\nu/X]]\!]\rho\} \end{array}$$

Now we show soundness.

Lemma 3.5 $a : A \ \Rightarrow\ \models a : A$

Proof. Proof by induction on the derivation of judgements. We consider each clause in turn.

- $in^\mu : A[\mu/X] \to \mu$

Fix $\mu \equiv \mu X.A$, ρ and $a \in [\![A[\mu/X]]\!]\rho$. It suffices to show $in\ a \in [\![\mu]\!]\rho$. Since $a \in \top$, $in\ a \in \top$. If $in\ a >^* in\ a'$ then $a >^* a'$, and $a' \in [\![A[\mu/X]]\!]\rho$ by G_1, hence $in\ a \in [\![\mu]\!]\rho$, by corollary 3.1. Conclude $\models in^\mu : A[\mu/X] \to \mu$.

- $R^\mu : \Delta Y.(\Delta X.(X \to \mu) \to (X \to Y) \to A \to Y) \to \mu \to Y$

Recall how the proof that a monotonic function g on a complete lattice has a least fixed point may proceed. By ordinal induction an ascending chain is defined:

$$\begin{array}{rcl} a_0 & \equiv & \bot \\ a_{\alpha+1} & \equiv & g(a_\alpha) \\ a_\lambda & \equiv & \sqcup\{a_\alpha \mid \alpha < \lambda\} \ \text{ for limit } \lambda. \end{array}$$

Any fixed point is an upper bound of this chain. By a cardinality argument there must be a least ordinal α such that for some $\beta > \alpha$, $a_\alpha = a_\beta$. But by anti-symmetry, $a_\alpha = a_{\alpha+1}$, and so a_α must be the least fixed point of g.

Fix $\mu \equiv \mu X.A$, ρ, $\xi \in \Xi$ and let $\rho_1 \equiv \rho[\xi/Y]$. This chain for $[\![\mu]\!]\rho_1$ is:

$$\begin{array}{rcl} \xi_0 & \equiv & \bot \\ \xi_{\alpha+1} & \equiv & \{c \in \top \mid \forall a.\ c >^* in\, a \Rightarrow a \in [\![A]\!]\rho_1[\xi_\alpha/X]\} \\ \xi_\lambda & \equiv & \bigcup_{\alpha<\lambda} \xi_\alpha, \text{ for limit } \lambda. \end{array}$$

Fix $a \in [\![\Delta X.(X \to \mu) \to (X \to Y) \to A \to Y]\!]\rho_1$. It suffices to prove:

$$\forall b \in [\![\mu]\!]\rho_1.\ R\, a\, b \in \xi,$$

which we will do by induction on the chain with limit $[\![\mu]\!]\rho_1$.

Base case: $b \in \xi_0$. Since b and a are in $\top$, and since b cannot reduce to a term of the form $in\, b'$, $R\, a\, b \in \bot$, so $R\, a\, b \in \xi$ by $G_2(\xi)$.

Inductive case: $b \in \xi_{\alpha+1}$. If b cannot reduce to a term of the form $in\, b'$, the argument is as above. Otherwise, let $\rho_2 \equiv \rho_1[\xi_\alpha/X]$. Note $[\![\mu]\!]\rho = [\![\mu]\!]\rho_1 = [\![\mu]\!]\rho_2$ by lemma 3.2. $R\, a\, b \in \top$: an infinite reduction of it must begin:

$$R\, a\, b >^* R\, a'\, (in\, b') > a'(\lambda x.x)(R\, a')b' > \cdots \tag{3.3}$$

where $a >^* a'$ and $b >^* in\, b'$. Since $b \in \xi_{\alpha+1}$, $b' \in [\![A]\!]\rho_2$. But (i) a is in $[\![(X \to \mu) \to (X \to Y) \to A \to Y]\!]\rho_2$ by the definition of $\boldsymbol{\Delta}$, and so is a', by G_1; (ii) since $\xi_\alpha \subseteq [\![\mu]\!]\rho_2$, $\lambda x.x \in [\![X \to \mu]\!]\rho_2$ by lemma 3.3; (iii) $R\, a$ is in $[\![X \to Y]\!]\rho_2$, by induction, and so is $R\, a'$, by G_1; and by these facts, $a'(\lambda x.x)(R\, a')b' \in \xi$, so (3.3) is finite and $R\, a\, b \in \top$. Now suppose $R\, a\, b >^* t$. The reduction sequence must look like (3.3), so there is a $a'(\lambda x.x)(R\, a')(in\, b') \in \xi$ that reduces to t. Thus, $t \in \xi$ by $G_1(\xi)$, and therefore $R\, a\, b \in \xi$ by $G_2(\xi)$. Conclude $\models R^\mu : \Delta Y.(\Delta X.(X \to \mu) \to (X \to Y) \to A \to Y) \to \mu \to Y$.

We can note here that a recursion combinator over simultaneously defined recursive types, where each defining type expression is positive in all the type variables being bound, will be valid by this same argument.

- $out^\nu : \nu \to A[\nu/X]$

Fix $\nu \equiv \nu X.A$, ρ and $c \in [\![\nu]\!]\rho$. It suffices to show $out\, c \in [\![A[\nu/X]]\!]\rho$, but that is immediate from lemma 3.1. Conclude $\models out^\nu : \nu \to A[\nu/X]$.

- $\Omega^\nu : \Delta Y.(\Delta X.(\nu \to X) \to (Y \to X) \to Y \to A) \to Y \to \nu$

Fix $\nu \equiv \nu X.A$, ρ, $\xi \in \Xi$, $a \in [\![(\Delta X.(\nu \to X) \to (Y \to X) \to Y \to A)]\!]\rho[\xi/Y]$, and $b \in \xi$. It suffices to show:

$$\Omega\, a\, b \in [\![\nu]\!]\rho[\xi/Y],$$

which is proven by induction on the descending chain with limit $[\![\nu]\!]\rho[\xi/Y]$, in a similar argument to the proof for R.

- $x^A : A$

Fix ρ. Given instance $\widehat{x^A}$ of x^A, there are two cases. If $\widehat{x^A}$ is x then $x \in [\![A]\!]\rho$ by $G_2(A)$; otherwise $\widehat{x^A}$ is b for some $b \in [\![A]\!]\rho$ by the definition of instance. Either way, conclude $\models x^A : A$.

- $$\frac{b : B}{\lambda x^A.b : A \to B}$$

Suppose $\models b : B$. Fix ρ. Given instance $\lambda x.\hat{b}$ of $\lambda x^A.b$, we must show it is in $[\![A \to B]\!]\rho$. First, $\lambda x.\hat{b} \in \top$: by assumption $\hat{b} \in [\![B]\!]\rho$, so $\hat{b} \in \top$ and thus $\lambda x.\hat{b} \in \top$. Second, given $a \in [\![A]\!]\rho$, $(\lambda x.\hat{b})a \in [\![B]\!]\rho$: $\hat{b}[a/x]$ is itself an instance of b so $\hat{b}[a/x] \in [\![B]\!]\rho$, by assumption; $a \in \top$ and so by lemma 3.3, $(\lambda x.\hat{b})a \in [\![B]\!]\rho$. Conclude $\models \lambda x^A.b : A \to B$.

- $$\frac{c : A \to B \quad a : A}{c\, a : B}$$

Suppose $\models c : A \to B$ and $\models a : A$. Fix ρ. Given instance $\widehat{c\,a} = \hat{c}\,\hat{a}$, by assumption, $\hat{c} \in [\![A \to B]\!]\rho$ and $\hat{a} \in [\![A]\!]\rho$. Thus, $\hat{c}\,\hat{a} \in [\![B]\!]\rho$, by the definition of $\to$. Conclude $\models c\, a : B$.

- $$\frac{b : B}{\Lambda X.b : \Delta X.B}$$

Suppose $\models b : B$. Fix ρ, and instance $\hat{b}$ of $\Lambda X.b$. Note that $\hat{b}$ is also an instance of b with respect to ρ, and by lemma 3.2, because of the restriction that no free variable in b has a type involving X, $\hat{b}$ is in fact an instance of b with respect to $\rho[\xi/X]$ for any $\xi \in \Xi$. Thus, by assumption, $\hat{b} \in [\![B]\!]\rho[\xi/X]$, so $\hat{b} \in [\![\Delta X.B]\!]\rho$ by the definition of $\boldsymbol{\Delta}$. Conclude $\models \Lambda X.b : \Delta X.B$.

- $$\frac{b : \Delta X.B}{b\, A : B[A/X]}$$

Suppose $\models b : \Delta X.B$. Fix ρ, and instance $\hat{b}$ of $b\,A$. Let $\xi \equiv [\![A]\!]\rho$; by assumption and the definition of $\boldsymbol{\Delta}$, $\hat{b} \in [\![B]\!]\rho[\xi/X]$, so by lemma 3.4, $\hat{b} \in [\![B[A/X]]\!]\rho$. Conclude $\models b\,A : B[A/X]$.
□

Lemma 3.6 *A typed term is strongly normalizable if the corresponding untyped term is strongly normalizable.*

Proof. Fix typed term a. The corresponding untyped term, written $|a|$, is usually referred to as the *stripped* term. Given a reduction sequence of typed terms, $a_1 > a_2 > \cdots$, for the corresponding sequence $|a_1|, |a_2|, \ldots$ either $|a_i| > |a_{i+1}|$ or $|a_i| = |a_{i+1}|$, and the latter implies a_i reduces to a_{i+1} by reductions 3 or 4, the type expression β and η reductions. Since these reductions reduce the number of Λ's in a term by one, only a finite number of them can occur in a row. Thus, the sequence of stripped terms corresponding to an infinite reduction of a will contain an infinite reduction of $|a|$. Contradiction.
□
Now, this section's result follows easily.

Theorem 3.1 *All typed terms are strongly normalizable.*

Proof. Suppose $a : A$. By lemma 3.5, $\models a : A$. Choose any environment ρ. By the definition of truth, $\hat{a} \in [\![A]\!]\rho$, for any instance $\hat{a}$ of a, in particular for $|a|$. Since $[\![A]\!]\rho \in \Xi$, we have $[\![A]\!]\rho \subseteq \top$, so $|a| \in \top$. Therefore, by lemma 3.6, we conclude a is strongly normalizable.
□

3.2 Equational type constraints

The goal of this section is to give a condition **P** on the set of type constraints which holds exactly when the resulting typed terms are strongly normalizable. The section is organized as follows. First, types, terms and reduction are defined. Second, we state condition **P** and show how its violation leads to the typing of diverging terms. Third, we give an equivalent formulation of **P** which identifies equivalence classes $[i]$ on I and a total ordering $\prec$ of these classes. Given this section's notion of type membership, we wish to

repeat the previous section's proof, using $\prec$ in defining a variation, $[\![\_]\!]'$, of $[\![\_]\!]$, but two complications appear: $\prec$ may not be well-founded, so there is no obvious well-founded order to type expressions; and in defining certain $[\![\tau_i]\!]'$ simultaneously, we must find fixed points of sets of operations which may not be monotonic, with respect to the $\subseteq$-ordering on Ξ. Fortunately, each complication can be overcome in turn and we may outline the proof of strong normalization as follows.

1. Let the definitions of $\top$, Ξ, environments ρ, $\longrightarrow$ and $\boldsymbol{\Delta}$ stand as in section 1.

2. Fix a derivation of $a^\star : A^\star$ and let I' be the union of classes $[i]$ for which τ_i appears in this derivation. Define level ordinals for type expressions with respect to I'.

3. Environments ρ are extended to mappings $[\![\_]\!]'\rho$ from type expressions to Ξ by level induction.

4. Define instances $\hat{a}$ as before and truth for the two forms of judgement by:

$$\begin{aligned} \models a : A &\equiv \forall\rho.\forall\hat{a}.\ \hat{a} \in [\![A]\!]'\rho \\ \models A = B &\equiv \forall\rho.\ [\![A]\!]'\rho = [\![B]\!]'\rho \end{aligned}$$

5. We can not expect soundness to hold, since we have made no effort to ensure $\models \tau_i = T_i$ for $i \notin I'$. However, by lemma 3.10, the other axioms and rules are sound, so we may conclude $\models a^\star : A^\star$.

6. As in the first section, from 5 we can argue that $a^\star$ is strongly normalizable. Since $a^\star$ was arbitrary, **P** implies all typed terms are strongly normalizable.

3.2.1 Type expressions and terms

We define type expressions as in the previous section, with the addition of atomic types τ_i for i in the fixed index set I. For notational simplicity, assume the type variables V_i are indexed by a superset of I. Given closed type expressions T_i for $i \in I$, define judgements by figure 3.2.

$$\frac{}{\tau_i = T_i}\ i \in I \qquad\qquad A = A$$

$$\frac{A = B}{B = A} \qquad\qquad \frac{A = B \quad B = C}{A = C}$$

$$\frac{A = B}{C = C'}\ \dagger \qquad\qquad \frac{a : A \quad A = B}{a : B}$$

$$x^A : A$$

$$\frac{b : B}{\lambda x^A.b : A \to B} \qquad\qquad \frac{b : B}{\Lambda X.b : \Delta X.B}\ *$$

$$\frac{c : A \to B \quad a : A}{c\,a : B} \qquad\qquad \frac{b : \Delta X.B}{b\,A : B[A/X]}$$

† *C' is C with some occurrences of A replaced by B.*
* *X is not free in A for any x^A free in b.*

Figure 3.2: *Definition of typed terms with equational constraints*

3.2.2 A condition on the constraints

The required condition **P** is that τ_i occurs positively in all types C equal to τ_i, that is:

P: $\quad \forall C, \tau_i.\, (C = \tau_i) \Rightarrow Pos(C, \tau_i).$

It is an easy exercise to given a polynomial algorithm for **P** when I is finite. If **P** is violated, C can be used in typing a diverging term. We illustrate now by assuming that $\tau = C$ for some C built using only $\rightarrow$, where τ occurs non-positively in C. Then C is C_{2k+1}, which is in the following form:

$$\begin{array}{rcl} C_{2k+1} & \equiv & A^1_{2k+1} \rightarrow \cdots \rightarrow A^{n_{2k+1}}_{2k+1} \rightarrow (C_{2k}) \rightarrow B_{2k+1} \\ & \vdots & \\ C_1 & \equiv & A^1_1 \rightarrow \cdots \rightarrow A^{n_1}_1 \rightarrow (C_0) \rightarrow B_1 \\ C_0 & \equiv & A^1_0 \rightarrow \cdots \rightarrow A^{n_0}_0 \rightarrow \tau. \end{array}$$

Some abbreviations are called for:

$$\begin{array}{rcl} \lambda \vec{x}_i.a & \equiv & \lambda x^1_i. \cdots \lambda x^{n_i}_i.a \\ t\, \vec{a}_i & \equiv & t\, a^1_i \cdots a^{n_i}_i. \end{array}$$

Assume in the following that a^j_i and x^j_i are variables of type A^j_i, y_i is a variable of type C_i and f^j_i is a variable of type $B_j \rightarrow B_i$. For the case $k = 0$ we define the terms c_i of type C_i as follows:

$$\begin{array}{rcl} c_1 & \equiv & \lambda \vec{x}_1.\lambda y_0.f^1_1(y_0\, \vec{a}_0\, \vec{a}_1\, y_0) \\ c_0 & \equiv & \lambda \vec{x}_0.c_1. \end{array}$$

Now we reduce $c_1\, \vec{a}_1\, c_0$:

$$>^* \quad f^1_1(c_0\, \vec{a}_0\, \vec{a}_1\, c_0) \quad >^* \quad f^1_1(c_1\, \vec{a}_1\, c_0).$$

We conclude this term is not normalizable. For the case $k > 0$ we define the terms c_i of type C_i as follows:

$$
\begin{array}{rcl}
c_1 & \equiv & \lambda\vec{x}_1.\lambda y_0.f_1^{2k+1}(y_0\ \vec{a}_0\ \vec{a}_{2k+1}\ y_{2k}) \\
c_3 & \equiv & \lambda\vec{x}_3.\lambda y_2.f_3^2(y_2\ \vec{a}_2\ c_1) \\
 & \vdots & \\
c_{2k+1} & \equiv & \lambda\vec{x}_{2k+1}.\lambda y_{2k}.f_{2k+1}^{2k}(y_{2k}\ \vec{a}_{2k}\ c_{2k-1}) \\
 & & \\
c_0 & \equiv & \lambda\vec{x}_0.c_{2k+1} \\
c_2 & \equiv & \lambda\vec{x}_2.\lambda y_1.f_2^1(y_1\ \vec{a}_1\ c_0) \\
 & \vdots & \\
c_{2k} & \equiv & \lambda\vec{x}_{2k}.\lambda y_{2k-1}.f_{2k}^{2k-1}(y_{2k-1}\ \vec{a}_{2k-1}\ c_{2k-2})
\end{array}
$$

Now we reduce $c_{2k+1}\ \vec{a}_{2k+1}\ c_{2k}$:

$$
\begin{array}{ll}
>^* f_{2k+1}^{2k}(c_{2k}\ \vec{a}_{2k}\ c_{2k-1}[c_{2k}/y_{2k}]) & >^* \cdots(c_{2k-1}[c_{2k}/y_{2k}]\ \vec{a}_{2k-1}\ c_{2k-2}) \\
\vdots & \\
>^* \cdots(c_2\ \vec{a}_2\ c_1[c_{2k}/y_{2k}])\cdots & >^* \cdots(c_1[c_{2k}/y_{2k}]\ \vec{a}_1\ c_0)\cdots \\
>^* \cdots(c_0\ \vec{a}_0\ \vec{a}_{2k+1}\ c_{2k})\cdots & >^* \cdots(c_{2k+1}\vec{a}_{2k+1}\ c_{2k})\cdots
\end{array}
$$

We conclude this term is not normalizable.

As an example, consider the second example given in the introduction of this chapter:

$$
\begin{array}{rcl}
\tau_0 & = & \tau_1 \to \tau_0 \\
\tau_1 & = & \tau_2 \to \tau_1 \\
\tau_2 & = & \tau_0 \to \tau_2.
\end{array}
$$

Here, $\tau_0 = ((\boxed{\tau_0} \to \tau_2) \to \tau_1) \to \tau_0$, and the right-hand type expression contains a non-positive occurrence of τ_0, as highlighted. The diverging term the previous prescription would construct is:

$(c_3)\lambda y^{\tau_0\to\tau_2}.w^{\tau_2\to\tau_1}(y\ c_3)$, where
$c_3 \equiv \lambda z^{(\tau_0\to\tau_2)\to\tau_1}.v^{\tau_1\to\tau_0}(z\lambda x^{\tau_0}.u^{\tau_0\to\tau_2}(xz))$.

The following states an equivalent formulation of **P**, which will prove more useful in the next subsection, when we will be looking for an ordering to type expressions.

Lemma 3.7 *Condition* **P** *is equivalent to the existence of an equivalence relation on I, whose classes $[i]$ are ordered by a relation $\prec$ such that if τ_i appears in T_j then $[i] = [j]$ or $[i] \prec [j]$. Furthermore, each class $[i]$ is divided in two parts, $[i]^+$ and $[i]^-$, and $j \in [i]^+ \Rightarrow Pos(T_i, \tau_j)$ and $j \in [i]^- \Rightarrow Neg(T_i, \tau_j)$.*

The remainder of this subsection is devoted to a proof of this. For notational convenience, we write $A(i)$ to indicate τ_i occurs in A and then let $A(B)$ stand for $A[B/\tau_i]$. Define:

$$\begin{aligned} i \leq j &\equiv \exists A(i).\tau_j = A(i) \\ i < j &\equiv \neg j \leq i \\ i \prec_0 j &\equiv i \leq j \wedge i < j \\ i \approx j &\equiv i \leq j \wedge j \leq i. \end{aligned}$$

Relation $\approx$ is an equivalence relation, and we write $[i]$ for the equivalence class containing i. If we temporarily define $[i] \prec_1 [j] \equiv i \prec_0 j$, we see the classes are only partially ordered by $\prec_1$, so extend relation $\prec_1$ to a total ordering $\prec_2$ and define $i \prec j \equiv [i] \prec_2 [j]$. Define:

$$A \ \& \ B \ \textit{agree on} \ i \equiv Pos(A, \tau_i) \wedge Pos(B, \tau_i) \vee Neg(A, \tau_i) \wedge Neg(B, \tau_i)$$

Lemma 3.8

1. $\forall C(i).\ C(A)$ & $C(B)$ *agree on* $i \Rightarrow A$ *and* B *agree on* i.
2. *If* $i \approx j$, $\tau_i = A(j)$ *and* $\tau_i = B(j)$ *then* $A(j)$ & $B(j)$ *agree on* j.

Proof.

1. By induction on $C(i)$.
2. Since $i \approx j$, choose $C(i)$ such that $\tau_j = C(i)$. Thus, $\tau_j = C(A(j))$, $\tau_j = C(B(j))$ and by **P**, $Pos(C(A(j)), \tau_j)$ and $Pos(C(B(j)), \tau_j)$. Therefore $A(j)$ & $B(j)$ agree on j by 1.

□

Definition 3.3 *If* $i \approx j$, *and* $A(i) = \tau_j$, *define:*

$$\begin{aligned} i \simeq j &\equiv Pos(A(i), \tau_i) \\ i \asymp j &\equiv Neg(A(i), \tau_i). \end{aligned}$$

□

By $i \approx j$, such an $A(i)$ exists; by lemma 3.8, these definitions are independent of the choice of $A(i)$ and $\tau_j = A(i)$ implies exactly one of $Pos(A(i), \tau_i)$ or $Neg(A(i), \tau_i)$, so exactly one of $i \simeq j$ and $i \asymp j$ holds for $i \approx j$.

Lemma 3.9

1. $Pos(A(j), \tau_j) \wedge Pos(A(B), \tau_i)$ *or* $Neg(A(j), \tau_j) \wedge Neg(A(B), \tau_i)$ *implies* $Pos(B, \tau_i)$.
2. $\simeq$ *is an equivalence relation.*
3. $Neg(A(j), \tau_j) \wedge Pos(A(B), \tau_i)$ *or* $Pos(A(j), \tau_j) \wedge Neg(A(B), \tau_i)$ *implies* $Neg(B, \tau_i)$.
4. $\asymp$ *is symmetric and* $i \asymp j \wedge j \asymp k \Rightarrow i \simeq k$

Proof.

1. By induction on $A(j)$.
2. The reflexivity of $\simeq$ is immediate.

   For symmetry, suppose $j \simeq i$, $\tau_i = A(j)$ and $\tau_j = B(i)$. By **P**, $Pos(A(B(i)), \tau_i)$, and by $j \simeq i$, $Pos(A(j), \tau_j)$; thus $Pos(B(i), \tau_i)$ by 1, so we conclude $i \simeq j$.

   For transitivity, suppose $i \simeq j$, $j \simeq k$, $\tau_j = A(i)$, $\tau_k = B(j)$ and $\tau_i = C(k)$. By **P**, $Pos(B(A(C(k))), \tau_k)$, and by $j \simeq k$, $Pos(B(j), \tau_j)$; thus $Pos(A(C(k)), \tau_k)$ by 1. By $i \simeq j$, $Pos(A(i), \tau_i)$, and $Pos(A(C(k)), \tau_k)$ implies $Pos(C(k), \tau_k)$ by 1, so we conclude $i \simeq k$.
3. By induction on $A(j)$.
4. The symmetry of $\asymp$ follows by the same argument used for the symmetry of $\simeq$, using 3 in place of 1.

   Suppose $i \asymp j$, $j \asymp k$, $\tau_j = A(i)$, $\tau_k = B(j)$ and $\tau_i = C(k)$. By **P**, $Pos(B(A(C(k))), \tau_k)$, and by $j \asymp k$, $Neg(B(j), \tau_j)$; thus we may conclude $Neg(A(C(k)), \tau_k)$ by 3. By $i \asymp j$, $Neg(A(i), \tau_i)$, and taken with $Neg(A(C(k)), \tau_k)$ that implies $Pos(C(k), \tau_k)$ by 1, and thus we conclude $i \simeq k$.

□

So, by lemma 3.9 we see $[i]$ is divided into at most two parts and $i \simeq j$ ($i \asymp j$) when i and j are in the same part (different parts) of $[i]$, and most importantly, $i \simeq j \Rightarrow Pos(T_i, \tau_j)$ and $i \asymp j \Rightarrow Neg(T_i, \tau_j)$. Finally, we can define:

$$
\begin{aligned}
[i]^+ &\equiv \{j \in [i] \mid i \simeq j\} \\
[i]^- &\equiv \{j \in [i] \mid i \asymp j\},
\end{aligned}
$$

and thus, we have proven lemma 3.7.

3.2.3 Strong normalization

In this subsection we assume **P** and from that argue that all typed terms are strongly normalizable. Let the definitions of Ξ, environments, $\longrightarrow$ and $\boldsymbol{\Delta}$ stand as in 3.1.2. We would like to again extend the environment ρ to a mapping $[\![\_]\!]'\rho$ from type expressions to Ξ, but we can not do structural induction on type expressions, because of atomic types τ_i. Order $\prec$ should be involved, but it is not necessarily well-founded: consider the case when I is the set of natural numbers and T_i is $\tau_{i+1} \rightarrow \tau_{i+1}$. The solution is to fix a derivation of $a^\star : A^\star$ and only be concerned with τ_i that occur there. Once we have concluded $a^\star$ is strongly normalizable, we can generalize to prove the final result.

So, fix a derivation of $a^\star : A^\star$ and let I' be the union of classes $[i]$ for τ_i appearing in this derivation. For $i \in I'$, let its rank, $\#i$, be 1 plus the finite number of distinct classes $[j] \subseteq I'$ for which $[j] \prec [i]$. Now we can define a level ordinal for each type expression:

$$
\begin{aligned}
\mathcal{L}(X) &\equiv 0 & \\
\mathcal{L}(\tau_i) &\equiv 0 & \text{if } i \notin I' \\
\mathcal{L}(B \rightarrow C) &\equiv sup(\mathcal{L}(B), \mathcal{L}(C)) + 1 & \\
\mathcal{L}(\Delta X.B) &\equiv \mathcal{L}(B) + 1 & \\
\mathcal{L}(\tau_i) &\equiv \omega * (\#i) & \text{if } i \in I'.
\end{aligned}
$$

We wish to extend an environment ρ by level induction to a mapping $[\![\_]\!]'\rho$, but a second complication emerges: at level $\omega * (\#i)$ we will want to find a fixed point of the operation $f \in \Xi^{[i]} \rightarrow \Xi^{[i]}$, where:

$$
f(\langle \xi_j \rangle_{j \in [i]}) \equiv \langle [\![T_k[V_j/\tau_j]_{j \in [i]}]\!]' \rho[\xi_j/V_j]_{j \in [i]} \rangle_{k \in [i]}. \tag{3.4}
$$

This is may not be monotonic, if we order $\Xi^{[i]}$ by the usual product ordering:

$$
\langle \xi_j \rangle_{j \in [i]} \sqsubseteq \langle \xi'_j \rangle_{j \in [i]} \equiv \forall j \in [i].\ \xi_j \subseteq \xi'_j,
$$

but f *is* monotonic if we choose the ordering:

$$\langle \xi_j \rangle_{j\in[i]} \sqsubseteq \langle \xi'_j \rangle_{j\in[i]} \equiv \forall j \in [i]^+.\ \xi_j \subseteq \xi'_j \wedge \forall j \in [i]^-.\ \xi_j \supseteq \xi'_j.$$

An example of such a situation is $\tau_0 = \tau_1 \to \tau_0$ and $\tau_1 = \tau_0 \to \tau_1$. This ordering means that taking the least fixed point of f will yield the least solutions of ξ_j for $j \in [i]^+$ and the greatest solutions for $j \in [i]^-$, but as it will turn out, *any* fixed point will suffice.

Finally, we can define $[\![\_]\!]'\rho$ by level induction:

$$\begin{array}{rcl} [\![\tau_i]\!]'\rho & \equiv & \bot, \quad \text{for } i \notin I' \\ [\![X]\!]'\rho & \equiv & \rho(X) \\ [\![A \to B]\!]'\rho & \equiv & [\![A]\!]'\rho \to [\![B]\!]'\rho \\ [\![\Delta X.A]\!]'\rho & \equiv & \boldsymbol{\Delta}(\boldsymbol{\lambda}\xi.[\![A]\!]'\rho[\xi/X]), \end{array}$$

and at level $\omega * (\#i)$, let $\langle [\![\tau_j]\!]'\rho \rangle_{j\in[i]}$ be the least fixed point of f, as defined in (3.4). Reinterpreting lemma 3.2, lemma 3.4 and corollary 3.1 in the context of this section, we see they hold by the same arguments. We may draw another corollary from lemma 3.4:

Corollary 3.2 *If $i \in I'$, then $\forall\rho.\ [\![\tau_i]\!]'\rho = [\![T_i]\!]'\rho$.*

Define *instance* as in 3.1.2 and truth for the two forms of judgement by:

$$\begin{array}{rcl} \models a : A & \equiv & \forall\rho.\forall\hat{a}.\ \hat{a} \in [\![A]\!]'\rho \\ \models A = B & \equiv & \forall\rho.\ [\![A]\!]'\rho = [\![B]\!]'\rho. \end{array}$$

As noted earlier, there is no reason to believe soundness will hold, because we have made no effort to ensure $\models \tau_i = T_i$ for $i \notin I'$, but we can show enough to conclude $\models a^\star : A^\star$:

Lemma 3.10 *Except for $\tau_i = T_i$ when $i \notin I'$, all the axioms and rules of figure 3.2 are sound.*

Proof. We consider each clause in turn. The arguments for the rules and axioms carried over from figure 3.1 are as in lemma 3.5. For $i \in I'$, $\models \tau_i = T_i$ holds by corollary 3.2. The rules for the reflexivity, symmetry and transitivity of $A = B$ are trivial to verify. To show $A = B \Rightarrow C = C'$, where for a suitable C'', $C' \equiv C''[B/X]$, $C \equiv C''[A/X]$ and X does not occur free in C, let $\xi \equiv [\![A]\!]'\rho = [\![B]\!]'\rho$; then the argument follows easily from lemma 3.4:

$$[\![C]\!]'\rho = [\![C''[A/X]]\!]'\rho = [\![C'']\!]'\rho[\xi/X] = [\![C''[B/X]]\!]'\rho = [\![C']\!]'\rho.$$

Finally, the soundness of $a : A \wedge A = B \Rightarrow a : B$ is immediate from the definition of truth.
□
Proposition 3.10 implies $\models a^\star : A^\star$, and as in 3.1.2, this implies $a^\star$ is strongly normalizable. Since $a^\star$ was arbitrary, we have proven the following.

Theorem 3.2 P *implies all typed terms are strongly normalizable.*

Summary

We have shown how to add general inductive type constructors to the second-order lambda calculus, while preserving the strong normalizability property of terms, and to decide when a collection of type constraints admits the typing of only strongly normalizable terms. In both cases a similar method of proof, based on the method of Girard, was employed.

We are now ready to give a semantics for the basic intuitionistic type theory and for its extension by inductive types. These proofs will require an involved structure just to identify the well-formed types, but the core of the argument concerning the inductive types will be a recognizable variation on what was presented in this chapter.

Chapter 4

Semantic Account of the Basic Theory

In this chapter we develop a semantic account of the basic type theory in order to show the basis on which we will build the more complex relativized construction of the next chapter. An important conclusion we will draw from this semantics will be the intuitionistic consistency of the theory.

This proof follows the general argument used by Tait [40] to show the strong normalization property of the simply typed lambda calculus, which was relativized by Girard [22,21] to show this property for the second-order, polymorphic, lambda calculus. In both cases, one can phrase their arguments in terms of the construction of a mapping from type expressions to some fixed set, defined by *induction on the structure* of type expressions. We have seen this method used twice in chapter 3, namely the mappings $[\![\_]\!]$ and $[\![\_]\!]'$.

With dependent type constructors, such as Π, Σ and $\{\_ \mid \_\}$, type universes U_i and equality types I, one can no longer define this mapping in such a straightforward manner, because type expressions and terms are now simultaneously defined. A solution for such a theory is to identify a well-founded order in which types may be thought of as being constructed, and to allow the semantics to be non-compositional, by interleaving evaluation with decomposition in its statement. We achieve this by viewing type construction as a monotonic operation on a suitable complete partial order. This is sufficient to give a semantics for a predicative type system, stratified by type universes.

We now outline the developments of this chapter. For a detailed study of such semantics, the reader is referred to [2].

1. In our model, a type is represented by a partial equivalence relation on closed, normalizing terms. In 4.1 we define Ξ, a collection of the so-called ground relations, to serve as the universe of possible relations.

2. For each type constructor, it is a straightforward matter to define a mapping which takes ground relations, or families of ground relations, to Ξ in such a way as to model the computational meaning of that type constructor. For example, given $\xi \in \Xi$ and a function ψ mapping equivalence classes of ξ to Ξ, we define $\boldsymbol{\Sigma}(\xi, \psi) \in \Xi$ by:

$$a = a' \in \boldsymbol{\Sigma}(\xi, \psi) \Leftrightarrow a \geq \langle b,\ c\rangle \wedge a' \geq \langle b',\ c'\rangle \wedge b \xi b' \wedge c(\psi[b]_\xi)c'.$$

3. In 4.2, the collection of type systems, $\mathcal{TS}$, is defined. For the purposes of this construction, a type system is a pair $\langle \tau,\ [\![\_]\!]\rangle$, where ground relation τ defines equality between terms representing types, and $[\![\_]\!]$ associates a ground relation representing membership with each equivalence class of types. We can view $\mathcal{TS}$ as a complete partial order under an ordering meant to capture the notion of consistently *extending* a type system — enlarging τ while maintaining the same membership relations on the preserved types.

4. Using the semantic operations of 2, we define an $\omega + 1$ sequence of monotonic operations on $\mathcal{TS}$: $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_\omega$, where $\mathcal{F}_\alpha$ maps a given type system to one containing the atomic types *void* and U_i for $i < \alpha$, and the compound types $A + B$, $(\Sigma x{:}A)B$, $(\Pi x{:}A)B$, $\{x{:}A |\ B\}$, and $I(a, b, A)$ whose immediate subtypes are taken from that given type system.

5. To take the least fixed point of such a $\mathcal{F}_\alpha$ is to define a type system $\langle \tau_\alpha,\ [\![\_]\!]_\alpha\rangle$ with the listed atomic types and inductively close it under the listed type constructors. Thus, we can reguard $\langle \tau_i,\ [\![\_]\!]_i\rangle$, for $i < \omega$, as a model of U_i and $\langle \tau_\omega,\ [\![\_]\!]_\omega\rangle$ as a model for the entire type hierarchy.

6. With $\langle \tau_\omega,\ [\![\_]\!]_\omega\rangle$ in hand, we define the predicates A *Type*, $A = B$, $a = b \in A$ and $a \in A$, and in lemmas 4.3–4.6 compile a number of

properties of these four predicates, almost all of which can be read off the definitions of Ξ and $\mathcal{F}_\alpha$. These are the properties of $\langle \tau_\omega, [\![\_]\!]_\omega \rangle$ which will be used in the soundness proof.

7. In 4.3, truth for sequents is defined and we verify the soundness of the proof rules.

8. Once the soundness of the proof rules has been established, in 4.4 we conclude all derivable sequents are true. For $\vdash a = b \in A$ this implies that $a = b \in A$, and we may conclude all the properties given in lemmas 4.3–4.6 about this predicate, including the non-inhabitation of *void*, which is the statement of intuitionistic consistency.

Viewed under the propositions-as-types principle, intuitionistic consistency is propositional consistency — *False* can not be derived.

4.1 Ground relations

Types will be modeled by partial equivalence relations over closed, normalizing terms. We begin with some basic definitions and notations for partial equivalence relations.

- A binary relation ξ over a set T is a *partial equivalence relation (PER) over* T iff it is symmetric and transitive. T may be dropped if it is clear from context.

- If ξ is a PER over T, let T/ξ be the set of nonempty equivalence classes that ξ defines. For $a \in T$, define:

  $$[a]_\xi \equiv \{b \in T \mid a \xi b\}.$$

  If ξ is clear from the context, we may simply write $[a]$. Write $Fi(\xi)$ for the field of relation ξ. Note that PER ξ is an equivalence relation over $Fi(\xi)$.

- If R is a relation over T, then PER ξ *respects* R, iff

  $$(a \in Fi(\xi) \vee b \in Fi(\xi)) \wedge aRb \Rightarrow a \xi b.$$

- For a collection S of PERs over T, and a given PER ξ, define:

$$\begin{aligned} FAM(\xi, S) &\equiv T/\xi \to S \\ FAM(S) &\equiv \bigcup_{\xi \in S} FAM(\xi, S) \\ IFAM(S) &\equiv \sum_{\xi \in S} FAM(\xi, S). \end{aligned}$$

As usual, we will be identifying terms that are alphabetic variants in their bound variables. Let $\mathcal{T}$ be the set of closed terms, $\geq$ the evaluation relation and $\mathcal{V}$ the set of closed, normalizing terms. Define $a \downarrow \equiv a \in \mathcal{V}$, and $a \simeq b \equiv \exists c.\, a \geq c \wedge b \geq c$. Let Ξ be the collection of PERs over $\mathcal{V}$ that respect $\geq$. Elements of Ξ are called *ground relations*. We will discuss its appropriateness after the definition of $\mathcal{TS}$. The metavariables we will use are as follows.

- i, j and k range over positive numbers.
- v, w, x, y and z range over variables.
- a, b, c, and d and A, B, C and D range over open terms.
- $\langle \xi,\ \psi \rangle$ ranges over $IFAM(\Xi)$.
- σ and $\langle \tau,\ [\![\_]\!] \rangle$ also range over $IFAM(\Xi)$.

The choice of an upper or lower case roman letter for an open term is meant to imply a term is playing the role of an element of a type, or a type, but there can be no semantic distinction, as such. We introduce the following notation for PERs.

$$\begin{aligned} A \in \xi &\equiv A \xi A \\ A = A' \in \xi &\equiv A \xi A' \\ \forall A \in \xi.\, P &\equiv \forall A.\, A \xi A \Rightarrow P \\ \forall A = A' \in \xi.\, P &\equiv \forall A, A'. A \xi A' \Rightarrow P \\ [\![A]\!] &\equiv [\![[A]]\!] \end{aligned}$$

4.1.1 Operations on ground relations

At this point in the construction, it is convenient to isolate the following functions. Their names are suggestive: $\boldsymbol{\Pi}$, for example, will define the semantic operation that will be the meaning given to a Π type.

Definition 4.1

1. *Define* $\boldsymbol{\Pi} \in IFAM(\Xi) \rightarrow \Xi$ *by:*

$$a = a' \in \boldsymbol{\Pi}(\xi, \psi) \equiv \begin{array}{l} a \in \mathcal{V} \wedge a' \in \mathcal{V} \wedge \\ \qquad \forall b = b' \in \xi.\ a(b) = a'(b') \in \psi[b] \end{array}$$

2. *Define* $\boldsymbol{\Sigma} \in IFAM(\Xi) \rightarrow \Xi$ *by:*

$$\langle b,\ c\rangle = \langle b',\ c'\rangle \in \boldsymbol{\Sigma}(\xi, \psi)^1 \equiv b \xi b' \wedge c = c' \in \psi[b]$$

3. *Define* $\{\_|\_\} \in IFAM(\Xi) \rightarrow \Xi$ *by*

$$a = a' \in \{\xi \mid \psi\} \equiv a \xi a' \wedge \exists c.\ c \in \psi[a]$$

4. *Define* $+ \in \Xi \times \Xi \rightarrow \Xi$ *by*

$$a = a' \in \xi + \xi' \equiv \begin{array}{l} a \geq \mathrm{inl}\, b \wedge a' \geq \mathrm{inl}\, b' \wedge b \xi b' \ \vee \\ \qquad a \geq \mathrm{inr}\, c \wedge a' \geq \mathrm{inr}\, c' \wedge c \xi' c' \end{array}$$

5. *Define* $\boldsymbol{I} \in \mathcal{T} \times \mathcal{T} \times \Xi \rightarrow \Xi$ *by*

$$true = true \in \boldsymbol{I}(b, b', \xi) \equiv b \xi b'$$

□

For further convenience, we identify the following "independent" versions of $\boldsymbol{\Pi}$ and $\boldsymbol{\Sigma}$:

$$\begin{array}{rcl} \xi \rightarrow \xi' & \equiv & \boldsymbol{\Pi}(\xi, \boldsymbol{\lambda}[b].\xi') \\ \xi \times \xi' & \equiv & \boldsymbol{\Sigma}(\xi, \boldsymbol{\lambda}[b].\xi'). \end{array}$$

[1] For a more concise presentation, we leave some things implicit: since $\boldsymbol{\Sigma}(\xi, \psi)$ is to be in Ξ, we realize it must respect evaluation. In full, this function would be defined as

$$a = a' \in \boldsymbol{\Sigma}(\xi, \psi) \equiv \exists b, b', c, '.a \geq \langle b,\ c\rangle \wedge a' \geq \langle b',\ c'\rangle \ldots$$

4.2 Type systems

Define the collection of type systems by $\mathcal{TS} \equiv IFAM(\Xi)$. The notation for a type system is $\langle \tau, [\![\_]\!] \rangle$, or simply σ. $IFAM(\Xi)$ is a CPO under ordering:

$$\langle \xi, \psi \rangle \sqsubseteq \langle \xi', \psi' \rangle \equiv \xi \subseteq \xi' \wedge \forall A \in \xi.\ \psi([A]) = \psi'([A]),$$

with least element $\langle \emptyset, \emptyset \rangle$ and the least upper bound of a chain being:

$$\bigsqcup \{\langle \xi_\alpha, \psi_\alpha \rangle \mid \alpha \in I\} \equiv \langle \bigcup_{\alpha \in I} \xi_\alpha, \bigcup_{\alpha \in I} \psi_\alpha \rangle.$$

How does Ξ capture the meaning of a type? Externally, we can view a type as a binary relation on closed terms. The minimum salient characteristics of this relation are embodied in elements of Ξ: it must relate only normalizing terms, be symmetric and transitive, and respect evaluation. In such a construction as we are about to develop, the nearly arbitrary collection of ground relations Ξ is ultimately motivated and justified in retrospect: it is an appropriate collection to prove the desired properties. Now, consider the meaning of an intensional type system. Externally, we can view it as consisting of two parts, a type equality relation, and for each class of equal types, a membership relation. As just discussed, all these relations can be found in Ξ, and what we have just described is an arbitrary element of $\mathcal{TS}$. (From this external viewpoint, that the type system is "intensional" is only reflected in the fact that the type equality relation is not necessarily extensional.) Finally, consider the ordering $\sqsubseteq$ on type systems. It is meant to capture the notion of "extending" a type system: if $\langle \tau, [\![\_]\!] \rangle \sqsubseteq \langle \tau', [\![\_]\!]' \rangle$, then types equated by τ are equated by τ' and they retain the same membership relation.

EqFam is a predicate asserting, for a given type system, that the arguments are associated with equal families of types, as verified in lemma 4.1.

Definition 4.2 *For* $\langle \tau, [\![\_]\!] \rangle \equiv \sigma \in \mathcal{TS}$, *and* $B, B', C, C' \in \mathcal{T}$, *define*

$$EqFam(\sigma, B, B', C, C') \equiv B \tau B' \wedge \forall b = b' \in [\![B]\!].\ C(b) \tau C'(b'),$$

and set

$$Fam(\sigma, B, C) \equiv EqFam(\sigma, B, B, C, C)$$

□

Lemma 4.1

1. *EqFam and Fam are monotone in* $\mathcal{TS}$.
2. $\langle [\![B]\!],\ \boldsymbol{\lambda}[b].[\![C(b)]\!]\rangle$ *and* $\langle [\![B']\!],\ \boldsymbol{\lambda}[b].[\![C'(b)]\!]\rangle$ *are equal and an element of* $IFAM(\Xi)$, *if* $EqFam(\langle \tau,\ [\![\_]\!]\rangle, B, B', C, C')$.

4.2.1 Constructing type systems

We are at the central construction of this chapter, the type systems σ_α. One way of viewing the definition of σ_α is as a formalization of the notion of closing a set of atomic types under type constructors $+$, Σ, Π, $\{\_ \mid \_\}$ and I. These σ_α are thought of as being constructed iteratively, since σ_i is the meaning given to U_i in the definition of $\mathcal{F}_\alpha$, for $\alpha > i$. It is this predicative property of the construction, type equality in universe U_i (i.e., τ_i) not being available in the definition of $\mathcal{F}_i$, that we must overcome in order to define inductive types, as will be borne out in the next chapter.

Definition 4.3 *Fix* $0 < \alpha \leq \omega$, *and by induction assume* $\sigma_j \equiv \langle \tau_j,\ [\![\_]\!]_j\rangle \in \mathcal{TS}$ *for* $0 < j < \alpha$ *are defined. Define* $\mathcal{F}_\alpha \in \mathcal{TS} \xrightarrow{\text{mon}} \mathcal{TS}$ *by:*

$$\mathcal{F}_\alpha(\langle \tau,\ [\![\_]\!]\rangle) \equiv \langle \tau',\ [\![\_]\!]'\rangle,$$

where for a given $\sigma \equiv \langle \tau,\ [\![\_]\!]\rangle \in \mathcal{TS}$, *we define* $A = A' \in \tau'$ *iff one of the following holds; and if so, define* $[\![A]\!]'$ *by the corresponding* $\bullet$ *clause.*

1. $A \geq \mathit{void} \wedge A' \geq \mathit{void}$
   - $[\![\mathit{void}]\!]' \equiv \emptyset$
2. $\exists j < \alpha.\ A \geq U_j \wedge A' \geq U_j$
   - $[\![U_j]\!]' \equiv \tau_j$
3. (a) $A \geq B + C \wedge A' \geq B' + C'$
   (b) $B \tau B' \wedge C \tau C'$
   - $[\![B+C]\!]' \equiv [\![B]\!] + [\![C]\!]$
4. (a) $A \geq (\Sigma x\!:\!B)C \wedge A' \geq (\Sigma x\!:\!B')C'$
   (b) $EqFam(\sigma, B, B', \lambda x.C, \lambda x.C')$

- $[\![(\Sigma x:B)C]\!]' \equiv \boldsymbol{\Sigma}([\![B]\!], \boldsymbol{\lambda}[b].[\![C[b/x]]\!])$

5. *(a)* $A \geq (\Pi x:B)C \wedge A' \geq (\Pi x:B')C'$
   (b) $EqFam(\sigma, B, B', \lambda x.C, \lambda x.C')$
   - $[\![(\Pi x:B)C]\!]' \equiv \boldsymbol{\Pi}([\![B]\!], \boldsymbol{\lambda}[b].[\![C[b/x]]\!])$

6. *(a)* $A \geq \{x:B|\ C\} \wedge A' \geq \{x:B'|\ C'\}$
   (b) $B\tau B' \wedge Fam(\sigma, B, \lambda x.C) \wedge Fam(\sigma, B', \lambda x.C')$
   (c) $\forall b \in [\![B]\!].\ \exists c \in [\![C[b/x]]\!] \Leftrightarrow \exists c' \in [\![C'[b/x]]\!]$
   - $[\![\{x:B|\ C\}]\!]' \equiv \{[\![B]\!] \mid \boldsymbol{\lambda}[b].[\![C[b/x]]\!]\}$

7. *(a)* $A \geq I(b, c, B) \wedge A' \geq I(b', c', B')$
   (b) $B\tau B' \wedge b = b' \in [\![B]\!] \wedge c = c' \in [\![B]\!]$
   - $[\![I(b, c, B)]\!]' \equiv \boldsymbol{I}(b, c, [\![B]\!])$

Let $\sigma_\alpha \equiv \langle \tau_\alpha,\ [\![\_]\!]_\alpha \rangle \equiv$ *the least fixed point of* $\mathcal{F}_\alpha$.

□

Well-formedness Proof. The verification that $\mathcal{F}_\alpha$ is a monotonic operation on $\mathcal{TS}$ is routine, following from definitions 4.1 and 4.2 and lemma 4.1. As an example, we present the slightly unusual case for $\{\_ \mid \_\}$.

Fix $\sigma_p \equiv \langle \tau_p,\ [\![\_]\!]_p \rangle \sqsubseteq \sigma_q \equiv \langle \tau_q,\ [\![\_]\!]_q \rangle$, and write $\sigma'_p \equiv \mathcal{F}_\alpha(\sigma_p)$ and $\sigma'_q \equiv \mathcal{F}_\alpha(\sigma_q)$. Assume $\{x:B|\ C\} = \{x:B'|\ C'\} \in \tau'_p$. First, one should check the well-definedness of $[\![\{x:B|\ C\}]\!]'_p$, but that is not difficult to see: by lemma 4.1, $\langle [\![B]\!]_p,\ \boldsymbol{\lambda}[b].[\![C[b/x]]\!]_p \rangle \in IFAM(\Xi)$, and condition 7(c) implies

$$\{[\![B]\!]_p |\ \boldsymbol{\lambda}[b].[\![C[b/x]]\!]_p\} = \{[\![B']\!]_p |\ \boldsymbol{\lambda}[b].[\![C'[b/x]]\!]_p\}.$$

As for the monotonicity of $\mathcal{F}_\alpha$, since $\sigma_p \sqsubseteq \sigma_q$, $B\tau_q B'$ and by lemma 4.1, $Fam(\sigma_q, B, \lambda x.C)$ and $Fam(\sigma_q, B', \lambda x.C')$, thus $\{x:B|\ C\} = \{x:B'|\ C'\} \in \tau'_q$. All that is left to verify is that τ'_q associates the same membership relation with the type. But by $\sigma_p \sqsubseteq \sigma_q$, it is easy to see that:

$$\langle [\![B]\!]_p,\ \boldsymbol{\lambda}[b].[\![C[b/x]]\!]_p \rangle = \langle [\![B]\!]_q,\ \boldsymbol{\lambda}[b].[\![C[b/x]]\!]_q \rangle,$$

and so $[\![\{x:B|\ C\}]\!]'_p = [\![\{x:B|\ C\}]\!]'_q$.

□

4.2.2 Properties of σ_α

An important property of our construction is *cumulativity*: equal types in universe level i will be equal types in level $i+1$, and their membership relation will be the same. One nice feature of our construction is that cumulativity follows easily from the fact that the $\mathcal{F}_\alpha$'s form a chain in $\mathcal{TS} \stackrel{\text{mon}}{\to} \mathcal{TS}$, as pointed out in the next lemma.

Lemma 4.2 *For $\alpha < \beta \leq \omega$, $\mathcal{F}_\alpha \sqsubseteq \mathcal{F}_\beta$, and so $\sigma_\alpha \sqsubseteq \sigma_\beta$.*

Proof. The only difference between $\mathcal{F}_\alpha$ and $\mathcal{F}_\beta$ is the inclusion of atomic types U_γ for $\alpha \leq \gamma < \beta$ in clause *2*, so it is easy to see that $\mathcal{F}_\alpha \sqsubseteq \mathcal{F}_\beta$. Since the operation of taking a least fixed point is itself monotonic, $\sigma_\alpha \sqsubseteq \sigma_\beta$.
□

Now we can define four familiar relations to express type equality and membership.

Definition 4.4

$$\begin{array}{rcl} A = B & \equiv & A \tau_\omega B \\ A \ \mathit{Type} & \equiv & A = A \\ a = b \in C & \equiv & C \in \tau_\omega \wedge a = b \in [\![C]\!]_\omega \\ a \in C & \equiv & a = a \in C \end{array}$$

□

Note $A = B \in U_i \Leftrightarrow A = B \in \tau_i$.

The following four lemmas list all the facts that will be used in showing the soundness of the proof rules. All of these facts are trivial to establish because the arduous work has been done in definition 4.3. Lemma 4.3 follows from definitions 4.3 and 4.4, except for the final two rules, the so-called cumulativity rules, which follow from lemma 4.2. Lemmas 4.4, 4.5 and 4.6 can also be read off of definitions 4.3 and 4.4.

Lemma 4.3

$$a \in A \Leftrightarrow a = a \in A \qquad\qquad A \ \mathit{Type} \Leftrightarrow A = A$$

$$\frac{a = b \in A}{A \ \mathit{Type}}$$

$$\frac{a = b \in A}{b = a \in A} \qquad\qquad \frac{A = B}{B = A}$$

$$\frac{a = b \in A \quad b = c \in A}{a = c \in A} \qquad\qquad \frac{A = B \quad B = C}{A = C}$$

$$\frac{a \in A \quad A = B}{a \in B} \qquad\qquad \frac{a = b \in A \quad A = B}{a = b \in B}$$

$$\frac{a = b \in A}{a \downarrow} \qquad\qquad \frac{A \ \mathit{Type}}{A \downarrow}$$

$$\frac{a = b \in A \quad a \simeq c}{c = b \in A} \qquad\qquad \frac{A \ \mathit{Type} \quad A \simeq B}{A = B}$$

$$\frac{A = B \in U_i}{A = B} \qquad\qquad \frac{A = B \in U_i}{A = B \in U_{i+1}}$$

Lemma 4.4

1. $\mathit{void} = \mathit{void}$
2. $U_j = U_j$
3. $B + C = B' + C' \Leftrightarrow B = B' \wedge C = C'$
4. $(\Sigma x\!:\!B)C = (\Sigma x\!:\!B')C' \Leftrightarrow B = B' \wedge \forall b = b' \in B.\ C[b/x] = C'[b'/x]$
5. $(\Pi x\!:\!B)C = (\Pi x\!:\!B')C' \Leftrightarrow B = B' \wedge \forall b = b' \in B.\ C[b/x] = C'[b'/x]$
6. $\{x\!:\!B|\ C\} = \{x\!:\!B'|\ C'\} \Leftrightarrow B = B' \wedge \forall b = b' \in B.\ C[b/x] = C[b'/x] \wedge C'[b/x] = C'[b'/x] \wedge \exists c \in C[b/x] \Leftrightarrow \exists c' \in C'[b/x]$
7. $I(b, c, B) = I(b', c', B') \Leftrightarrow B = B' \wedge b = b' \in B \wedge c = c' \in B$

Lemma 4.5

1. $\mathit{void} = \mathit{void} \in U_i$
2. $U_j = U_j \in U_i \Leftrightarrow j < i$
3. $B + C = B' + C' \in U_i \Leftrightarrow B = B' \in U_i \wedge C = C' \in U_i$
4. $(\Sigma x\!:\!B)C = (\Sigma x\!:\!B')C' \in U_i \Leftrightarrow B = B' \in U_i \wedge \forall b = b' \in B.\ C[b/x] = C'[b'/x] \in U_i$
5. $(\Pi x\!:\!B)C = (\Pi x\!:\!B')C' \in U_i \Leftrightarrow B = B' \in U_i \wedge \forall b = b' \in B.\ C[b/x] = C'[b'/x] \in U_i$
6. $\{x\!:\!B|\ C\} = \{x\!:\!B'|\ C'\} \in U_i \Leftrightarrow B = B' \in U_i \wedge \forall b = b' \in B.\ C[b/x] = C[b'/x] \in U_i \wedge C'[b/x] = C'[b'/x] \in U_i \wedge \exists c \in C[b/x] \Leftrightarrow \exists c' \in C'[b/x]$
7. $I(b, c, B) = I(b', c', B') \in U_i \Leftrightarrow B = B' \in U_i \wedge b = b' \in B \wedge c = c' \in B$

Lemma 4.6

1. $\neg(a = a' \in \mathit{void})$
2. $a = a' \in B + C \Leftrightarrow B + C\ \mathit{Type} \wedge a \geq \mathit{inl}(b) \wedge a' \geq \mathit{inl}(b') \wedge b = b' \in B \vee a \geq \mathit{inr}(c) \wedge a' \geq \mathit{inr}(c') \wedge c = c' \in C$
3. $a = a' \in (\Sigma x\!:\!B)C \Leftrightarrow (\Sigma x\!:\!B)C\ \mathit{Type} \wedge a \geq \langle b,\ c\rangle \wedge a' \geq \langle b',\ c'\rangle \wedge b = b \in B \wedge c = c' \in C[b/x]$
4. $a = a' \in (\Pi x\!:\!B)C \Leftrightarrow (\Pi x\!:\!B)C\ \mathit{Type} \wedge a\downarrow \wedge a'\downarrow \wedge \forall b = b \in B.\ a(b) = a'(b) \in C[b/x]$
5. $a = a' \in \{x\!:\!B|\ C\} \Leftrightarrow \{x\!:\!B|\ C\}\ \mathit{Type} \wedge a = a' \in B \wedge \exists c \in C[a/x]$
6. $a = a' \in I(b, c, B) \Leftrightarrow a \geq \mathit{true} \wedge a' \geq \mathit{true} \wedge b = c \in B$

4.3 Truth and soundness

Now that we have built σ_α and isolated the salient properties of it in the previous four lemmas, we can define truth for sequents and show the proof rules are sound.

First, we introduce some abbreviations. For $k \in \{1,2\}$, write $\boldsymbol{a}_k$ for $a_{k,1}, \ldots, a_{k,n-1}$, and for a given open term c, write $[c]_k$ for $c[\boldsymbol{a}_k/x_1, \ldots x_{n-1}]$. Write Γ for the context $x_1 : A_1, \ldots, x_{n-1} : A_{n-1}$.

We now define what it means for two vectors of terms to be equal in a context, and use that in defining truth for sequents. The reader familiar with the NUPRL logic will notice that we have chosen a weaker, and simpler, notion of context well-formedness than the one given in [12], but this weakening is not exploited in the proofs of soundness.

Definition 4.5

1. $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$ *is defined by induction on* Γ. *The empty case is true; otherwise define* $\boldsymbol{a}_1, a_{1,n} = \boldsymbol{a}_2, a_{2,n} \in \Gamma, x_n : A_n$ *by:*

$$\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma \ \wedge \ [A_n]_1 = [A_n]_2 \ \wedge \ a_{1,n} = a_{2,n} \in [A_n]_1.$$

2. $\Gamma \models b = b' \in B$ *is defined by:*

$$\forall \boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma.\ [B]_1 = [B]_2 \wedge [b]_1 = [b']_2 \in [B]_1.$$

□

The next step is to verify the soundness of the proof rules, which is a straightforward exercise, working from lemmas 4.3–4.6. As an example of this verification, we present the soundness proofs for the Π type proof rules. But before we do that, we separate out the following, often used fact.

Lemma 4.7 $\boldsymbol{a}_1, b = \boldsymbol{a}_2, b' \in \Gamma, x : B$ *is implied by*

1. $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$
2. $\Gamma \models B = B' \in U_j$
3. $b = b' \in [B]_1$

4.3.1 Soundness of the Π type rules

Lemma 4.8 $\Gamma \models (\Pi x{:}B)C = (\Pi x{:}B')C' \in U_j$ *is implied by*

1. $\Gamma \models B = B' \in U_j$
2. $\Gamma,\ x{:}B \models C = C' \in U_j$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove:

3. $[(\Pi x{:}B)C]_1 = [(\Pi x{:}B')C']_2 \in U_j$.

By *1*,

4. $[B]_1 = [B']_2 \in U_j$.

Fix $b = b' \in [B]_1$; lemma 4.7 and *2* imply

5. $[C]_1[b/x] = [C']_2[b'/x] \in U_j$.

By lemma 4.5, 4 and 5 imply 3.
□

Lemma 4.9 $\Gamma \models \lambda x.c = \lambda x.c' \in (\Pi x{:}B)C$ *is implied by*

1. $\Gamma \models B \in U_j$
2. $\Gamma,\ x{:}B \models c = c' \in C$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove:

3. $[(\Pi x{:}B)C]_1 = [(\Pi x{:}B')C']_2$, and
4. $\lambda x.[c]_1 = \lambda x.[c']_2 \in [(\Pi x{:}B)C]_1$.

By *1*,

5. $[B]_1 = [B]_2 \in U_j$.

Fix $b = b' \in [B]_1$; lemma 4.7 and *2* imply

6. $[C]_1[b/x] = [C]_2[b'/x]$, and
7. $[c]_1[b/x] = [c']_2[b/x] \in [C]_1[b/x]$.

By cumulativity and lemma 4.4, 5 and 6 imply 3, and by the lemma 4.6, 7 implies 4.
□

Lemma 4.10 $\Gamma \models d(b) = d'(b') \in C[b/x]$ *is implied by*

1. $\Gamma \models d = d' \in (\Pi x\!:\!B)C$
2. $\Gamma \models b = b' \in B$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove

3. $[C]_1[[b]_1/x] = [C]_2[[b]_2/x]$, and
4. $[d]_1([b]_1) = [d']_2([b']_2) \in [C]_1[[b]_1/x]$.

By *2*,

5. $[b]_1 = [b']_2 \in [B]_1$,

and by *1*,

6. $[(\Pi x\!:\!B)C]_1 = [(\Pi x\!:\!B)C]_2$, and
7. $[d]_1 = [d']_2 \in [(\Pi x\!:\!B)C]_1$.

By lemma 4.4, 5 and 6 imply 3, and by lemma 4.6, 5 and 7 imply 4.
□

Lemma 4.11 $\Gamma \models d = d' \in (\Pi x\!:\!B)C$ *is implied by*

1. $\Gamma \models B \in U_j$
2. $\Gamma,\ x\!:\!B \models d(x) = d'(x) \in C$
3. $\Gamma \models d \in D$
4. $\Gamma \models d' \in D'$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove

5. $[(\Pi x\!:\!B)C]_1 = [(\Pi x\!:\!B)C]_2$, and
6. $[d]_1 = [d']_2 \in [(\Pi x\!:\!B)C]_1$.

By *1*,

7. $[B]_1 = [B]_2 \in U_j$.

Fix $b = b' \in [B]_1$; lemma 4.7 and *2* imply

8. $[C]_1[b/x] = [C]_2[b'/x]$, and

9. $[d]_1(b) = [d]_2(b') \in [C]_1[b/x]$.

By cumulativity, 7 and 8 imply 5. *3* and *4* imply $[d]_1 \in \mathcal{V}$ and $[d']_2 \in \mathcal{V}$, and so 9 implies 6 holds.
□

4.4 Concluding consistency

By an induction on derivations, if $\Gamma \vdash a = b \in A$ is derivable, then $\Gamma \models a = b \in A$, and in the case of an empty context that implies $a = b \in A$. What are some consequences of this? We can conclude all the facts listed in lemmas 4.3–4.6, including $\neg(a \in \mathit{void})$ for any term a, so we have shown the intuitionistic consistency of the theory.

Summary

We have given a semantics for the basic type theory and used it to prove the soundness of the proof rules. The intuitionistic consistency of that theory was a corollary of soundness. We are now ready to extend the semantics of this chapter to account for inductive types.

Chapter 5

Semantic Account of Inductive Types

In this chapter we take the semantic account of the basic theory given in chapter 4 and relativize it, by Girard's method, to account for inductive types. As in the previous chapter, intuitionistic consistency is an immediate consequence of this account.

The construction used in chapter 4 will not suffice here because it can not justify the impredicativity introduced into the definition of types by inductive types. Consider the introduction rule:

$$\Gamma \vdash (\mu x{:}U_j)B \in U_j$$
1. $\Gamma,\ x{:}U_j \vdash B \in U_j$
2. $\Gamma,\ x{:}U_j,\ y{:}U_j,\ x \subseteq y \vdash t \in B \subseteq B[y/x]$.

Subgoal 1 is stating that, in order to admit the μ type into universe U_j, $\lambda x.B$ must map equal types in U_j to equal types in U_j. This is an impredicative definition of U_j and can not be expressed by the construction used in chapter 4. The solution is to relativize this construction in the manner of Girard's relatization of Tait's computability method [40,22,21]. The framework we build here can also be used to justify a version of the Δ type constructor, as we will demonstrate in 5.7.

The chapter is organized along the same lines as chapter 4, as we repeat the developments of that chapter with the addition of types $(\mu x{:}U_j)B$ and $(\nu x{:}U_j)B$. We may outline the chapter as follows.

1. Inductive types introduce impredicativity which is "broken" using a technique similar to the one used in chapter 3. In section 1, the two major differences in the way we apply Girard's method in this chapter are detailed. First, we need not a single collection of ground sets, but a series of collections $\Xi_1, \Xi_2, \ldots$. Second, it desirable to abandon the "environment style" approach of assigning meaning to open type expressions, and to consider only closed type expressions.

2. We extend the language through a series of collections of constants $\mathcal{C}_1, \mathcal{C}_2, \ldots$ called the *ground types*, in such a way that each relation in Ξ_i is represented by a constant in $\mathcal{C}_i$. Given this, inductive types can be described without the impredicative quantification over all types of the universe level in question.

3. Also in section 1, new semantic operations corresponding to the $\subseteq$, μ and ν type constructors are defined.

4. In section 2, type systems are defined as in chapter 4. Again, we define monotonic operations on type systems, named $\mathcal{G}_\alpha$ here, whose least fixed points model type universes.

5. In a key lemma of this chapter, 5.8, we prove a stratified version of the substitution lemma, which allows us to conclude some basic properties about inductive types.

6. In section 3, truth is defined for sequents and the proof rules for the $\subseteq$, μ and ν types are proven sound. Thus is section 4 we can conclude the intuitionistic consistency of the theory.

7. In section 5 we show the soundness of introduction rules that use the syntactic property of strong positivity instead of explicitly requiring monotonicity.

8. Up until this point, the construction has been for the simple inductive types, yet it can readily be altered to justify the parameterized inductive types. In section 6 we outline these alterations.

9. Finally, in section 7, we show how the semantics of this chapter can justify a version of the Δ type constructor of chapter 3.

5.1 Ground relations and ground types

For this section, we will need a specialized notion of partial equivalence relations over terms, parameterized by a set of constants and with a distinguished constant Ω. So, assume we have fixed the term constructors and rules of reduction for the basic type theory plus simple inductive types. Let Ω be a new constant, intended to represent any arbitrary normalizable term, and suppose we are given a set S of fresh constants, intended to represent atomic types. Terms and closed terms in this setting are as before, with the possibility of occurrences of Ω or elements of S in them. It is also obvious how one should extend evaluation: Ω and each constant in S has itself as its value. We write $\geq$ for the evaluation relation, assuming S is determinable from the context.

Define PER(S) to be the set of partial equivalence relations ξ over normalizing terms (with possible occurrences of Ω and elements of S) that respect $\geq$ and *ignore* Ω, that is to say, $a[\Omega/x]\xi a[b/x]$ for any $a[\Omega/x] \in Fi(\xi)$ and $b \in \mathcal{V}$. The intuition behind ignoring Ω is that since Ω is representing arbitrary, unspecified, normalizing terms, any instance of it can be replaced by such a term and the result will be an equal term.

We continue with a series of definitions that culminate in the definition of the ultimate set of terms $\mathcal{T}$ and the series of collections of ground relations.

- Let $\mathcal{C}_0 \equiv \emptyset$, let $\mathcal{T}_0$ be the set of closed terms with possible occurrences of Ω and let $\mathcal{V}_0$ be the normalizing terms in $\mathcal{T}_0$.

- For $i > 0$, define the sets $\mathcal{C}_i$, $\mathcal{T}_i$ and $\mathcal{V}_i$, by induction on i. Let $\mathcal{C}_{<i} \equiv \bigcup_{j<i} \mathcal{C}_j$, and let $\mathcal{C}_i$, the *ground types of level* i, be a set of new constant symbols, two for each element of PER($\mathcal{C}_{<i}$). Let X_i, Y_i and Z_i range over $\mathcal{C}_i$. Let $\mathcal{T}_i$ be the set of closed terms constructed with possible occurrences of Ω and the constants in $\mathcal{C}_{<i} \cup \mathcal{C}_i$. Finally, let $\mathcal{V}_i$ be the normalizing terms in $\mathcal{T}_i$.

- Let $\mathcal{C} \equiv \bigcup_j \mathcal{C}_j$, $\mathcal{T} \equiv \bigcup_j \mathcal{T}_j$ and $\mathcal{V} \equiv \bigcup_j \mathcal{V}_j$. Thus, $\mathcal{T}$ is the set of terms constructed with possible occurrences of constants in $\mathcal{C}$, and $\mathcal{V}$ are the normalizing terms in $\mathcal{T}$.

- For $i \geq 0$, define $|\_|_i \in \mathcal{T} \to \mathcal{T}_i$ (a kind of stripping operation) by

letting $|a|_i$ be the term constructed from a by replacing every instance of any ground type of level $i+1$, or higher, by Ω.

- Let $\Xi \equiv \mathrm{PER}(\mathcal{C})$. For $i > 0$, define the *ground relations of level i*, by:

$$\Xi_i \equiv \{\xi \in \Xi \mid \forall a \in Fi(\xi).\ a\xi|a|_{i-1}\}.$$

- For $i > 0$, define $\_|_i \in \Xi_i \to \mathrm{PER}(\mathcal{C}_{<i})$ to be restriction to $\mathcal{V}_{i-1} \times \mathcal{V}_{i-1}$.

The motivation behind these definitions is to take the idea of ground sets used in chapter 3 and generalize it to the situation we have in a type theory stratified by universes, where all terms are untyped, but types can be terms themselves. To attempt to identify a collection of ground relations on terms and then to add constants for each such relation seems doomed to vicious circularity problems, *unless* we recognize that we can identify a separate collection of ground relations for each universe level and have these relations *ignore* the constants of its own, and higher, levels.

We add *two* constants for each relation so that we can have intensionally distinct but extensionally equal ground types. This is important only for one base case in the proof of lemma 5.8.

The only other problem to overcome is that, at the moment of introduction of $\mathcal{C}_i$, the ultimate set of terms is not defined — after all, we are in the midst of defining it! So, we can only approximate Ξ_i by $\mathrm{PER}(\mathcal{C}_{<i})$. But this is is justifiable, as the next lemma demonstrates.

Lemma 5.1 *The functions* $\_|_i \in \Xi_i \to \mathrm{PER}(\mathcal{C}_{<i})$ *are bijections.*

Proof. Fix i and abbreviate $|a|_{i-1}$ by $|a|$. First, we note:

$$\forall \xi \in \Xi_i.\ \forall a, b \in \mathcal{T}.\ a\xi b \Leftrightarrow |a|(\xi|_i)|b| \tag{5.1}$$

Suppose $a\xi b$; then $a\xi|a|$ and $b\xi|b|$, and so $|a|\xi|b|$. Since $\xi|_i$ is the restriction of ξ to $\mathcal{V}_{i-1}$, $|a|(\xi|_i)|b|$. For the other direction, suppose for $a, b \in \mathcal{T}$, that $|a|(\xi|_i)|b|$; then $|a|\xi|b|$ and since ξ ignores Ω, $a\xi b$. The injectivity of $\_|_i$ follows easily from (5.1).

Now we show $\_|_i$ is surjective. Fix $\xi' \in \mathrm{PER}(\mathcal{C}_{<i})$; define $\xi \in \mathcal{V} \times \mathcal{V}$ by $a\xi b \equiv |a|\xi'|b|$. The symmetry and transitivity of ξ are obvious, so it suffices to show ξ ignores Ω and that $a \in Fi(\xi)$ implies $a\xi|a|$.

ξ ignores Ω: suppose $a \in Fi(\xi)$. It is convenient to write a as:

$$a_1[\Omega, \ldots, \Omega, X^1_{j_1}, \ldots, X^n_{j_n}/x_1, \ldots, x_m, y_1, \ldots, y_m],$$

where each x_k occurs exactly once in a_1, each $X^k_{j_k}$ is a ground type of level i, or higher, and no Ω or ground type of level i, or higher, occurs in a_1. Given this notation, $|a|$ is:

$$a_1[\Omega, \ldots, \Omega, \Omega, \ldots, \Omega/x_1, \ldots, x_m, y_1, \ldots, y_m].$$

Now to show ξ ignores Ω, it suffices to show $a\xi a_2$, where:

$$a_2 \equiv a_1[b_1, \ldots, b_m, X^1_{j_1}, \ldots, X^n_{j_n}/x_1, \ldots, x_m, y_1, \ldots, y_m],$$

and $b_1, \ldots, b_m \in \mathcal{V}$. Note $|a_2|$ is:

$$a_1[|b_1|, \ldots, |b_m|, \Omega, \ldots, \Omega/x_1, \ldots, x_m, y_1, \ldots, y_m].$$

By the definition of ξ, $|a| \in Fi(\xi')$, and since ξ' ignores Ω, $|a|\xi'|a_2|$. Conclude that $a\xi a_2$, and so ξ ignores Ω.

$a \in Fi(\xi) \Rightarrow a\xi|a|$: This is obvious, since $a\xi|a| \Leftrightarrow |a|\xi'||a||$ and $|\_|$ is idempotent. Thus ξ is in Ξ_i, and so we conclude $\_|_i$ is surjective.
□

$\mathcal{C}_i$ was defined so that two constants in it were associated with each element of $\mathrm{PER}(\mathcal{C}_{<i})$, so by lemma 5.1, we can transfer this association to Ξ_i: let $val_i \in \mathcal{C}_i \to \Xi_i$ be this association, and let $rep_i \in \Xi_i \to \mathcal{C}_i$ be the right inverse of val_i.

Lemma 5.2 *Each Ξ_i is a complete lattice under $\subseteq$.*

Proof. It is easy to see that Ξ_i is closed under arbitrary intersection. Notice also that the least upper bound of a chain is its union.
□

5.1.1 Operations on ground relations

We have redefined the sets of terms $\mathcal{T}$ and $\mathcal{V}$, and the collection of partial equivalence relations Ξ. However, the functions given in definition 4.1 are well-defined when reinterpreted in this context and they display the following properties with respect to Ξ_i.

Lemma 5.3 *When $\boldsymbol{\Pi}$, $\boldsymbol{\Sigma}$, or $\{\_ \mid \_\}$ is restricted to $IFAM(\Xi_i)$ — or $+$ to $\Xi_i \times \Xi_i$ — then its range is within Ξ_i. $\boldsymbol{I}$'s range is within Ξ_1.*

Proof. Straightforward. As an example, we present the case for $\boldsymbol{\Pi}$. Fix $\langle \xi,\ \psi \rangle \in IFAM(\Xi_i)$ and $a \in \boldsymbol{\Pi}(\xi,\psi)$. Let $|\_|$ abbreviate $|\_|_{i-1}$; Since $a \in \mathcal{V}$ implies $|a| \in \mathcal{V}$, we must show $a = |a| \in \boldsymbol{\Pi}(\xi,\psi)$. Fix $b = b' \in \xi$, it suffices to show $a(b) = |a|(b') \in \psi[b]$. By the definition of $\boldsymbol{\Pi}$, $a(b) = a(b') \in \psi[b]$; since $\psi[b] \in \Xi_i$, $a(b') = |a(b)| \in \psi[b]$; factoring the stripping operation, $|a(b')|$ is the term $|a|(|b'|)$; and since $\psi[b]$ ignores Ω, $|a|(|b'|) = |a|(b') \in \psi[b]$. Stringing the equalities together, we conclude $a(b) = |a|(b') \in \psi[b]$.
□

Definition 5.1

1. *Define* $\subseteq \in \Xi \times \Xi \to \Xi_1$ *by:*

$$true = true \in \xi \subseteq \xi' \equiv \xi \subseteq \xi'.$$

2. *For a given* $j > 0$, *define* $\boldsymbol{\mu}_j \in (\Xi_j \overset{\text{mon}}{\to} \Xi_j) \to \Xi_j$ *by:*

$$\boldsymbol{\mu}_j(\phi) \equiv \text{ the least fixed point of } \phi.$$

3. *For a given* $j > 0$, *define* $\boldsymbol{\nu}_j \in (\Xi_j \overset{\text{mon}}{\to} \Xi_j) \to \Xi_j$ *by: letting* $\boldsymbol{\nu}_j(\phi)$ *be the greatest fixed point* $\xi \in \Xi_j$ *of*

$$a = a' \in \xi \Leftrightarrow out(a) = out(a') \in \phi(\xi).$$

□

5.2 Type systems

As in chapter 4, let the collection of type systems $\mathcal{TS}$ be $IFAM(\Xi)$. Note that the definitions of *EqFam* and *Fam* given in definition 4.2 are sensible in this context, and that lemma 4.1 holds here, by the same argument.

5.2.1 Constructing type systems

We repeat the construction of $\mathcal{F}_\alpha$ given in definition 4.3, with the addition of extra clauses, for the $\subseteq$, μ and ν types. In particular, we need separate clauses (11 and 13) for the μ and ν types *just* to guarantee the cumulativity of type universes.

Definition 5.2 *Fix* $0 < \alpha \leq \omega$, *and by induction assume* $\sigma_j \equiv \langle \tau_j, [\![\_]\!]_j \rangle \in \mathcal{TS}$ *for* $0 < j < \alpha$ *are defined. Define* $\mathcal{G}_\alpha \in \mathcal{TS} \stackrel{\text{mon}}{\to} \mathcal{TS}$ *by:*

$$\mathcal{G}_\alpha(\langle \tau, [\![\_]\!] \rangle) \equiv \langle \tau', [\![\_]\!]' \rangle,$$

where for a given $\sigma \equiv \langle \tau, [\![\_]\!] \rangle \in \mathcal{TS}$, *we define* $A = A' \in \tau'$ *iff one of the following holds; and if so, define* $[\![A]\!]'$ *by the corresponding* $\bullet$ *clause.*

1. $A \geq \mathit{void} \wedge A' \geq \text{void}$
   - $[\![\mathit{void}]\!]' \equiv \emptyset$
2. $\exists j < \alpha.\ A \geq U_j \wedge A' \geq U_j$
   - $[\![U_j]\!]' \equiv \tau_j$
3. $\exists j \leq \alpha.\ A \geq X_j \wedge A' \geq X_j$
   - $[\![X_j]\!]' \equiv \mathit{val}_j(X_j)$
4. *(a)* $A \geq B + C \wedge A' \geq B' + C'$
   (b) $B \tau B' \wedge C \tau C'$
   - $[\![B + C]\!]' \equiv [\![B]\!] + [\![C]\!]$
5. *(a)* $A \geq B \subseteq C \wedge A' \geq B' \subseteq C'$
   (b) $B \tau B' \wedge C \tau C'$
   - $[\![B \subseteq C]\!]' \equiv [\![B]\!] \subseteq [\![C]\!]$
6. *(a)* $A \geq (\Sigma x\!:\!B)C \wedge A' \geq (\Sigma x\!:\!B')C'$
   (b) $\mathit{EqFam}(\sigma, B, B', \lambda x.C, \lambda x.C')$
   - $[\![(\Sigma x\!:\!B)C]\!]' \equiv \boldsymbol{\Sigma}([\![B]\!], \boldsymbol{\lambda}[b].[\![C[b/x]]\!])$
7. *(a)* $A \geq (\Pi x\!:\!B)C \wedge A' \geq (\Pi x\!:\!B')C'$
   (b) $\mathit{EqFam}(\sigma, B, B', \lambda x.C, \lambda x.C')$
   - $[\![(\Pi x\!:\!B)C]\!]' \equiv \boldsymbol{\Pi}([\![B]\!], \boldsymbol{\lambda}[b].[\![C[b/x]]\!])$
8. *(a)* $A \geq \{x\!:\!B |\ C\} \wedge A' \geq \{x\!:\!B' |\ C'\}$
   (b) $B \tau B' \wedge \mathit{Fam}(\sigma, B, \lambda x.C) \wedge \mathit{Fam}(\sigma, B', \lambda x.C')$
   (c) $\forall b \in [\![B]\!].\ \exists c \in [\![C[b/x]]\!] \Leftrightarrow \exists c' \in [\![C'[b/x]]\!]$

- $[\![\{x{:}B|\ C\}]\!]' \equiv \{[\![B]\!] \mid \boldsymbol{\lambda}[b].[\![C[b/x]]\!]\}$

9. *(a)* $A \geq I(b,c,B) \wedge A' \geq I(b',c',B')$
   (b) $B \tau B' \wedge b = b' \in [\![B]\!] \wedge c = c' \in [\![B]\!]$
   - $[\![I(b,c,B)]\!]' \equiv \boldsymbol{I}(b,c,[\![B]\!])$

10. *(a)* $\alpha < \omega \wedge A \geq (\mu x{:}U_\alpha)B \wedge A' \geq (\mu x{:}U_\alpha)B'$
    (b) $\forall X_\alpha \in \mathcal{C}_\alpha.\ B[X_\alpha/x] = B'[X_\alpha/x] \in \tau.$
    (c) $\phi \equiv \boldsymbol{\lambda}\xi.\ [\![B[rep_\alpha(\xi)/x]]\!] \in \Xi_\alpha \overset{\text{mon}}{\rightarrow} \Xi_\alpha$
    - $[\![(\mu x{:}U_\alpha)B]\!]' \equiv \boldsymbol{\mu}_\alpha(\phi)$

11. *(a)* $\exists j < \alpha.\ A \geq (\mu x{:}U_j)B \wedge A' \geq (\mu x{:}U_j)B'$
    (b) $(\mu x{:}U_j)B = (\mu x{:}U_j)B' \in \tau_j$
    - $[\![(\mu x{:}U_j)B]\!]' \equiv [\![(\mu x{:}U_j)B]\!]_j$

12. *(a)* $\alpha < \omega \wedge A \geq (\nu x{:}U_\alpha)B \wedge A' \geq (\nu x{:}U_\alpha)B'$
    (b) $\forall X_\alpha \in \mathcal{C}_\alpha.\ B[X_\alpha/x] = B'[X_\alpha/x] \in \tau.$
    (c) $\phi \equiv \boldsymbol{\lambda}\xi.\ [\![B[rep_\alpha(\xi)/x]]\!] \in \Xi_\alpha \overset{\text{mon}}{\rightarrow} \Xi_\alpha$
    - $[\![(\nu x{:}U_\alpha)B]\!]' \equiv \boldsymbol{\nu}_\alpha(\phi)$

13. *(a)* $\exists j < \alpha.\ A \geq (\nu x{:}U_j)B \wedge A' \geq (\nu x{:}U_j)B'$
    (b) $(\nu x{:}U_j)B = (\nu x{:}U_j)B' \in \tau_j$
    - $[\![(\nu x{:}U_j)B]\!]' \equiv [\![(\nu x{:}U_j)B]\!]_j$

Let $\sigma_\alpha \equiv \langle \tau_\alpha,\ [\![\_]\!]_\alpha \rangle \equiv$ *the least fixed point of* $\mathcal{G}_\alpha$.

□

Well-formedness Proof. As in definition 4.3, it is a routine matter to verify the well-formedness of $\mathcal{G}_\alpha$, and the new clauses present no further difficulties.

□

5.2.2 Properties of σ_α

In the previous definition we used the well-known fact that monotonic operations on complete partial orders have least fixed points, which can be found by iterating the operation to an ordinal of greater cardinality than the CPO. These iterates are useful for proving properties of the least fixed point, so we adopt the following notations for them.

Definition 5.3 *Define the iterates of $\mathcal{G}_\alpha$ as follows.*

$$\begin{array}{lcll} \sigma_\alpha^0 & \equiv & \langle \tau_\alpha^0,\ [\![\_]\!]_\alpha^0 \rangle & \equiv \ \bot_{\mathcal{TS}} = \langle \emptyset,\ \emptyset \rangle \\ \sigma_\alpha^{\gamma+1} & \equiv & \langle \tau_\alpha^{\gamma+1},\ [\![\_]\!]_\alpha^{\gamma+1} \rangle & \equiv \ \mathcal{G}_\alpha(\sigma_\alpha^\gamma) \\ \sigma_\alpha^\lambda & \equiv & \langle \tau_\alpha^\lambda,\ [\![\_]\!]_\alpha^\lambda \rangle & \equiv \ \bigsqcup_{\gamma<\lambda} \sigma_\alpha^\gamma, \quad \textit{for limit } \lambda \end{array}$$

Thus, $\sigma_\alpha = \bigsqcup_\gamma \sigma_\alpha^\gamma$.

□

Because of clauses 11 and 13, $\mathcal{G}_i \not\sqsubseteq \mathcal{G}_{i+1}$, as was the case in lemma 4.2. However, the γ-iterates form a chain, and cumulativity follows.

Lemma 5.4 *For $\alpha < \beta \leq \omega$, $\sigma_\alpha^\gamma \sqsubseteq \sigma_\beta^\gamma$, and so $\sigma_\alpha \sqsubseteq \sigma_\beta$.*

Proof. Fix $\alpha < \beta \leq \omega$. The cases for $\gamma = 0$ and limit γ are trivial, so we consider the successor case. Suppose $\sigma_\alpha^\gamma \sqsubseteq \sigma_\beta^\gamma$; The cases for the clauses 1 through 9 are as in the proof of the monotonicity of $\mathcal{G}_\alpha$. For clause 10 of $\mathcal{G}_\alpha$, suppose $(\mu x : U_\alpha)B = (\mu x : U_\alpha)B' \in \tau_\alpha^{\gamma+1}$. Let $\mu \equiv (\mu x : U_\alpha)B$, $\mu' \equiv (\mu x : U_\alpha)B'$ and $\xi \equiv [\![\mu]\!]_\alpha^{\gamma+1}$. Since $\sigma_\alpha^{\gamma+1} \sqsubseteq \sigma_\alpha$, $\mu = \mu' \in \tau_\alpha$ and $\xi = [\![\mu]\!]_\alpha$. By clause 11 of $\mathcal{G}_\beta$, $\mu = \mu' \in \tau_\beta^{\gamma+1}$ and $\xi = [\![\mu]\!]_\beta^{\gamma+1}$. Clause 12 follows by the same argument, and clauses 11 and 13 are immediate. From this fact we conclude $\sigma_\alpha \sqsubseteq \sigma_\beta$.

□

We define the four predicates $A = B$, A *Type*, $a = b \in A$ and $a \in A$ as in definition 4.4:

$$\begin{array}{rcl} A = B & \equiv & A \tau_\omega B \\ A \ \textit{Type} & \equiv & A = A \\ a = b \in C & \equiv & C \in \tau_\omega \wedge a = b \in [\![C]\!]_\omega \\ a \in C & \equiv & a = a \in C. \end{array}$$

As before, $A = B \in U_i \Leftrightarrow A = B \in \tau_i$. The facts about these predicates listed in lemmas 4.3–4.6 carry over to this chapter. As for the new type constructors, we can list the corresponding facts for $\subseteq$ types now because they follow easily from definition 5.2, but more work must be done to justify the facts we will assert for the μ and ν types.

Lemma 5.5

1.	$B \subseteq C = B' \subseteq C'$	$\Leftrightarrow$	$B = B' \wedge C = C'$
2.	$B \subseteq C = B' \subseteq C' \in U_i$	$\Leftrightarrow$	$B = B' \in U_i \wedge C = C' \in U_i$
3.	$a = a' \in B \subseteq C$	$\Leftrightarrow$	$B \subseteq C$ *Type* $\wedge\ a \geq$ *true* $\wedge\ a' \geq$ *true* $\forall b = b' \in B.\ b = b' \in C$

Definition 5.4 $B \trianglelefteq C \equiv \mathit{true} \in B \subseteq C$

□

The type and membership relations of σ_i fall into certain classes of ground relations, as delineated by the next lemma. As a consequence, for $A \in U_i$, there exists an extensionally equal constant in $\mathcal{C}_i$, and therefore our choice of collections of ground relations is justified.

Lemma 5.6 *For* $0 < i < \omega$, $\tau_i \in \Xi_{i+1} \wedge \forall A \in \tau_i.\ [\![A]\!]_i \in \Xi_i$.

Proof. Let $\mathcal{TS}_i \equiv \{\langle \tau,\ [\![\_]\!]\rangle \in \mathcal{TS} \mid \tau \in \Xi_{i+1} \wedge \forall A \in \tau.\ [\![A]\!] \in \Xi_i\}$. $\mathcal{TS}_i$ is a sub-CPO of $\mathcal{TS}$ and by lemma 5.3 and definition 5.1, $\sigma \in \mathcal{TS}_i \Rightarrow \mathcal{G}_i(\sigma) \in \mathcal{TS}_i$, so $\mathcal{G}_i$'s least fixed point, τ_i, is in $\mathcal{TS}_i$.
□

We define the notion of extensional equality on types.

Definition 5.5 *For A Type and B Type,*

$$A =_\mathcal{E} B \equiv [\![A]\!]_\omega = [\![B]\!]_\omega.$$

□

An important property of the type systems σ_i is a stratified version of the substitution property seen earlier in lemma 3.4. The proof for lemma 3.4 was entirely straightforward, but here it is complicated by dependent type constructors and intensional type equality. We proceed by temporarily extending evaluation to open terms, letting variables evaluate to themselves.

Let $\geq_O$ be this new relation. We use the following obvious fact about evaluation in the lemma 5.8, but note that it depends crucially on there being no "universe elimination" rule in evaluation, that is, on a type never being a principle argument in a redex.

Lemma 5.7 *For $D \in \mathcal{T}$, If $A \geq_O B$, and B is not y, then $A[D/y] \geq_O B[D/y]$; and if $A \geq_O y$ and $D \geq_O E$, then $A[D/y] \geq_O E$.*

Lemma 5.8 *$\forall i.\ \forall \gamma.\ \forall Y_i, Z_i \in \mathcal{C}_i$. where Y_i and Z_i are distinct, yet extensionally equal, $\forall A[Y_i/y], A'[Y_i/y], D_1, D_2 \in \mathcal{T}$. such that $D_1 = D_2 \in U_i$ and $D_1 =_\varepsilon Y_i$,*

$$A[Y_i/y] = A'[Y_i/y] \in \tau_i^\gamma \text{ and } A[Z_i/y] = A'[Z_i/y] \in \tau_i^\gamma \text{ implies}$$

$$A[D_1/y] = A'[D_2/y] \in U_i \quad \text{and} \quad A[D_1/y] =_\varepsilon A[Y_i/y] =_\varepsilon A[Z_i/y].$$

Proof. Fix i and induct on γ. Fix Y_i, Z_i, A, A', D_1 and D_2. For a given open term E, let $[E]_Y \equiv E[Y_i/y]$, $[E]_Z \equiv E[Z_i/y]$, $[E]_1 \equiv E[D_1/y]$ and $[E]_2 \equiv E[D_2/y]$. We consider each clause in the definition of $\mathcal{G}_i$ in turn.

1. Suppose $A \geq_O$ *void* and $A' \geq_O$ *void*, or $A \geq_O U_j$ and $A' \geq_O U_j$, for $j < i$. This case is trivial.

2. Suppose $A \geq_O X_j$ and $A' \geq_O X_j$, or $A \geq_O y$ and $A' \geq_O y$. Both cases are immediate, but note how cases like $A \geq_O X_i$ and $A' \geq_O y$ is ruled out by Y_i and Z_i being nonequal types. In fact, the sole purpose for having extensionally equal but intensionally distinct ground types is this case!

3. Suppose $A \geq_O (\Sigma x\!:\!B)C$ and $A' \geq_O (\Sigma x\!:\!B')C'$. By definition:

   (a) $[B]_Y = [B']_Y \in U_i$ (let $\xi \equiv [\![[B]_Y]\!]_i$)

   (b) $\forall b = b' \in \xi.\ [C]_Y[b/x] = [C']_Y[b'/x] \in U_i$

   (c) $[B]_Z = [B']_Z \in U_i$ (let $\xi' \equiv [\![[B]_Z]\!]_i$)

   (d) $\forall b = b' \in \xi'.\ [C]_Z[b/x] = [C']_Z[b'/x] \in U_i$.

   By induction:

   (e) $[B]_1 = [B']_2 \in U_i$

(f) $[B]_1 =_\mathcal{E} [B]_Y =_\mathcal{E} [B]_Z$

(g) $\forall b = b' \in \xi.\ [C]_1[b/x] = [C']_2[b'/x] \in U_i$

(h) $\forall b \in \xi.\ [C]_1[b/x] =_\mathcal{E} [C]_Y[b/x] =_\mathcal{E} [C]_Z[b/x]$.

From (e) and (g), we conclude $[A]_1 = [A']_2 \in U_i$, and from (f) and (h), $[A]_1 =_\mathcal{E} [A]_Y =_\mathcal{E} [A]_Z$.

The cases for the Π type is similar and those for $+$ and $\subseteq$ types are even simpler.

4. Suppose $A \geq \{x\!:\!B |\ C\}$ and $A' \geq \{x\!:\!B' |\ C'\}$. By definition:

(a) $[B]_Y = [B']_Y \in U_i$ (let $\xi \equiv [\![[B]_Y]\!]_i$)

(b) $\forall b = b' \in \xi.\ [C]_Y[b/x] = [C]_Y[b'/x] \in U_i$, and
$[C']_Y[b/x] = [C']_Y[b'/x] \in U_i$

(c) $\forall b \in \xi.\ \exists c \in [C]_Y[b/x] \Leftrightarrow \exists c \in [C']_Y[b/x]$

(d) $[B]_Z = [B']_Z \in U_i$ (let $\xi' \equiv [\![[B]_Z]\!]_i$)

(e) $\forall b = b' \in \xi'.\ [C]_Z[b/x] = [C]_Z[b'/x] \in U_i$, and
$[C']_Z[b/x] = [C']_Z[b'/x] \in U_i$

(f) $\forall b \in \xi'.\ \exists c \in [C]_Z[b/x] \Leftrightarrow \exists c \in [C']_Z[b/x]$.

By induction (and for (i) and (j), instantiating the inductive hypothesis with $D_1 = D_1 \in U_i$ and $D_2 = D_2 \in U_i$):

(g) $[B]_1 = [B']_2 \in U_i$

(h) $[B]_1 =_\mathcal{E} [B]_Y =_\mathcal{E} [B]_Z$

(i) $\forall b = b' \in \xi.\ [C]_1[b/x] = [C]_1[b'/x] \in U_i$, and
$[C']_2[b/x] = [C']_2[b'/x] \in U_i$

(j) $\forall b \in \xi.\ [C]_1[b/x] =_\mathcal{E} [C]_Y[b/x] =_\mathcal{E} [C]_Z[b/x]$, and
$[C']_2[b/x] =_\mathcal{E} [C']_Y[b/x] =_\mathcal{E} [C']_Z[b/x]$.

By (c) and (j):

(k) $\forall b \in \xi.\ \exists c \in [C]_1[b/x] \Leftrightarrow \exists c \in [C']_2[b/x]$.

Thus, by (g), (i) and (k), $[A]_1 = [A']_2 \in U_i$. By (c), (f) and (j), for $b \in \xi$, the inhabitation of $[C]_1[b/x]$, $[C]_Y[b/x]$ and $[C]_Z[b/x]$ are coincident, and therefore $[A]_1 =_\mathcal{E} [A]_Y =_\mathcal{E} [A]_Z$.

5. Suppose $A \geq_O I(b, c, B)$ and $A' \geq_O I(b', c', B')$. By definition:

   (a) $[B]_Y = [B']_Y \in U_i$ (let $\xi \equiv [\![[B]_Y]\!]_i$)

   (b) $[b]_Y = [b']_Y \in \xi \wedge [c]_Y = [c']_Y \in \xi$

   (c) $[B]_Z = [B']_Z \in U_i$ (let $\xi' \equiv [\![[B]_Z]\!]_i$)

   (d) $[b]_Z = [b']_Z \in \xi' \wedge [c]_Z = [c']_Z \in \xi'$.

   By induction and the fact that $\xi \in \Xi_i$ (a crucial use of ground relations of level i ignoring Y_i and Z_i):

   (e) $[B]_1 = [B']_2 \in U_i$

   (f) $[B]_1 =_\mathcal{E} [B]_Y =_\mathcal{E} [B]_Z$

   (g) $[b]_Y = [b]_1 = [b]_Z = [b']_2 \in \xi$

   (h) $[c]_Y = [c]_2 = [c]_Z = [c']_2 \in \xi$.

   The results follow from these facts.

6. Suppose $A \geq_O (\mu x{:}U_i)B$ and $A' \geq_O (\mu x{:}U_i)B'$. By definition:

   (a) $\forall X_i \in \mathcal{C}_i.\ [B]_Y[X_i/x] = [B']_Y[X_i/x] \in U_i$

   (b) $\boldsymbol{\lambda}\xi.[\![[B]_Y[rep_i(\xi)/x]]\!]_i \in \Xi_i \overset{\text{mon}}{\rightarrow} \Xi_i$

   (c) $\forall X_i \in \mathcal{C}_i.\ [B]_Z[X_i/x] = [B']_Z[X_i/x] \in U_i$.

   By induction:

   (d) $\forall X_i \in \mathcal{C}_i.\ [B]_1[X_i/x] = [B']_2[X_i/x] \in U_i$

   (e) $\forall X_i \in \mathcal{C}_i.\ [B]_1[X_i/x] =_\mathcal{E} [B]_Y[X_i/x] =_\mathcal{E} [B]_Z[X_i/x]$.

   The monotonicity of $\boldsymbol{\lambda}\xi.[\![[B]_1[rep_i(\xi)/x]]\!]_i$ follows from (b) and (e), and the result follows from these facts. The case for clause 11 is identical.

7. Suppose $A \geq_O (\mu x{:}U_j)B$ and $A' \geq_O (\mu x{:}U_j)B'$, for $j < i$. Then by definition:

   (a) $(\mu x{:}U_j)[B]_Y = (\mu x{:}U_j)[B']_Y \in U_j$.

Since $\tau_j \in \Xi_j$, it ignores Y_i, so all of the types $(\mu x : U_j)[B]_Y$, $(\mu x : U_j)[B']_Y$, $(\mu x : U_j)[B]_Z$, $(\mu x : U_j)[B']_Z$, $(\mu x : U_j)[B]_1$ and $(\mu x : U_j)[B']_2$ are equal in U_j, and thus, by cumulativity, equal in U_i. The case for clause 12 is identical.

□

As noted in step 5, this lemma depends on type stratification. For example, if it had asserted:

$$A[Y_i/y] = A'[Y_i/y] \in \tau_{i+1}^{\gamma} \textit{ and } A[Z_i/y] = A'[Z_i/y] \in \tau_{i+1}^{\gamma} \textit{ implies}$$

$$A[D_1/y] = A'[D_2/y] \in U_{i+1} \quad \textit{and} \quad A[D_1/y] =_{\varepsilon} A[Y_i/y] =_{\varepsilon} A[Z_i/y],$$

it would be false: let C be an inhabited type in U_1 (say, $X_1 \to X_1$), let D_1 and D_2 be $\{Y_i|\ C\}$ and let A and A' be $I(Y_i, y, U_i)$. The contradiction is that $I(Y_i, Y_i, U_i)$ is inhabited, while $I(Y_i, D_1, U_i)$ and $I(Y_i, Z_i, U_i)$ are not. This shows the edge between intensionality and extensionality that the lemma must balance upon.

Now we can list the facts for the μ and ν types.

Lemma 5.9 *Abbreviate* $\mu \equiv (\mu x : U_j)B$, $\mu' \equiv (\mu x : U_j)B'$, $\nu \equiv (\nu x : U_j)B$ *and* $\nu' \equiv (\nu x : U_j)B'$.

1. $\mu = \mu' \in U_j \;\Leftrightarrow\; \forall A = A' \in U_j.\ B[A/x] = B'[A'/x] \in U_j \wedge \forall A, A' \in U_j.\ A \trianglelefteq A' \Rightarrow B[A/x] \trianglelefteq B[A'/x]$
2. $a = a' \in \mu \;\Leftrightarrow\; \mu\ \textit{Type} \wedge a = a' \in B[\mu/x]$
3. $\nu = \nu' \in U_j \;\Leftrightarrow\; \forall A = A' \in U_j.\ B[A/x] = B'[A'/x] \in U_j \wedge \forall A, A' \in U_j.\ A \trianglelefteq A' \Rightarrow B[A/x] \trianglelefteq B[A'/x]$
4. $a = a' \in \nu \;\Leftrightarrow\; \nu\ \textit{Type} \wedge \textit{out}(a) = \textit{out}(a') \in B[\nu/x]$

Given $P \subseteq \mathcal{V} \times \mathcal{V}$, *define* $\mathcal{P}(A) \equiv \forall b = b' \in A.P(b, b')$; *then:*

5. $$\frac{\forall A \in U_j.\ A \trianglelefteq \mu \Rightarrow \mathcal{P}(A) \Rightarrow \mathcal{P}(B[A/x])}{\mathcal{P}(\mu)}$$

Given $C \to \nu$ *Type, and terms* $\textit{ind} \equiv \lambda w.\nu\_\textit{ind}(w;\ z, w.b)$ *and* $\textit{ind}' \equiv \lambda w.\nu\_\textit{ind}(w;\ z, w.b')$ *we can conclude* $\textit{ind} = \textit{ind}' \in C \to \nu$, *from:*

6. $\forall A \in U_j.\ \nu \trianglelefteq A \Rightarrow$ $$\frac{\textit{ind} = \textit{ind}' \in C \to A}{\lambda w.b[\textit{ind}/z] = \lambda w.b'[\textit{ind}'/z] \in C \to B[A/x]}$$

Proof.

1. Suppose $\mu = \mu' \in U_j$; then by the definition of $\mathcal{G}_j$:

   (a) $\forall X_j \in \mathcal{C}_j.\ B[X_j/x] = B'[X_j/x] \in U_j$, and

   (b) $\boldsymbol{\lambda}\xi.\ [\![B[rep_j(\xi)/x]]\!]_j \in \Xi_j \overset{\text{mon}}{\to} \Xi_j$.

   Fix $A = A' \in U_j$. Let $X_j \equiv rep_j([\![A]\!]_j)$; by lemma 5.8, $B[A/x] = B'[A'/x] \in U_j$. Fix $A, A' \in U_j$ such that $A \trianglelefteq A'$. Let $X_j \equiv rep_j([\![A]\!]_j)$ and $X'_j \equiv rep_j([\![A']\!]_j)$; then $X_j \trianglelefteq X'_j$, and so $[\![B[X_j/x]]\!]_j \subseteq [\![B[X'_j/x]]\!]_j$. By lemma 5.8, $[\![B[A/x]]\!]_j \subseteq [\![B[A'/x]]\!]_j$, and therefore $B[A/x] \trianglelefteq B[A'/x]$.

   For the converse, suppose:

   (a) $\forall A = A' \in U_j.\ B[A/x] = B'[A'/x] \in U_j$, and

   (b) $\forall A, A' \in U_j.\ A \trianglelefteq A' \Rightarrow B[A/x] \trianglelefteq B[A'/x]$.

   Since $X_j = X_j \in U_j$, (a) implies:

   $$\forall X_j \in \mathcal{C}_j.\ B[X_j/x] = B'[X_j/x] \in U_j.$$

   Fix $\xi, \xi' \in \Xi_j$ such that $\xi \subseteq \xi'$, and let $X_j \equiv rep_j(\xi)$ and $X'_j \equiv rep_j(\xi')$. Then by (b), $[\![B[X_j/x]]\!]_j \subseteq [\![B[X'_j/x]]\!]_j$, and thus:

   $$\boldsymbol{\lambda}\xi.\ [\![B[rep_j(\xi)/x]]\!]_j \in \Xi_j \overset{\text{mon}}{\to} \Xi_j,$$

   and by the definition of $\mathcal{G}_j$, $\mu = \mu' \in U_j$.

2. Suppose $a = a' \in \mu$; by the definition of $\mathcal{G}_\omega$, $\mu = \mu' \in U_j$ and μ *Type*. Let $X_j \equiv rep_j([\![\mu]\!]_j)$; by lemma 5.8, conclude $B[\mu/x] =_\mathcal{E} B[X_j/x]$, and by the definition of $\boldsymbol{\mu}_j$, $\mu =_\mathcal{E} B[X_j/x]$, thus $a = a' \in B[\mu/x]$.

3. – 4. By the same arguments for 1– 2.

5. Fix $P \subseteq \mathcal{V} \times \mathcal{V}$ and assume $\forall A \in U_j.\ A \trianglelefteq \mu \Rightarrow \mathcal{P}(A) \Rightarrow \mathcal{P}(B[A/x])$. We will prove, for $b = b' \in \mu$, that $P(b, b')$ by induction on an ascending chain in Ξ_j with limit $[\![\mu]\!]_j$. Define:

$$\begin{array}{rcl} \xi^0 & \equiv & \emptyset \\ \xi^{\gamma+1} & \equiv & [\![B[X_j^\gamma/x]]\!]_j \\ \xi^\lambda & \equiv & \bigcup_{\gamma<\lambda} \xi^\gamma, \text{ for limit } \lambda, \end{array}$$

where $X_j^\gamma \equiv rep_j(\xi^\gamma)$. The base and limit cases are trivial. Fix $b = b' \in X_j^{\gamma+1}$. Since $[\![\mu]\!]_j$ is an upper bound on the chain, $X_j^{\gamma+1} \trianglelefteq \mu$, and by induction $\mathcal{P}(X_j^\gamma)$, so by the initial assumption, $\mathcal{P}(B[X_j^\gamma/x])$. Since $X_j^{\gamma+1} =_\mathcal{E} B[X_j^\gamma/x]$, conclude $P(b,b')$.

6. Fix $C \to \nu$ *Type*, *ind* and *ind'*; assume:

$$\forall A \in U_j.\ \nu \trianglelefteq A \Rightarrow \frac{ind = ind' \in C \to A}{\lambda w.b[ind/z] = \lambda w.b'[ind'/z] \in C \to B[A/x]}.$$

We will prove $ind = ind' \in C \to \nu$ by induction on a descending chain in Ξ_j with limit $[\![\nu]\!]_j$. Define:

$$\begin{array}{rcl} \xi^0 & \equiv & \mathcal{V} \times \mathcal{V} \\ c = c' \in \xi^{\beta+1} & \equiv & out(c) = out(c') \in B[X_j^\beta/x] \\ \xi^\lambda & \equiv & \bigcap_{\beta<\lambda} \xi^\beta, \text{ for limit } \lambda, \end{array}$$

where $X_j^\beta \equiv rep_j(\xi^\beta)$. The base and limit cases are trivial. Fix X_j^γ and $e = e' \in C$; since $[\![\nu]\!]_j$ is a lower bound of the chain, $\nu \trianglelefteq X_j^\gamma$. By induction, $ind = ind' \in C \to X_j^\gamma$, so by the initial assumption, $b[ind, e/z, w] = b'[ind', e'/z, w] \in B[X_j^\gamma/x]$. Since $out(ind(e)) \simeq b[ind, e/z, w]$ and $out(ind'(e')) \simeq b'[ind', e'/z, w]$, we may conclude $ind = ind' \in C \to X_j^{\gamma+1}$.

□

5.3 Truth and soundness

We define the predicates $a_1 = a_2 \in \Gamma$ and $\Gamma \models b = b' \in B$ as in 4.3. All the proofs of soundness of rules from that section carry over to the context of this section. Thus, all that remains is to by verified is the soundness of the proof rules for the $\subseteq$, μ and ν types.

5.3.1 Soundness of the $\subseteq$ type rules

Lemma 5.10 $\Gamma \models B \subseteq C = B' \subseteq C' \in U_j$ *is implied by*

1. $\Gamma \models B = B' \in U_j$
2. $\Gamma \models C = C' \in U_j$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove:

3. $[B \subseteq C]_1 = [B' \subseteq C']_2 \in U_j$.

1 and *2* imply

4. $[B]_1 = [B']_2 \in U_j$, and
5. $[C]_1 = [C']_2 \in U_j$,

and by lemma 5.5, 4 and 5 imply 3.
□

Lemma 5.11 $\Gamma \models \mathit{true} = \mathit{true} \in B \subseteq C$ *is implied by*

1. $\Gamma,\ x{:}B \models x \in C$
2. $\Gamma \models B \subseteq C \in U_j$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove:

3. $[B \subseteq C]_1 = [B \subseteq C]_2$ and
4. $\mathit{true} = \mathit{true} \in [B \subseteq C]_1$.

3 is implied by cumulativity and *2*. Fix $b = b' \in [B]_1$; *1* implies

5. $b = b' \in [C]_1$,

thus 4 holds by the lemma 5.5.
□

Lemma 5.12 $\Gamma \models b = b' \in C$ *is implied by*

1. $\Gamma \models b = b' \in B$
2. $\Gamma \models t \in B \subseteq C$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove:

3. $[C]_1 = [C]_2$ and
4. $[b]_1 = [b']_2 \in [C]_1$.

1 and *2* imply

5. $[b]_1 = [b']_2 \in [B]_1$,
6. $[B \subseteq C]_1 = [B \subseteq C]_2$ and
7. $[t] \in [B \subseteq C]_1$.

By lemma 5.5, 6 implies 3, and 7 and 5 imply 4.
□

5.3.2 Soundness of the μ type rules

For the next three lemmas, let $\mu \equiv (\mu x{:}U_j)B$ and $\mu' \equiv (\mu x{:}U_j)B'$.

Lemma 5.13 $\Gamma \models \mu = \mu' \in U_j$ *is implied by*

1. $\Gamma,\ x{:}U_j \models B = B' \in U_j$
2. $\Gamma,\ x{:}U_j,\ y{:}U_j,\ x \subseteq y \models t \in B \subseteq B[y/x]$

Proof. Fix $\boldsymbol{a}_1 = \boldsymbol{a}_2 \in \Gamma$; it suffices to prove:

3. $[\mu]_1 = [\mu']_2 \in U_j$,

which, by lemma 5.9 is implied by

4. $\forall A = A' \in U_j.\ [B]_1[A/x] = [B']_2[A'/x] \in U_j$, and
5. $\forall A, A' \in U_j.\ A \trianglelefteq A' \Rightarrow [B]_1[A/x] \trianglelefteq [B]_1[A'/x]$.

Now, *1* implies 4 and *2* implies 5.
□

Lemma 5.14 $\Gamma \models b = b' \in \mu$ *is implied by*

1. $\Gamma \models b = b' \in B[\mu/x]$

2. $\Gamma \models \mu \in U_j$

Proof. Fix $\boldsymbol{a} = \boldsymbol{a}' \in \Gamma$; it suffices to show

3. $[\mu]_1 = [\mu]_2$, and

4. $[b]_1 = [b']_2 \in [\mu]_1$.

1 and *2* imply

5. $[b]_1 = [b']_2 \in [B[\mu/x]]_1$, and

6. $[\mu]_1 = [\mu]_2 \in U_j$.

By cumulativity, 6 implies 3; by lemma 5.9, 5 and 6 imply 4.
□

Lemma 5.15 $\Gamma \models \mu\_ind(b;\ z, y.d) = \mu\_ind(b';\ z, y.d') \in D[b/y]$ *is implied by*

1. $\Gamma,\ x{:}U_j,\ x \subseteq \mu,\ z{:}(\Pi y{:}x)D,\ y{:}B \models d = d' \in D$

2. $\Gamma \models b = b' \in \mu$

Proof. Fix $\boldsymbol{a} = \boldsymbol{a}' \in \Gamma$. Let $ind \equiv \lambda y.\mu\_ind(y;\ z, y.[d]_1)$ and $ind' \equiv \lambda y.\mu\_ind(y;\ z, y.[d']_2)$. Let Γ' be the context of subgoal *1*. By *2*,

3. $[\mu]_1 = [\mu]_2$, and

4. $[b]_1 = [b']_2 \in [\mu]_1$.

The lemma follows from showing, for any $c = c' \in [\mu]_1$,

5. $[D]_1[c/y] = [D]_2[c'/y]$ and

6. $ind(c) = ind'(c') \in [D]_1[c/y]$,

which we prove by using the induction principle of lemma 5.9.

Fix $A \in U_j$ such that $A \trianglelefteq [\mu]_1$, and assume 5 and 6 hold for $c = c' \in A$. (Note $[B]_1[A/x] \in U_j$, by lemma 5.9.) For $c = c' \in [B]_1[A/x]$ we have

$$\boldsymbol{a}_1, A, true, ind, c = \boldsymbol{a}_2, A, true, ind', c' \in \Gamma',$$

so by *1*,

7. $[D]_1[c/y] = [D]_2[c'/y]$, and

8. $[d]_1[ind, c/z, y] = [d']_2[ind', c'/z, y] \in [D]_1[c/y]$.

$ind(c) \simeq [d]_1[ind, c/z, y]$ and $ind'(c') \simeq [d']_2[ind', c'/z, y]$, so 8 implies 6. This completes the argument.
□

5.3.3 Soundness of the ν type rules

For the next three lemmas, let $\nu \equiv (\nu x{:}U_j)B$ and $\nu' \equiv (\nu x{:}U_j)B'$.

Lemma 5.16 $\Gamma \models \nu = \nu' \in U_i$ *is implied by*

1. $\Gamma,\ x{:}U_j \models B = B' \in U_j$
2. $\Gamma,\ x{:}U_j,\ y{:}U_j,\ x \subseteq y \models t \in B \subseteq B[y/x]$

Proof. Like lemma 5.13.
□

Lemma 5.17 $\Gamma \models out(b) = out(b') \in B[\nu/x]$ *is implied by*

1. $\Gamma \models b = b' \in \nu$
2. $\Gamma \models \nu \in U_j$

Proof. Like lemma 5.14.
□

Lemma 5.18 $\Gamma \models \nu\_ind(c;\ z, w.b) = \nu\_ind(c';\ z, w.b') \in \nu$ *is implied by*

1. $\Gamma,\ x{:}U_j,\ \nu \subseteq x,\ z{:}C \to x,\ w{:}C \models b = b' \in B$
2. $\Gamma \models c = c' \in C$
3. $\Gamma \models \nu \in U_j$

Proof. Fix $\boldsymbol{a} = \boldsymbol{a}' \in \Gamma$. Let $ind \equiv \lambda w.\nu\_ind(w;\ z, w.[b]_1)$ and $ind' \equiv \lambda w.\nu\_ind(w;\ z, w.[b'])$. Let Γ' be the context of subgoal *1*. By *2* and *3*,

4. $[d]_1 = [d']_2 \in [C]_1$, and

5. $[\nu]_1 = [\nu]_2 \in U_j$.

Fix $A \in U_j$ such that $[\nu]_1 \trianglelefteq A$ and $ind = ind' \in [C]_1 \to A$. Fix $e = e' \in [C]_1$; we have

$$\boldsymbol{a}_1, A, true, ind, e = \boldsymbol{a}_2, A, true, ind', e' \in \Gamma',$$

so by *1*,

$$b[ind, e/z, w] = b'[ind', e'/z, w] \in [B]_1[A/x].$$

Since $out(ind(e)) \simeq b[ind, e/z, w]$ and $out(ind'(e')) \simeq b'[ind', e'/z, w]$, we see $ind = ind' \in [C]_1 \to [B]_1[A/x]$. So by lemma 5.9, $ind = ind' \in [C]_1 \to [\nu]_1$. This completes the argument.
□

5.4 Concluding consistency

As in 4.4, we can conclude all derivable sequents are true, and in the case of $\vdash b = b' \in B$, this again implies $b = b' \in B$. As in 4.4, from this we can conclude the intuitionistic consistency of the theory.

5.5 Strong positivity

One property that the introduction rule for inductive types must enforce is monotonicity of the body of the type. Our given rules do this directly with $\subseteq$ types, but in this section we introduce alternative introduction rules, using the syntactic condition of strong positivity, and show their soundness. The situation here is more complicated than in chapter 3, because the type expression may computationally "juggle" the variable of interest, only to have it land on the left hand side of a Π type (that is, in subterm B of $(\Pi x : B)C$). The solution is to ensure that the strong positivity condition is preserved under evaluation.

Definition 5.6 *x is* strongly positive *with respect to B ($SP(B, x)$) iff x does not occur free:*

1. *on the left hand side of a Π or $\subseteq$ type in B.*
2. *in a term b where $c(b)$ is a subterm of B.*

3. in a principle argument of any other eliminations form which is a subterm of B.

□

The approach to showing soundness will be as in chapter 3: show certain monotonicity properties of the operations in definitions 4.1 and 5.3, then conclude strong positivity implies monotonicity by an induction on the construction of σ_i. We begin by defining two partial orderings of $IFAM(\Xi)$.

Definition 5.7 *For* $\langle \xi,\ \psi\rangle, \langle \xi',\ \psi'\rangle \in IFAM(\Xi)$*:*

$$\begin{aligned}\langle \xi,\ \psi\rangle \sqsubseteq_1 \langle \xi',\ \psi'\rangle &\equiv \xi \subseteq \xi' \wedge \forall a \in \xi.\ \psi([a]_\xi) \subseteq \psi'([a]_{\xi'})\\ \langle \xi,\ \psi\rangle \sqsubseteq_2 \langle \xi',\ \psi'\rangle &\equiv \xi = \xi' \wedge \forall a \in \xi.\ \psi([a]_\xi) \subseteq \psi'([a]_{\xi'})\end{aligned}$$

□

So for functions in $IFAM(\xi) \to \Xi$ we will have the notion of 1-monotonic and 2-monotonic.

Lemma 5.19 *Operation $+$ is monotonic in both arguments, $\subseteq$ is monotonic in its second argument, and $\boldsymbol{I}$ is monotonic; $\boldsymbol{\Sigma}$ and $\{\_|\_\}$ are 1-monotonic, while $\boldsymbol{\Pi}$ is 2-monotonic; $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are monotonic.*

Proof.

Each case is easily verified; we present the $\boldsymbol{\Sigma}$ and $\boldsymbol{\Pi}$ cases as examples.

Suppose $\langle \xi,\ \psi\rangle \sqsubseteq_1 \langle \xi',\ \psi'\rangle$; fix $\langle b,\ c\rangle = \langle b',\ c'\rangle \in \boldsymbol{\Sigma}(\xi,\psi)$. By 1-monotonicity, $b\xi' b$ and $c = c' \in \psi'([b])$, thus $\langle b,\ c\rangle = \langle b',\ c'\rangle \in \boldsymbol{\Sigma}(\xi',\psi')$. Conclude $\boldsymbol{\Sigma}$ is 1-monotonic.

Suppose $\langle \xi,\ \psi\rangle \sqsubseteq_2 \langle \xi',\ \psi'\rangle$; fix $\lambda x.c = \lambda x.c' \in \boldsymbol{\Pi}(\xi,\psi)$ and $b = b' \in \xi'$. By 2-monotonicity, $b\xi b$ and $c[b/x] = c'[b'/x] \in \psi'([b])$. Conclude $\boldsymbol{\Pi}$ is 2-monotonic.

□

Lemma 5.20 $\forall X_i, Y_i \in \mathcal{C}_i$*. such that* $X_i \trianglelefteq Y_i$, $\forall \gamma.\ \forall A[X_i/y], A'[X_i/y] \in \mathcal{T}$*. such that* $SP(A,y)$ *and* $SP(A',y)$*:*

(i) $A[X_i/y] = A'[X_i/y] \in \tau_i^\gamma \wedge A[Y_i/y] = A'[Y_i/y] \in \tau_i^\gamma$ *implies*

(ii) $A[X_i/y] \trianglelefteq A[Y_i/y]$.

Proof. Fix i and $X_i, Y_i \in \mathcal{C}_i$ such that $X_i \trianglelefteq Y_i$, and induct on γ. Each clause in the definition of $\mathcal{G}_i$ is straightforward, and we present the most problematic case here, for the Σ type.

Suppose $A \geq_O (\Sigma x : B)C$ and $A' \geq_O (\Sigma x : B')C'$. Assume *(i)*; by the 1-monotonicity of $\boldsymbol{\Sigma}$, $(\Sigma x : [B]_X)[C]_X \trianglelefteq (\Sigma x : [B]_Y)[C]_Y$ follows from:

1. $[B]_X \trianglelefteq [B]_Y$ and
2. $\forall b \in [B]_X.\ [C]_X[b/x] \trianglelefteq [C]_Y[b/x]$.

By the definition of $\mathcal{G}_i$ and *(i)*,

3. $[B]_X = [B']_X \in U_i \wedge [B]_Y = [B']_Y \in U_i$

and 1 follows by induction. Fix $b = b' \in [B]_X$; by 1, $b = b' \in [B]_Y$, so by the definition of $\mathcal{G}_i$ and *(i)*,

4. $[C]_X[b/x] = [C']_X[b'/x] \in U_i \wedge [C]_Y[b/x] = [C']_Y[b'/x] \in U_i$,

and by induction, 4 implies 2.
□

Corollary 5.1 *If* $SP(A, x)$, $SP(A', x)$ *and* $\forall X_i \in \mathcal{C}_i.\ A[X_i/x] = A'[X_i/x] \in U_i$ *then* $\boldsymbol{\lambda}\xi.[\![A[rep_i(\xi)/x]]\!]_i \in \Xi_i \overset{\text{mon}}{\longrightarrow} \Xi_i$.

Now the soundness of following introduction rules follow from corollary 5.1 without difficulty.

$$\Gamma \vdash (\mu x : U_j)B = (\mu x : U_j)B' \in U_j \quad (\text{if } SP(B, x))$$
$$\Gamma,\ x : U_j \vdash B = B' \in U_j$$

$$\Gamma \vdash (\nu x : U_j)B = (\nu x : U_j)B' \in U_j \quad (\text{if } SP(B, x))$$
$$\Gamma,\ x : U_j \vdash B = B' \in U_j$$

5.6 Parameterized inductive types

In this section we take the constructions of sections 1-4 and adapt them to model parameterized inductive types. The change is a uniform one, and basically amounts to shifting the intended meanings of the constants in $\mathcal{C}$ from types to families of types. The following is an outline of the changes to be made. We will use the abbreviations:

$$
\begin{array}{rcl@{\qquad}rcl}
\mu@t & \equiv & (\mu x{:}C \to U_\alpha)B@t & \mu & \equiv & \lambda w.\mu@w \\
\mu'@t & \equiv & (\mu x{:}C' \to U_\alpha)B'@t & \mu' & \equiv & \lambda w.\mu'@w \\
\nu@t & \equiv & (\nu x{:}C \to U_\alpha)B@t & \nu & \equiv & \lambda w.\nu@w \\
\nu'@t & \equiv & (\nu x{:}C' \to U_\alpha)B'@t & \nu' & \equiv & \lambda w.\nu'@w \\
A \trianglelefteq_C A' & \equiv & \forall c \in C.\ A(c) \trianglelefteq A'(c) & & & \\
\end{array}
$$

5.6.1 Ground relations and ground type families

1. Instead of the constants in $\mathcal{C}_i$ representing ground relations, they should now represent *families* of ground relations. So $\mathcal{C}_i$ is defined to be a set of new constants, two for each element of $FAM(PER(\mathcal{C}_{<i}))$. These constants should now more properly be called *ground type families.*

2. Evaluation must treat $X_i \in \mathcal{C}_i$ as a function constant, so we add the following clause to the definition of $\geq$.

$$\frac{a \geq X_i}{a(b) \geq X_i(b)}$$

3. The mappings val_i and rep_i can be moved up to families in the manner of step 1, so we have $val_i \in \mathcal{C}_i \to FAM(\Xi_i)$ and rep_i as its right inverse. Let $dom(X_i) \equiv \xi$, where ξ is the relation in Ξ_i such that $val_i(X_i) \in FAM(\xi, \Xi_i)$.

4. By the point-wise ordering inherited from Ξ_i, $FAM(\Xi_i)$ is a complete lattice. Redefine operations $\boldsymbol{\mu}_j$ and $\boldsymbol{\nu}_j$ to be in:

$$\prod_{\xi \in \Xi_j} (\Psi \stackrel{\text{mon}}{\to} \Psi) \stackrel{\text{mon}}{\to} \Psi,$$

where $\Psi \equiv FAM(\xi, \Xi_j)$, by letting:

$$\boldsymbol{\mu}_j(\xi)(\phi) \equiv \text{ the least fixed point of } \phi,$$

and letting $\boldsymbol{\nu}_j(\xi)(\phi)$ be the greatest fixed point $\psi \in \Psi$ of:

$$\forall b \in \xi.\ a = a' \in \psi[b] \Leftrightarrow out(a) = out(a') \in \phi(\psi)[b].$$

5.6.2 Type systems

Alter definition 5.2 by replacing clause 3 by:

3. (a) $\exists j \leq \alpha.\ A \geq X_j(c) \wedge A' \geq X_j(c')$
   (b) $c = c' \in dom(X_j)$

   - $[\![X_j(c)]\!]' \equiv val_j(X_j)([c])$

Also, replace 11 and 13 with the obvious analogues, and 10 and 12 by:

10. (a) $\alpha < \omega.\ A \geq (\mu x{:}C \to U_\alpha)B@c \wedge A' \geq (\mu x{:}C' \to U_\alpha)B'@c'$
    (b) $C \tau C' \wedge c = c' \in [\![C]\!]$ (let $\xi \equiv [\![C]\!]$ and $\Psi \equiv FAM(\xi, \Xi_\alpha)$)
    (c) $\forall X_\alpha \in \mathcal{C}_\alpha.dom(X_\alpha) = \xi \Rightarrow B[X_\alpha/x] = B'[X_\alpha/x] \in \xi \to \tau$
    (d) $\phi \equiv \lambda\psi.\lambda[d].[\![B[rep_\alpha(\psi)/x](d)]\!] \in \Psi \xrightarrow{\text{mon}} \Psi$

    - $[\![(\mu x{:}C)B@c]\!]' \equiv \boldsymbol{\mu}_j(\xi)(\phi)[c]$

12. (a) $\alpha < \omega.\ A \geq (\nu x{:}C \to U_\alpha)B@c \wedge A' \geq (\nu x{:}C' \to U_\alpha)B'@c'$
    (b) $C \tau C' \wedge c = c' \in [\![C]\!]$ (let $\xi \equiv [\![C]\!]$ and $\Psi \equiv FAM(\xi, \Xi_\alpha)$)
    (c) $\forall X_\alpha \in \mathcal{C}_\alpha.dom(X_\alpha) = \xi \Rightarrow B[X_\alpha/x] = B'[X_\alpha/x] \in \xi \to \tau$
    (d) $\phi \equiv \lambda\psi.\lambda[d].[\![B[rep_\alpha(\psi)/x](d)]\!] \in \Psi \xrightarrow{\text{mon}} \Psi$

    - $[\![(\nu x{:}C)B]\!]'@c \equiv \boldsymbol{\nu}_j(\xi)(\phi)[c]$

In the style of lemma 5.9, by a substitution lemma similar to 5.8, we will be able to conclude the following facts about the parameterized μ and ν types.

1. $\mu@c = \mu'@c' \Leftrightarrow$
   (a) $C = C' \in U_j$
   (b) $c = c' \in C$
   (c) $\forall A = A' \in C \to U_j.B[A/x] = B'[A'/x] \in C \to U_j$
   (d) $\forall A, A' \in C \to U_j.A \trianglelefteq_C A' \Rightarrow B[A/x] \trianglelefteq_C B'[A'/x]$
2. $a = a' \in \mu@c \Leftrightarrow \mu@c\ Type \wedge a = a' \in B[\mu/x](c)$

3. Given $P \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{V} \times \mathcal{V}$, define $\mathcal{P}(A) \equiv \forall c = c' \in C.\ a = a' \in A(c).P(c, c', a, a')$; then:

$$\frac{\forall A \in C \to U_j.\ A \trianglelefteq_C \mu \Rightarrow \mathcal{P}(A) \Rightarrow \mathcal{P}(B[A/x])}{\mathcal{P}(\mu)}$$

4. $\nu@c = \nu'@c' \Leftrightarrow$
   (a) $C = C' \in U_j$
   (b) $c = c' \in C$
   (c) $\forall A = A' \in C \to U_j.B[A/x] = B'[A'/x] \in C \to U_j$
   (d) $\forall A, A' \in C \to U_j.A \trianglelefteq_C A' \Rightarrow B[A/x] \trianglelefteq_C B'[A'/x]$

5. $a = a' \in \nu(c) \Leftrightarrow \nu@c\ \mathit{Type} \wedge \mathit{out}(a) = \mathit{out}(a') \in B[\nu/x](c)$

6. Given $(\Pi w:C)\nu@w\ \mathit{Type}$, and terms $\mathit{ind} \equiv \lambda w.\nu\_\mathit{ind}(w;\ z, w.b)$ and $\mathit{ind}' \equiv \lambda w.\nu\_\mathit{ind}(w;\ z, w.b')$ we can conclude $\mathit{ind} = \mathit{ind}' \in (\Pi w : C)\nu@w$, if for all $A \in C \to U_j$ such that $\nu \trianglelefteq_C A$:

$$\frac{\mathit{ind} = \mathit{ind}' \in (\Pi w:C)A(w)}{\lambda w.b[\mathit{ind}/z] = \lambda w.b'[\mathit{ind}'/z] \in (\Pi w:C)B[A/x](w)}.$$

5.6.3 Truth and soundness

With the six facts just listed we can prove the soundness of the proof rules for parameterized inductive type by arguments similar to the ones for their simple versions. As an example, we prove the soundness of the μ induction rule.

Lemma 5.21
$\Gamma \models \mu\_\mathit{ind}(c; b;\ z, w, y.d) = \mu\_\mathit{ind}(c'; b';\ z, w, y.d') \in D[c, b/w, y]$

1. $\Gamma,\ x:C \to U_j,\ x \subseteq_C \mu,\ z:(\Pi w:C)(\Pi y:x(w))D,\ w:C,\ y:B(w)$
   $\models d = d' \in D$
2. $\Gamma \models c = c' \in C$
3. $\Gamma \models b = b' \in \mu@c$

Proof. Fix $\boldsymbol{a} = \boldsymbol{a}' \in \Gamma$. Let $ind \equiv \lambda w, y.\mu\_ind(w; y;\ z, w, y.[d]_1)$ and $ind' \equiv \lambda w, y.\mu\_ind(w; y;\ z, w, y.[d']_2)$. By *2* and *3*:

4. $[C]_1 = [C]_2$
5. $[c]_1 = [c']_2 \in [C]_1$
6. $[\mu@c]_1 = [\mu@c]_2$
7. $[b]_1 = [b']_2 \in [\mu@c]_1$

The lemma follows from showing, for $e = e' \in [C]_1$ and $f = f' \in [\mu@e]_1$,

8. $[D]_1[e, f/w, y] = [D]_2[e', f'/w, y]$ and
9. $ind(e)(f) = ind'(e')(f') \in [D]_1[e, f/w, y]$,

which we prove by the induction principle of the previous subsection.

Fix $A \in [C]_1 \to U_j$ such that $A \trianglelefteq_C [\mu]_1$, and assume 7 and 8 hold for $e = e' \in [C]_1$ and $f = f' \in [A]_1(e)$. For $e = e' \in [C]_1$ and $f = f' \in [B]_1[A/x](e)$ we have

$$\boldsymbol{a}_1, A, \lambda w.true, ind, e, f = \boldsymbol{a}_2, A, \lambda w.true, ind', e', f' \in \Gamma',$$

where Γ' is the context of *1*. By *1*,

10. $[D]_1[e, f/w, y] = [D]_2[e', f'/w, y]$, and
11. $[d]_1[ind, e, f/z, w, y] = [d']_2[ind', e', f'/z, w, y] \in [D]_1[e, f/w, y]$.

Thus, $ind(e)(f) \simeq [d]_1[ind, e, f/z, w, y]$, $ind'(e')(f) \simeq [d']_2[ind', e', f'/z, w, y]$, and so 11 implies 9. This completes the argument.
□

As in the simple case, consistency follows from soundness.

5.7 The Δ type

In this section we demonstrate how the methods of this chapter can be used to prove the consistency of a stratified version of Δ, the type abstraction type constructor of chapter 3. We begin by listing the expected proof rules for such a type constructor. (For this section, let $\Delta \equiv (\Delta x : U_j)B$ and $\Delta' \equiv (\Delta x : U_j)B'$.)

$$\Gamma \vdash \Delta = \Delta' \in U_j$$
$$\Gamma,\ x{:}U_j \vdash B = B' \in U_j$$

$$\Gamma \vdash b = b' \in \Delta$$
$$\Gamma,\ x{:}U_j \vdash b = b' \in B$$
$$\Gamma \vdash \Delta \in U_j$$

$$\Gamma \vdash b = b' \in B[A/x]$$
$$\Gamma \vdash b = b' \in \Delta$$
$$\Gamma \vdash A \in U_j$$

Note that the second-order lambda calculus can be embedded in U_1 using this type constructor. It would appear that the semantic account of the previous sections should support such a type constructor, and indeed, this is the case. In this final section we list the alterations needed to verify the soundness of the Δ rules.

1. For $j > 0$, define the operations $\boldsymbol{\Delta}_j \in (\Xi_j \to \Xi) \to \Xi$, for $j > 0$, by:

   $$\boldsymbol{\Delta}_j(\phi) \equiv \bigcap_{\xi \in \Xi_j} \phi(\xi).$$

   When restricted to $\Xi_j \to \Xi_j$, $\boldsymbol{\Delta}_j$'s range is Ξ_j.

2. Add to the definition of $\mathcal{G}_\alpha$ the two clauses:

   (a) $\alpha < \omega \wedge A \geq (\Delta x{:}U_\alpha)B \wedge A' \geq (\Delta x{:}U_\alpha)B'$

   (b) $\forall X_\alpha \in \mathcal{C}_\alpha.\ B[X_\alpha/x] = B'[X_\alpha/x] \in \tau$

   - $[\![(\Delta x{:}U_\alpha)B]\!]' \equiv \boldsymbol{\Delta}_\alpha(\boldsymbol{\lambda}\xi.[\![B[rep_\alpha(\xi)/x]]\!])$

   (a) $\exists j < \alpha.\ A \geq \Delta \wedge A' \geq \Delta'$

   (b) $\Delta = \Delta' \in \tau_j$

   - $[\![\Delta]\!]' \equiv [\![\Delta]\!]_j$

3. In the style of lemma 5.9 we have the following properties.

   (a) $\Delta = \Delta' \in U_j \Leftrightarrow \forall A = A' \in U_j.\ B[A/x] = B'[A'/x] \in U_j$

   (b) $a = a' \in \Delta \Leftrightarrow \Delta\ \mathit{Type} \wedge \forall A \in U_j.\ a = a' \in B[A/x]$

The soundness of the Δ type rules follow from this without difficulty.

Summary

We have relativized the semantics given in chapter 4 to account for simple and parameterized inductive types. The main obstacle, the impredicativity of their definition, was overcome by identifying an appropriate collection of relations for each universe level and introducing constants for each such relation. In a key lemma of this chapter, we showed how operations on a universe level must preserve extensional equality, which lead to a justification of the inductive types' proof rules. We also demonstrated how this semantics readily could be used to justify a stratified version of the Δ type constructor of chapter 3.

Chapter 6

Conclusions

We have addressed the problem of extending a type theory by inductive types, in the context of the NuPrl type theory. We review the results of the previous chapters and discuss research directions.

6.1 Results

What we wished to capture directly in the type theory were forms of inductive and co-inductive definition. We achieved this through the μ and ν type constructors, respectively, which allow us to solve certain type equations for their least and greatest solutions. The induction principle associated with the μ types let us define well-founded recursive functions, and the dual principle for the ν types let us inductively define their "infinite" elements. A domain theoretic solution would not have been acceptable in this context, because it would not be sensible in the framework of propositions-as-types.

In chapter 2, after an introduction to type theory, we presented proof rules for these new type constructors. We demonstrated how they can be used to define types such as the natural numbers, lists, finite trees, well-founded trees, mutually defined data types, inductively defined predicates, parameterized inductive data types, streams and infinite trees. Among the features of the proof rules, we demonstrated how an inductive principle for the μ type, which takes advantage of the information hiding properties of the $\{\_ \mid \_\}$ type, can be used to constructively define an unbounded search operator, or more generally, to compute under the assumption of inhabitation.

In the following chapters we turned our attention to matters of semantics and consistency. Before attempting to formally verify the consistency of the type theory with inductive types, in chapter 3 we considered the same question in the simpler setting of the second-order lambda calculus, which allowed the separation of the core of the consistency argument from the additional concerns of the type theory. The property corresponding to consistency in this situation is strong normalizability. We examined recursive definition in two ways in this chapter. In 3.1, type constructors μ and ν are added and strong normalizability of typed terms was shown using Girard's *candidat de réductibilité* method. While in 3.2, we considered typing terms in the presence of equational type constraints. We gave a condition for determining if a set of such constraints admits the typing of only strongly normalizing terms, or the typing of a diverging term.

Returning to the type theory introduced in chapter 2, in chapter 4 we built a type-free model for it. In such a type theory, with dependent type constructors, type universes and intensional equality types, one is defining type expressions and terms simultaneously, so the methods used in the previous chapter must be modified. We identified a well-founded order in which types may be thought of as being constructed, by viewing type construction as a monotonic operation on a suitable complete partial order, denoted $\mathcal{TS}$. The elements of $\mathcal{TS}$ represent systems of types and their order relation corresponds to the consistent enlargement of a system of types. Monotonic operations $\mathcal{F}_i$ were defined on $\mathcal{TS}$ to model the closure of a type universe U_i under the basic type constructors. The model of U_i is used in the definition of $\mathcal{F}_j$, for $j > i$, because U_i is an element of U_j. Thus, the models in the hierarchy of universes are built in an iterative fashion.

There is a subtle point in this construction. A type in U_1, or a term in such a type, may have subterms of higher universe level — how is this justified, given that we have just said these models are built iteratively? Simply by having a non-compositional definition that exploits the computational nature of terms, interleaving evaluation with decomposition. For example, an expression A is a admitted as a type in U_1 if it *evaluates* (as defined by the type-free relation $\geq$) to $(\Pi x : B)C$, and B has been previously admitted to U_1, and for $b = b' \in B$, $C[b/x]$ and $C[b'/x]$ are equal types, previously admitted into U_1. This provides an direct solution to the seemingly impredicative definition of terms and types.

With such a model, it was a straightforward matter to define the judgements of typehood, type equality, member equality in a type and membership in a type, and to isolate the basic properties of these judgements. We extended the meanings of judgements to define truth for sequents and showed the soundness of the proof rules. From soundness, we concluded intuitionistic consistency, because in our model *void* is uninhabited, and therefore under the propositions-as-types correspondence, we had propositional consistency — *False* is not provable.

In chapter 5 we based our relativization of chapter 4's construction on the approach of chapter 3, but before reviewing further, it is illuminating to look at the proof of the strong normalizability of the second-order calculus given in [19]. There, rather than using an environment to give meaning to the free type variables, the definition of types was extended by introducing a type constant for each ground set. Its proof had a circularity problem: by introducing more types, more ground sets are created, as ground sets are defined with respect to sets of typed terms. This error can be corrected by using untyped terms in ground sets, as in chapter 3. While terms in our type theory are untyped, there is a corresponding problem, because types are first-class terms, e.g., $\lambda x.x \rightarrow U_1$ is a term. In this situation, to attempt to identify a collection of ground relations on terms and then to add constants for each such relation seems doomed to the same circularity problems, unless we recognize that we can identify a separate collection of ground relations for each universe level and have these relations ignore the constants of its own, and higher, levels. Given these constants to represent the classes of ground relations, it is easy to break the impredicativity in the definitions of the μ and ν types. From this point on, we merely follow the line of argument of chapter 4, and eventually conclude with the consistency of the type theory, extended by inductive types.

In 5.5, we showed how a syntactic positivity condition can often replace an explicit monotonicity subgoal in the formation rules for inductive types. While the earlier proof rules in chapter 2 more immediately reflect the semantics, in practice these other rules are more convenient. Up to this point, all the work in chapter 5 had been carried out in terms of the simple μ and ν types, but in section 5.6 we outlined its modification to the general, parameterized case. Finally, in 5.7 we presented a stratified version of the Δ operator and showed that our semantic account could justify the impredicativity in its definition as well.

As for the computer implementation of the proof rules: an earlier version of μ types were implemented in the NUPRL system, by Tim Griffin, at Cornell. They have been used in the work reported in the thesis of Knoblock [31], to represent data types of possibly incomplete proofs, and in the thesis of Cleaveland [9], where they were used to model synchronization trees. The current versions of the μ and ν types are expected to be included in the next version of the system, which is currently being designed.

6.2 Research directions

Constructive type theory is an exciting and open area of study. We end with a brief discussion of other work now being done on relevant matters, and other possible research directions.

One major research direction for the NUPRL project is explored in [31]: the reflection of the NUPRL metatheory into a related type theory, thus allowing one to bring the power of the system to bear on metatheoretic issues. Inductively defined types play an important part in this setting, and there is much work to be done in this area.

We did not consider admitting diverging terms as members of types, but such an approach to constructive type theory is sensible and natural, if one considers the computation itself to be an important matter for analysis. This is currently being investigated by Constable, Smith and Basin [38].

Recent work by Harper and Mitchell is being done on classes of models for type theory [24]. One item on their agenda is to extend their models to type constructors like the ones described in this thesis. Such models would be especially interesting if they could yield a compositional semantics.

Another direction in type theory is the classification of the expressive powers of particular theories. Aczel has studied a type theory with one universe level [1], but there is a problem that as one develops more complex, non-standard type theories, logics which can capture their expressiveness mirror the type theory's definition so closely that they beg the question.

Comparisons between different type theories can also be illuminating. One of the most interesting would be between a predicative theory, such as the NUPRL logic, and an impredicative one, such as the Theory of Constructions. This thesis is a step in that direction because it shows how to

treat in a predicative manner some interesting impredicative concept.

Appendix A

Definition of the Basic Type Theory

The core of the intuitionistic type theory used in this thesis is defined in this appendix. In appendix B, we give its extension by inductive types. In each appendix we begin by defining the terms of the theory and the evaluation relation $\geq$; then we present the proof rules.

A.1 Terms

We begin with an inductive definition of terms, then specify how variables become bound in them. As usual, we identify terms that are alphabetic variants in their bound variables.

Assume we have a infinite list of variables $v_1, v_2, \ldots$ and let w, x, y and z range over them. Let a, b, c, d, e and A, B, C, D and E range over terms. We define the terms of the basic theory inductively, by the following clauses.

- Variables are terms.
- *void* is a term, and U_i is a term for $i > 0$.
- $B + C$, $(\Pi x{:}B)C$, $(\Sigma x{:}B)C$, $\{x{:}B|\ C\}$ and $I(a, b, B)$ are terms.
- *true*, $\lambda x.b$, $\langle a,\ b\rangle$, $inl(a)$, and $inr(a)$ are terms.
- $a(b)$, $spread(a; x, y.b)$, $decide(a; x.b; y.c)$ and $any(a)$ are terms.

We now define how occurrences of variables become bound in terms.

- In $(\Pi x : B)C$, $(\Sigma x : B)C$ and $\{x : B|\ C\}$, the x in front of the colon and all free occurrences of x in C become bound.
- In $\lambda x.b$, the x in front of the dot and all free occurrences of x in b become bound.
- In $spread(a; x, y.b)$, the x and y in front of the dot and all free occurrences of x and y in b become bound.
- In $decide(a; x.b; y.c)$, the x in front of the dot and all free occurrences of x in b become bound; the y in front of the dot and all free occurrences of y in c become bound.

A.2 Evaluation

Evaluation ($\geq$) is a partial function on closed terms — a term can evaluate to at most one term. The reduction algorithm is often referred to as *lazy* or *head* reduction. Terms which evaluate to themselves are called *canonical.* Being canonical is a property of a closed term's outermost form. We define evaluation first by listing the canonical terms, then by giving a list of clauses for computing with the other terms. We write $a[b/x]$ for the substitution of b for all free occurrences of x in a, implicitly renaming bound variables to avoid capture, and we write $a[b, c/x, y]$ for simultaneous substitutions.

The following terms, when closed, are canonical.

$void$	U_i	$B + C$	$(\Pi x : B)C$	$(\Sigma x : B)C$
$\{x : B|\ C\}$	$inr(a)$	$inl(a)$	$\lambda x.b$	$\langle a,\ b\rangle$

We complete the definition of $\geq$ with the following clauses.

$$\frac{a \geq \lambda x.b \quad b[c/x] \geq e}{a(c) \geq e} \qquad \frac{a \geq \langle c,\ d\rangle \quad b[c, d/x, y] \geq e}{spread(a; x, y.b) \geq e}$$

$$\frac{a \geq inl(d) \quad b[d/x] \geq e}{decide(a; x.b; y.c) \geq e} \qquad \frac{a \geq inr(d) \quad c[d/x] \geq e}{decide(a; x.b; y.c) \geq e}$$

When x does not occur free in B we sometimes abbreviate $(\Pi x : B)C$ by $B \rightarrow C$ and $(\Sigma x : B)C$ by $B \times C$.

A.3 Proof rules

The logic is given in terms of sequents, and the rules are presented in a refinement, or top-down, style. Recall that a sequent is of the form:

$$x_1 : A_1, \ldots, x_n : A_n \vdash b = b' \in B,$$

where:

1. $0 \leq n$.
2. $x_1, \ldots, x_n$ are distinct variables.
3. For $1 \leq i \leq n$, variables occurring free in A_i are among $x_1, \ldots, x_{i-1}$.
4. the variables occurring free in b, b' and B are among $x_1, \ldots, x_n$.

The A_i's are called *hypotheses*, and $x_1 : A_1, \ldots, x_n : A_n$ is called a context. Let Γ range over contexts. When b and b' are identical, $\Gamma \vdash b = b' \in B$ may be written as $\Gamma \vdash b \in B$.

A.3.1 Void

$$\Gamma \vdash \mathit{void} = \mathit{void} \in U_j$$

$$\frac{\Gamma \vdash \mathit{any}(b) = \mathit{any}(b') \in C \quad \Gamma \vdash b = b' \in \mathit{void}}{}$$

A.3.2 Union

$$\Gamma \vdash B + C = B' + C' \in U_j$$
$$\Gamma \vdash B = B' \in U_j$$
$$\Gamma \vdash C = C' \in U_j$$

$$\Gamma \vdash inl(b) = inl(b') \in B + C$$
$$\Gamma \vdash b = b' \in B$$
$$\Gamma \vdash C \in U_j$$

$$\Gamma \vdash inr(c) = inr(c') \in B + C$$
$$\Gamma \vdash c = c' \in C$$
$$\Gamma \vdash B \in U_j$$

$$\Gamma \vdash decide(d;\ x.b;\ y.c) = decide(d';\ x.b';\ y.c') \in T[d/z]$$
$$\Gamma,\ x{:}B,\ I(d, inl(x), B + C) \vdash b = b' \in T[inl(x)/z]$$
$$\Gamma,\ y{:}C,\ I(d, inr(y), B + C) \vdash c = c' \in T[inr(y)/z]$$
$$\Gamma \vdash d = d' \in B + C$$

$$\Gamma \vdash decide(inl(d);\ x.b;\ y.c) = b[d/x] \in T$$
$$\Gamma \vdash b[d/x] \in T$$

$$\Gamma \vdash decide(inr(d);\ x.b;\ y.c) = c[d/y] \in T$$
$$\Gamma \vdash c[d/y] \in T$$

A.3.3 Product

$$\Gamma \vdash (\Pi x : B)C = (\Pi x : B')C' \in U_j$$
$$\Gamma \vdash B = B' \in U_j$$
$$\Gamma, x : B \vdash C = C' \in U_j$$

$$\Gamma \vdash \lambda x.c = \lambda x.c' \in (\Pi x : B)C$$
$$\Gamma \vdash B \in U_j$$
$$\Gamma, x : B \vdash c = c' \in C$$

$$\Gamma \vdash d = d' \in (\Pi x : B)C$$
$$\Gamma, x : B \vdash d(x) = d'(x) \in C$$
$$\Gamma \vdash d \in D$$
$$\Gamma \vdash d' \in D'$$
$$\Gamma \vdash B \in U_j$$

$$\Gamma \vdash d(b) = d'(b') \in C[b/x]$$
$$\Gamma \vdash d = d' \in (\Pi x : B)C$$
$$\Gamma \vdash b = b' \in B$$

$$\Gamma \vdash \lambda x.c(a) = c[a/x] \in T$$
$$\Gamma \vdash c[a/x] \in T$$

A.3.4 Sum

$$\Gamma \vdash (\Sigma x\!:\!B)C = (\Sigma x\!:\!B')C' \in U_j$$
$$\Gamma \vdash B = B' \in U_j$$
$$\Gamma,\ x\!:\!B \vdash C = C' \in U_j$$

$$\Gamma \vdash \langle b,\ c\rangle = \langle b',\ c'\rangle \in (\Sigma x\!:\!B)C$$
$$\Gamma \vdash b = b' \in B$$
$$\Gamma \vdash c = c' \in C[b/x]$$
$$\Gamma,\ x\!:\!B \vdash C \in U_j$$

$$\Gamma \vdash \mathit{spread}(d;\ x,y.t) = \mathit{spread}(d';\ x,y.t') \in T[d/z]$$
$$\Gamma,\ x\!:\!B,\ y\!:\!C,\ I(d,\langle x,\ y\rangle,(\Sigma x\!:\!B)C) \vdash t = t' \in T[\langle x,\ y\rangle/z]$$
$$\Gamma \vdash d = d' \in (\Sigma x\!:\!B)C$$

$$\Gamma \vdash \mathit{spread}(\langle a,\ b\rangle;\ x,y.t) = t[a,b/x,y] \in T$$
$$\Gamma \vdash t[a,b/x,y] \in T$$

A.3.5 Subtype

$$\Gamma \vdash \{x{:}B|\ C\} = \{x{:}B'|\ C'\} \in U_j$$
$$\Gamma \vdash B = B' \in U_j$$
$$\Gamma,\ x{:}B \vdash C \in U_j$$
$$\Gamma,\ x{:}B \vdash C' \in U_j$$
$$\Gamma,\ x{:}B,\ y{:}C \vdash t' \in C'$$
$$\Gamma,\ x{:}B,\ y{:}C' \vdash t \in C$$

(In the previous rule, if C and C' are identical, then the last three subgoals can be omitted.)

$$\Gamma \vdash b = b' \in \{x{:}B|\ C\}$$
$$\Gamma \vdash b = b' \in B$$
$$\Gamma \vdash c \in C[b/x]$$
$$\Gamma,\ x{:}B \vdash C \in U_j$$

$$\Gamma \vdash t[b/x] = t'[b'/x] \in T[b/x]$$
$$\Gamma,\ x{:}B,\ y{:}C,\ I(x,b,B) \vdash t = t' \in T$$
$$\Gamma \vdash b = b' \in \{x{:}B|\ C\}$$
$$\Gamma,\ x{:}B \vdash C \in U_j$$

A.3.6 Equality

$$\Gamma \vdash I(a,b,B) = I(a',b',B') \in U_j$$
$$\Gamma \vdash a = a' \in B$$
$$\Gamma \vdash b = b' \in B$$

$$\Gamma \vdash \mathit{true} = \mathit{true} \in I(a,b,B)$$
$$\Gamma \vdash a = b \in B$$

$$\Gamma \vdash a = b \in B$$
$$\Gamma \vdash t \in I(a,b,B)$$

A.3.7 Universe

$$\Gamma \vdash U_i = U_i \in U_j \qquad (\text{if } i < j)$$

$$\Gamma \vdash B = B' \in U_j \qquad (\text{if } i < j)$$
$$\Gamma \vdash B = B' \in U_i$$

A.3.8 Miscellany

$$\frac{\Gamma \vdash t = t' \in T}{\Gamma \vdash t' = t \in T}$$

$$\frac{\Gamma \vdash t = t' \in T}{\Gamma \vdash t = t'' \in T \quad \Gamma \vdash t'' = t' \in T}$$

$$\frac{\Gamma \vdash t = t' \in T}{\Gamma \vdash t' = t \in T' \quad \Gamma \vdash T = T' \in U_j}$$

$$\Gamma,\ x{:}B,\ \Gamma' \vdash x \in B$$

$$\frac{\Gamma \vdash t = t' \in T[b/x]}{\Gamma \vdash t = t' \in T[b'/x] \quad \Gamma \vdash b = b' \in B \quad \Gamma,\ x{:}B \vdash T \in U_j}$$

$$\frac{\Gamma \vdash t[b/x] = t'[b'/x] \in T}{\Gamma \vdash b = b' \in B \quad \Gamma,\ x{:}B \vdash t = t' \in T}$$

$$\frac{\Gamma,\ x{:}A,\ \Gamma' \vdash b = b' \in B \quad \text{(If } x \text{ does not occur free in } \Gamma', b, b', B)}{\Gamma,\ \Gamma' \vdash b = b' \in B}$$

A.3.9 Direct computation

We complete the proof rules with rules for *direct computation*[12]. Each of the following rules has the restriction that for any closed instance-pairs C^* and D^* of C and D,

$$\exists e.\ C^* \geq e \Leftrightarrow D^* \geq e$$

(By closed instance-pairs, we mean that identical terms are substituted for the same variables in C and D.) This condition is undecidable, so in practice one would strengthen it to a decidable condition for a useful subset of it. Note that all the computation rules are instances of a direct computation rule, so they are included purely for convenience.

$$\frac{\Gamma,\ x{:}C,\ \Gamma' \vdash b = b' \in B}{\Gamma,\ x{:}D,\ \Gamma' \vdash b = b' \in B}$$

$$\frac{\Gamma \vdash C = b' \in B}{\Gamma \vdash D = b' \in B}$$

$$\frac{\Gamma \vdash b = b' \in C}{\Gamma \vdash b = b' \in D}$$

Appendix B

Definition of the Extension by Inductive Types

We give extensions to the previous appendix's definition of terms and evaluation, and give the proof rules for the simple and the parameterized versions of inductive types.

B.1 Terms

We add the following clauses to the definition of terms.

- $B \subseteq C$, $(\mu x : U_j)C$, $(\nu x : U_j)C$, $(\mu x : B \to U_j)C@b$ and $(\nu x : B \to U_j)C@b$ are terms.
- $out(b)$, $\mu\_ind(b; z, y.c)$, $\mu\_ind(a; b; z, w, y.c)$ and $\nu\_ind(b; z, y.c)$.

The variable bindings for these new terms are as follows.

- In $(\mu x : B)C$, $(\nu x : B)C$, $(\mu x : B \to U_j)C@b$ and $(\nu x : B \to U_j)C@b$, the x in front of the colon and all free occurrences of x in C become bound.
- In $\mu\_ind(b; z, y.c)$ and $\nu\_ind(b; z, y.c)$, the z and y in front of the dot and all free occurrences of z and y in c become bound. In the remaining term $\mu\_ind(a; b; z, w, y.c)$, the z, w and y in front of the dot and all free occurrences of z, w and y in c become bound.

B.2 Evaluation

The following terms, if closed, are canonical.

$$\begin{array}{lll} B \subseteq C & (\mu x : B)C & (\nu x : B)C \\ \nu\_ind(b; z, y.c) & (\mu x : B \to U_j)C@b & (\nu x : B \to U_j)C@b \end{array}$$

Finally, we may add the following clauses to the definition of $\geq$.

$$\frac{c[\lambda y.\mu\_ind(y; z, y.c), b/z, y] \geq e}{\mu\_ind(b; z, y.c) \geq e}$$

$$\frac{c[\lambda y.\nu\_ind(y; z, y.c), b/z, y] \geq e}{out(\nu\_ind(b; z, y.c)) \geq e}$$

$$\frac{c[\lambda w.\lambda y.\mu\_ind(w; y; z, w, y.c), a, b/z, w, y] \geq e}{\mu\_ind(a; b; z, w, y.c) \geq e}$$

B.3 Proof rules

B.3.1 Containment

$$\frac{\Gamma \vdash B = B' \in U_j \quad \Gamma \vdash C = C' \in U_j}{\Gamma \vdash B \subseteq C = B' \subseteq C' \in U_j}$$

$$\frac{\Gamma,\ x{:}B \vdash x \in C \quad \Gamma \vdash B \subseteq C \in U_j}{\Gamma \vdash \mathit{true} = \mathit{true} \in B \subseteq C}$$

$$\frac{\Gamma \vdash b = b' \in B \quad \Gamma \vdash t \in B \subseteq C}{\Gamma \vdash b = b' \in C}$$

B.3.2 Simple μ

For this subsection, let:

$$\begin{array}{rcl} \mu & \equiv & (\mu x{:}U_j)B \\ \mu' & \equiv & (\mu x{:}U_j)B'. \end{array}$$

$$\begin{array}{l} \Gamma \vdash \mu = \mu' \in U_j \\ \quad \Gamma,\ x{:}U_j \vdash B = B' \in U_j \\ \quad \Gamma,\ x{:}U_j,\ y{:}U_j,\ x \subseteq y \vdash t \in B \subseteq B[y/x] \end{array}$$

$$\begin{array}{l} \Gamma \vdash b = b' \in \mu \\ \quad \Gamma \vdash b = b' \in B[\mu/x] \\ \quad \Gamma \vdash \mu \in U_j \end{array}$$

$$\begin{array}{l} \Gamma \vdash \mu\_ind(b;\ z, y.d) = \mu\_ind(b';\ z, y.d') \in D[b/y] \\ \quad \Gamma,\ x{:}U_j,\ x \subseteq \mu,\ z{:}(\Pi y{:}x)D,\ y{:}B \vdash d = d' \in D \\ \quad \Gamma \vdash b = b' \in \mu \end{array}$$

$$\begin{array}{l} \Gamma \vdash \mu\_ind(b;\ z, y.d) = d[\lambda y.\mu\_ind(y;\ z, y.d), b/z, y] \in T \\ \quad \Gamma \vdash d[\lambda y.\mu\_ind(y;\ z, y.d), b/z, y] \in T \end{array}$$

B.3.3 Simple ν

For this subsection, let:

$$\begin{array}{rcl} \nu & \equiv & (\nu x{:}U_j)B \\ \nu' & \equiv & (\nu x{:}U_j)B'. \end{array}$$

$$\begin{array}{l} \Gamma \vdash \nu = \nu' \in U_j \\ \quad \Gamma,\ x{:}U_j \vdash B = B' \in U_j \\ \quad \Gamma,\ x{:}U_j,\ y{:}U_j,\ x \subseteq y \vdash t \in B \subseteq B[y/x] \end{array}$$

$$\begin{array}{l} \Gamma \vdash \mathit{out}(b) = \mathit{out}(b') \in B[\nu/x] \\ \quad \Gamma \vdash b = b' \in \nu \end{array}$$

$$\begin{array}{l} \Gamma \vdash \nu\_\mathit{ind}(c;\ z, y.d) = \nu\_\mathit{ind}(c';\ z, y.d') \in \nu@c \\ \quad \Gamma,\ x{:}C \to U_j,\ \nu \subseteq_C x,\ z{:}(\Pi y{:}C)x(y),\ y{:}C \vdash d = d' \in B \\ \quad \Gamma \vdash \nu@c = \nu@c' \in U_j \end{array}$$

$$\begin{array}{l} \Gamma \vdash \mathit{out}(\nu\_\mathit{ind}(b;\ z, y.d)) = d[\lambda y.\nu\_\mathit{ind}(y;\ z, y.d), b/z, y] \in T \\ \quad \Gamma \vdash d[\lambda y.\nu\_\mathit{ind}(y;\ z, y.d), b/z, y] \in T \end{array}$$

B.3.4 Parameterized μ

For this subsection, let:

$$\begin{array}{rcl@{\qquad}rcl}
\mu@t & \equiv & (\mu x{:}C \to U_j)B@t & \mu & \equiv & \lambda w.\mu@w \\
\mu'@t & \equiv & (\mu x{:}C' \to U_j)B'@t & \mu' & \equiv & \lambda w.\mu'@w \\
t \subseteq_C t' & \equiv & (\Pi w{:}C)t(w) \subseteq t'(w). & & &
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash \mu@c = \mu@c' \in U_j \\
\quad \Gamma,\ x{:}C \to U_j \vdash B = B' \in C \to U_j \\
\quad \Gamma,\ x{:}C \to U_j,\ y{:}C \to U_j,\ x \subseteq_C y \vdash t \in B \subseteq_C B[y/x] \\
\quad \Gamma \vdash C = C' \in U_j \\
\quad \Gamma \vdash c = c' \in C
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash \mu_ind(c, b;\ z, w, y.d) = \mu_ind(c', b';\ z, w, y.d') \in T[c, b/w, y] \\
\quad \Gamma,\ x{:}C \to U_j,\ x \subseteq_C \mu,\ z{:}(\Pi w{:}C)(\Pi y{:}x(w))T,\ w{:}C,\ y{:}B(w) \\
\qquad \vdash d = d' \in T \\
\quad \Gamma \vdash c = c' \in C \\
\quad \Gamma \vdash b = b' \in \mu@c
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash \mu_ind(c;\ z, w.d) = \mu_ind(c';\ z, w.d') \in T[c/w] \\
\quad \Gamma,\ x{:}C \to U_j,\ x \subseteq_C \mu,\ z{:}(\Pi w{:}\{w{:}C \mid x(w)\})T,\ w{:}C,\ B(w) \\
\qquad \vdash d = d' \in T \\
\quad \Gamma \vdash c = c' \in C \\
\quad \Gamma \vdash b \in \mu@c
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash \mu_ind(b; c;\ z, w, y.d) = \\
\qquad d[\lambda w.\lambda y.\mu_ind(w; y;\ z, w, y.d), b, c/z, w, y] \in T \\
\quad \Gamma \vdash d[\lambda w.\lambda y.\mu_ind(w; y;\ z, w, y.d), b, c/z, w, y] \in T
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash \mu_ind(b;\ z, w.d) = \\
\qquad d[\lambda w.\mu_ind(w;\ z, y.d), b, c/z, w, y] \in T \\
\quad \Gamma \vdash d[\lambda w.\lambda y.\mu_ind(w; y;\ z, w, y.d), b, c/z, w, y] \in T
\end{array}$$

B.3.5 Parameterized ν

For this subsection, let:

$$\begin{array}{rcl@{\qquad}rcl}
\nu@t & \equiv & (\nu x{:}C \to U_j)B@t & \nu & \equiv & \lambda w.\nu@w \\
\nu'@t & \equiv & (\nu x{:}C' \to U_j)B'@t & \nu' & \equiv & \lambda w.\nu'@w \\
t \subseteq_C t' & \equiv & (\Pi w{:}C)t(w) \subseteq t'(w). & & &
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash \nu@c = \nu@c' \in U_j \\
\quad \Gamma,\ x{:}C \to U_j \vdash B = B' \in C \to U_j \\
\quad \Gamma,\ x{:}C \to U_j,\ y{:}C \to U_j,\ x \subseteq_C y \vdash t \in B \subseteq_C B[y/x] \\
\quad \Gamma \vdash C = C' \in U_j \\
\quad \Gamma \vdash c = c' \in C
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash \nu_ind(c;\ z, y.d) = \nu_ind(c';\ z, y.d') \in \nu@c \\
\quad \Gamma,\ x{:}C \to U_j,\ \nu \subseteq_C x,\ z{:}(\Pi y{:}C)x(y),\ y{:}C \vdash d = d' \in B \\
\quad \Gamma \vdash \nu@c = \nu@c' \in U_j
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash out(b) = out(b') \in B[\nu/x](c) \\
\quad \Gamma \vdash b = b' \in \nu@c
\end{array}$$

$$\begin{array}{l}
\Gamma \vdash out(\nu_ind(b;\ z, y.d)) = d[\lambda y.\nu_ind(y;\ z, y.d), b/z, y] \in T \\
\quad \Gamma \vdash d[\lambda y.\nu_ind(y;\ z, y.d), b/z, y] \in T
\end{array}$$

B.3.6 Rules encorporating positivity

As discussed in section 5.5, we can use a syntactic condition of positivity to guarantee the monotonicity of a recursive type's body. We restate that condition here, then present the new formation rules. In each case, it is the earlier formation rule with the monotonicity subgoal omitted.

x is strongly positive *with respect to B ($SP(B,x)$) iff x does not occur free:*

1. *on the left hand side of a Π or $\subseteq$ type in B.*
2. *in a term b where $c(b)$ is a subterm of B.*
3. *in a principle argument of any other eliminations form which is a subterm of B.*

The new formation rules are as follows.

$$\Gamma \vdash (\mu x{:}U_j)B = (\mu x{:}U_j)B' \in U_j \quad (\text{if } SP(B,x))$$
$$\Gamma,\ x{:}U_j \vdash B = B' \in U_j$$

$$\Gamma \vdash (\nu x{:}U_j)B = (\nu x{:}U_j)B' \in U_j \quad (\text{if } SP(B,x))$$
$$\Gamma,\ x{:}U_j \vdash B = B' \in U_j$$

$$\Gamma \vdash (\mu x{:}C \to U_j)B@c = (\mu x{:}C' \to U_j)B'@c' \in U_j \quad (\text{if } SP(B,x))$$
$$\Gamma,\ x{:}C \to U_j \vdash B = B' \in C \to U_j$$
$$\Gamma \vdash C = C' \in U_j$$
$$\Gamma \vdash c = c' \in C$$

$$\Gamma \vdash (\nu x{:}C \to U_j)B@c = (\nu x{:}C' \to U_j)B'@c' \in U_j \quad (\text{if } SP(B,x))$$
$$\Gamma,\ x{:}C \to U_j \vdash B = B' \in C \to U_j$$
$$\Gamma \vdash C = C' \in U_j$$
$$\Gamma \vdash c = c' \in C$$

Bibliography

[1] P. Aczel. The strength of Martin-Löf's intuitionistic type theory with one universe. In S. Miettinen and J. Vaananen, editors, *Proceedings of the Symposium on Mathematical Logic*, pages 1–32, Department of Philosophy, University of Helsinki, 1977.

[2] S.F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. PhD thesis, Cornell University, 1987.

[3] R. Amadio, K.B. Bruce, and G. Longo. The finitary projection model for second order lambda calculus and solutions to higher order domain equations. In *Symposium on Logic in Computer Science*, IEEE Computer Society, 1986.

[4] E. Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, New York, 1967.

[5] V. Breazu-Tannan and A. R. Meyer. Lambda calculus with constrained types. In R. Parikh, editor, *LNCS #193: Logics of Programs*, pages 23–40, Springer-Verlag, 1985.

[6] K.B. Bruce and A.R. Meyer. The semantics of second order polymorphic lambda-calculus. In *Symposium on Semantics of Data Types, LNCS 173*, Springer-Verlag, 1984.

[7] N.G. deBruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In *Symposium on Automatic Demonstration*, pages 29–61, Springer-Verlag, 1970.

[8] N.G. deBruijn. A survey of the project AUTOMATH. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays in Combinatory*

Logic, Lambda Calculus and Formalism, pages 589–606, Academic Press, New York, 1980.

[9] W.R. Cleaveland. *Type-Theoretic Models of Concurrency.* PhD thesis, Cornell University, 1987.

[10] R.L. Constable. Constructive mathematics and automatic program writers. In *Proceedings of IFIP Congress*, pages 229–233, Ljublyana, 1971.

[11] R.L. Constable. Programs as proofs. *Information Processing Letters*, 16(3):105–112, 1983.

[12] R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[13] R.L. Constable and C.D. Johnson, S.D. Eichenlaub. Introduction to the PL/CV2 programming logic. In *Logics of Programs, LNCS 135*, Springer-Verlag, 1982.

[14] R.L. Constable and D.R. Zlatin. The type theory of PL/CV3. *ACM Transactions on Programming Languages and Systems*, 7(1):72–93, 1984.

[15] M. Coppo and M. Zacchi. Type inference and logical relations. In *Symposium on Logic in Computer Science*, IEEE Computer Society, 1986.

[16] T. Coquand and G. Huet. Constructions: a higher-order proof system for mechanizing mathematics. In *Proceedings of EUROCAL '85, LNCS 203*, pages 35–49, Springer-Verlag, 1985.

[17] H.B. Curry, R. Feys, and W. Craig. *Combinatory Logic.* Volume 1, North-Holland, Amsterdam, 1958.

[18] M. Dummett. *Elements of Intuitionism. Oxford Lecture Series*, Claredon Press, Oxford, 1977.

[19] S. Fortune, D. Leivant, and M. O'Donnell. The expressiveness of simple and second-order type structures. *Journal of the Association for Computing Machinery*, 30(1):151–185, January 1983.

[20] G. Gentzen. Investigations into logical deduction. In M.E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, North-Holland, Amsterdam, 1969.

[21] J.-Y. Girard. *Interprétation fonctionelle et elimination des coupures dans l'arithmetique d'order superieur.* PhD thesis, Paris, 1972.

[22] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92, North-Holland, 1971.

[23] M. Gordon, A. Milner, and C. Wadsworth. Edinburgh LCF: a mechanized logic of computation. In *Logics of Programs, LNCS 78*, Springer-Verlag, 1979.

[24] R. Harper. 1987. Private communication.

[25] R. Harper. A model construction for Martin-Löf's type theory. 1987. Unpublished manuscript.

[26] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings of the Symposium on Logic in Computer Science*, pages 194–204, IEEE, 1987.

[27] S. Hayashi and H. Nakano. *PX, a Computational Logic.* Technical Report RIMS-573, Research Institute for Mathematical Sciences, Kyoto University, 1987.

[28] J. van Heijenoort. *From Frege to Gödel: A Sourcebook in Mathematical Logic.* Harvard University Press, Cambridge, MA, 1967.

[29] W. Howard. The formulae-as-types notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490, Academic Press, New York, 1980.

[30] D.J. Howe. The computational behavior of Girard's paradox. In *Proceedings of the Symposium on Logic in Computer Science*, pages 205–214, IEEE, 1987.

[31] T.B. Knoblock. *Metamathematical Extensibility in Type Theory.* PhD thesis, Cornell University, 1987.

[32] P. Martin-Löf. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, North-Holland, Amsterdam, 1982.

[33] P. Martin-Löf. An intuitionistic theory of types: predicative part. In E.H. Rose and J.C. Sheperdson, editors, *Logic Colloquium '73*, pages 73–118, North-Holland, Amsterdam, 1973.

[34] D. Prawitz. *Natural Deduction.* Almqvist and Wiksell, Stockholm, 1965.

[35] J. Reynolds. Towards a theory of type structures. In *Colloque sur la Programmation, LNCS 19*, pages 403–425, Springer-Verlag, 1974.

[36] B. Russell and A.N. Whitehead. *Principia Mathematica.* Volume 1, Cambridge University Press, Cambridge, MA, 1925.

[37] D. Scott. Constructive validity. In *Symposium on Automatic Demonstration*, pages 237–275, Springer-Verlag, 1970.

[38] S.F. Smith and R.L. Constable. Partial objects in constructive type theory. In *Proceedings of the Symposium on Logic in Computer Science*, pages 183–193, IEEE, 1987.

[39] S. Stenlund. *Combinators, λ-terms and Proof Theory.* D. Reidel, Dordrecht, 1972.

[40] W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32:198–212, 1967.

[41] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 1(5):285–309, 1955.