Intuitionistic Type Theory

Peter Dybjer* Erik Palmgren†

May 6, 2015

Abstract

Intuitionistic Type Theory (also Constructive Type Theory or Martin-Löf Type Theory) is a formal logical system and philosophical foundation for *constructive mathematics* (link). It is a full-scale system which aims to play a similar role for constructive mathematics as *Zermelo-Fraenkel Set Theory* (link) does for classical mathematics. It is based on the propositions-as-types principle and clarifies the Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic. It extends this interpretation to the more general setting of Intuitionistic Type Theory and thus provides a general conception not only of what a constructive proof is, but also of what a constructive mathematical object is. The main idea is that mathematical concepts such as elements, sets and functions are explained in terms of concepts from programming such as data structures, data types and programs. This article describes the formal system of Intuitionistic Type Theory and its semantic foundations.

*peterd@chalmers.se
†palmgren@math.su.se

---

1 Overview

We begin with a bird's eye view of some important aspects of Intuitionistic Type Theory. Readers who are unfamiliar with the theory may prefer to skip it on a first reading.

The origins of Intuitionistic Type Theory are Brouwer's Intuitionism and Russell's Type Theory. Like Church's classical Simple Theory of Types (link) it is based on the lambda calculus with types, but differs from it in that it is based on the propositions-as-types principle, discovered by Curry (1958) for propositional logic and extended to predicate logic by Howard (1980) and de Bruijn (1970). This extension was made possible by the introduction of indexed families of types (dependent types) for

representing the predicates of predicate logic. In this way all logical connectives and quantifiers can be interpreted by type formers. In Intuitionistic Type Theory further types are added, such as a type of natural numbers, a type of small types (a universe) and a type of well-founded trees. The resulting theory contains intuitionistic number theory (Heyting Arithmetic) and much more.

The theory is formulated in natural deduction where the rules for each type former are classified as formation, introduction, elimination, and equality rules. These rules exhibit certain symmerties between the introduction and elimination rules following Gentzen's and Prawitz' treatment of natural deduction, as explained in the article on *proof-theoretic semantics* (link).

The elements of propositions, when interpreted as types, are called *proof-objects*. When proof-objects are added to the natural deduction calculus it becomes a typed lambda calculus with dependent types, which extends Church's original typed lambda calculus. The equality rules are computation rules for the terms of this calculus. Each function definable in the theory is total and computable. Intuitionistic Type Theory is thus a typed functional programming language with the unusual property that all programs terminate.

Intuitionistic Type Theory is not only a formal logical system but also provides a comprehensive philosophical framework for intuitionism. It is an *interpreted language*, where the distinction between the *demonstration of a judgment* and the *proof of a proposition* plays a fundamental role (Sundholm 2012). The framework clarifies the Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic and extends it to the more general setting of Intuitionistic Type Theory. In doing so it provides a general conception not only of what a constructive proof is, but also of what a constructive mathematical object is. The meaning of the judgments of Intuitionistic Type Theory is explained in terms of computations of the canonical forms of types and terms. These informal, intuitive meaning explanations are "pre-mathematical" and should be contrasted to formal mathematical models developed inside a standard mathematical framework such as set theory.

This meaning theory also justifies a variety of inductive, recursive, and inductive-recursive definitions. Although proof-theoretically strong notions can be justified, such as analogues of certain large cardinals, the system is considered predicative. Impredicative definitions of the kind found in higher-order logic, intuitionistic set theory, and topos theory are not part of the theory. Neither is Markov's principle, and thus the theory is distinct from Russian constructivism.

An alternative formal logical system for predicative constructive mathematics is Myhill and Aczel's *Constructive Zermelo-Fraenkel Set Theory (CZF)* (link). This theory, which is based on intuitionistic first-order predicate logic and omits some of the axioms of classical Zermelo-Fraenkel Set Theory, has a natural interpretation in Intuitionistic Type Theory. Martin-Löf's meaning explanations thus also indirectly form a basis for CZF.

Variants of Intuitionistic Type Theory underlie several widely used proof assistants, including NuPRL, Coq, and Agda. These proof assistants are computer systems that have been used for formalizing large and complex proofs of mathematical theorems, such as the Four Colour Theorem in graph theory and the Feit-Thompson Theorem in finite group theory. They have also been used to prove the correctness of computer software.

Philosophically and practically, Intuitionistic Type Theory is a foundational framework where constructive mathematics and computer programming are, in a deep sense, the same. This point has been emphasized by Gonthier (2008) in the paper in which he describes his proof of the Four Colour Theorem:

> The approach that proved successful for this proof was to turn almost every mathematical concept into a data structure or a program in the Coq system, thereby converting the entire enterprise into one of program verification.

2 Propositions as types

Intuitionistic Type Theory - a new way of looking at logic?

Intuitionistic Type Theory offers a new way of analyzing logic, mainly through its introduction of explicit proof objects. This provides a direct computational interpretation of logic, since there are computation rules for proof objects. As regards expressive power, Intuitionistic Type Theory may be considered as an extension of first-order logic, much as higher order logic, but predicative.

A Type Theory. Russell developed Type Theory (link) in response to his discovery of a paradox in naive set theory. In his Ramified Type Theory mathematical objects are classified according to their *types*: the type of propositions, the type of objects, the type of properties of objects, etc. When Church developed his Simple Theory of Types (link) on the basis of a typed version of his lambda calculus he added the rule that there is a type of functions between any two types of the theory. Intuitionistic Type Theory further extends the simply typed lambda calculus with dependent types, that is, indexed families of types. An example is the family of types of n-tuples indexed by n.

Types have been widely used in programming for a long time. Early high-level programming languages introduced types of integers and floating point numbers. Modern programming languages often have rich type systems with many constructs for forming new types. Intuitionistic Type Theory is a functional programming language where the type system is so rich that practically any conceivable property of a program can be expressed as a type. Types can thus be used as specifications of the task of a program.

An intuitionstic logic with proof-objects. Brouwer's analysis of logic led him to an intuitionistic logic which rejects the law of excluded middle and the law of double negation. These laws are not valid in Intuitionistic Type Theory. Thus it does not contain classical (Peano) arithmetic but only intuitionistic (Heyting) arithmetic. (It is another matter that Peano arithmetic can be interpreted in Heyting arithmetic by the double negation interpretation, link to article about *Intuitionistic Logic*)

Consider a theorem of intuitionistic arithmetic, such as the division theorem

$$\forall m, n.m > 0 \supset \exists q, r.mq + r = n$$

A formal proof (in the usual sense) of this theorem is a sequence (or tree) of formulas, where the last (root) formula is the theorem and each formula in the sequence is either an axiom (a leaf) or the result of applying an inference rule to some earlier (higher) formulas.

When the division theorem is proved in Intuitionistic Type Theory, we do not only build a formal proof in the usual sense but also a *construction* (or *proof-object*) "div" which witnesses the truth of the theorem. We write

$$\mathrm{div} : \forall m, n : \mathbf{N}.m > 0 \supset \exists q, r : \mathbf{N}.mq + r = n$$

to express that div is a proof-object for the division theorem, that is, an element of the type representing the division theorem. When propositions are represented as types, the $\forall$-quantifier is identified with the dependent function space former (or general cartesian product) Π, the $\exists$-quantifier with the dependent pairs type former (or general disjoint sum) Σ, the identity relation $=$ with the type former $\mathbf{I}$ of proof-objects of identities, and the greater than relation $>$ with the type former $\mathbf{GT}$ of proof-objects of greater-than statements. Using "type-notation" we thus write

$$\mathrm{div} : \Pi m, n : \mathbf{N}.\mathbf{GT}(m, 0) \to \Sigma q, r : \mathbf{N}.\mathbf{I}(\mathbf{N}, mq + r, n)$$

to express that the proof object "div" is a function which maps two numbers m and n and a proof p that $m > 0$ to a triple $(q, (r, s))$, where q is the quotient and r is the remainder obtained when dividing n by m. The third component s is a proof-object witnessing the fact that $mq + r = n$.

Crucially, div is not only a function in the classical sense; it is also a function in the intuitionistic sense, that is, a program which computes the output $(q, (r, s))$ when given m, n, p as inputs. This program is a term in a lambda calculus with special constants, that is, a program in a functional programming language.

Type theory as an extension of first-order predicate logic. Intuitionistic Type Theory can be considered as an extension of first-order logic, much as higher order logic is an extension of first order logic. In higher order logic we find some individual domains which can be interpreted as any sets we like. If there are relational constants in the signature these can be interpreted as any relations between the sets interpreting the individual domains. On top of that we can quantify over relations, and over relations of relations, etc. We can think of higher order logic as first-order logic equipped with a way of introducing new domains of quantification: if $S_1, \ldots, S_n$ are domains of quantification then $(S_1, \ldots, S_n)$ is a new domain of quantification consisting of all the n-ary relations between the domains $S_1, \ldots, S_n$. Higher order logic has a straightforward set-theoretic interpretation where $(S_1, \ldots, S_n)$ is interpreted as the power set $P(A_1 \times \cdots \times A_n)$ where A_i is the interpretation of S_i, for $i = 1, \ldots, n$. This is the kind of higher order logic or simple theory of types that Ramsey, Church and others introduced.

Intuitionistic Type Theory can be viewed in a similar way, only here the possibilities for introducing domains of quantification are richer, one can use $\Sigma, \Pi, +, \mathbf{I}$ to construct new ones from old. (Section 3.1, Martin-Löf 1972). Intuitionistic Type Theory has a straightforward set-theoretic interpretation as well, where Σ, Π etc are interpreted as the corresponding set-theoretic constructions; see below. We can add to Intuitionistic Type Theory unspecified individual domains just as in HOL. These are interpreted as sets as for HOL. Now we exhibit a difference from HOL: in Intuitionistic Type Theory we can introduce unspecified family symbols. We can introduce T as a family of types over the individual domain S:

$$T(x) \text{ type } (x : S).$$

If S is interpreted as A, T can be interpreted as any family of sets indexed by A. As a non-mathematical example, we can render the binary relation *loves* between member of an individual domain of *people* as follows. Introduce the binary family Loves over the domain People

$$\text{Loves}(x, y) \text{ type } (x : \text{People}, y : \text{People}).$$

The interpretation can be any family of sets $B_{x,y}$ ($x : A$, $y : A$). How does this cover the standard notion of relation? Suppose we have a binary relation R on A in the familiar set-theoretic sense. We can make a binary family corresponding to this as follows

$B_{x,y} = \{0\}$ if $R(x, y)$ holds

$B_{x,y} = \emptyset$ if $R(x, y)$ is false.

Now clearly $B_{x,y}$ is nonempty if and only if $R(x, y)$ holds. (We could have chosen any other element from our set theoretic universe than 0 to indicate truth.) Thus from any relation we can construct a family whose truth of x, y is equivalent to $B_{x,y}$ being non-empty. Note that this interpretation does not care what the proof for $R(x, y)$ is, just that it holds. Recall that Intuitionistic Type Theory interprets propositions as types, so $p : \text{Loves}(\text{John}, \text{Mary})$ means that $\text{Loves}(\text{John}, \text{Mary})$ is true.

The interpretation of relations as families allows for keeping track of proofs or evidence that $R(x, y)$ holds, but we may also chose to ignore it.

In Montague semantics (link) higher order logic is used to give semantics of natural language (and examples as above). Ranta (1994) introduced the idea to instead employ Intuitionistic Type Theory to better capture sentence structure with the help of dependent types.

In contrast, how would the mathematical relation $>$ between natural numbers be handled in Intuitionistic Type Theory? First of all we need a type of numbers $\mathbf{N}$. We could in principle introduce an

unspecified individual domain $\mathbf{N}$, and then add axioms just as we do in first-order logic when we set up the axiom system for Peano arithmetic. However this would not give us the desirable computational interpretation. So as explained below we lay down introduction rules for constructing new natural numbers in $\mathbf{N}$ and elimination and computation rules for defining functions on $\mathbf{N}$ (by recursion). The standard order relation $>$ should satisfy

$$x > y \text{ iff there } z : \mathbf{N} \text{ such that } y + z + 1 = x.$$

The right hand is rendered as $\Sigma z : \mathbf{N}.\mathbf{I}(\mathbf{N}, y + z + 1, x)$ in Intuitionistic Type Theory, and we take this as definition of relation $>$. ($+$ is defined by recursive equations, $\mathbf{I}$ is the identity type construction). Now all the properties of $>$ are determined by the mentioned introduction and elimination and computation rules for $\mathbf{N}$.

A logic with several forms of judgment. The type system of Intuitionistic Type Theory is very expressive. As a consequence the well-formedness of a type is no longer a simple matter of parsing, it is something which needs to be proved. Well-formedness of a type is one form of *judgment* of intuitionistic type theory. Well-typedness of a term with respect to a type is another. Furthermore, there are equality judgments for types and terms. This is yet another way in which Intuitionistic Type Theory differs from ordinary first order logic with its focus on the sole judgment expressing the truth of a proposition.

Semantics. While a standard presentation of first-order logic would follow Tarski in defining the notion of model, Intuitionistic Type Theory follows the tradition of Brouwerian meaning theory as further developed by Heyting and Kolmogorov, the so called BHK-interpretation of logic. The key point is that the proof of an implication $A \supset B$ is a *method* that transforms a proof of A to a proof of B. In Intuitionistic Type Theory this method is formally represented by the program $f : A \supset B$ or $f : A \to B$: the type of proofs of an implication $A \supset B$ is the type of functions which maps proofs of A to proofs of B.

Moreover, whereas Tarski semantics is usually presented meta-mathematically, and assumes set theory, Martin-Löf's meaning theory of Intuitionistic Type Theory should be understood directly and "pre-mathematically", that is, without assuming a meta-language such as set theory.

A functional programming language. Readers with a background in lambda calculus and functional programming can get an alternative first approximation of Intuitionistic Type Theory by thinking about it as a typed functional programming language in the style of Haskell or one of the dialects of ML. However, it differs from these in two crucial aspects: (i) it has dependent types (see below) and (ii) all typable programs terminate. (Note that Intuitionistic Type Theory has influenced recent extensions of Haskell with *generalized algebraic datatypes* which sometimes can play a similar role as inductively defined dependent types.)

2.1 The Curry-Howard correspondence

As already mentioned, the principle that

a proposition is the type of its proofs.

is fundamental to Intuitionistic Type Theory. This principle is also known as the Curry-Howard correspondence or even Curry-Howard isomorphism. Curry discovered a correspondence between the implicational fragment of intuitionistic logic and the simply typed lambda-calculus. Howard extended this correspondence to first-order predicate logic. In Intuitionistic Type Theory this correspondence

becomes an *identification* of proposition and types, which has been extended to include quantification over higher types and more.

2.2 Sets of proof-objects

So what are these proof-objects like? They should not be thought of as logical derivations, but rather as some (structured) symbolic evidence that something is true. Another term for such evidence is "truth-maker".

It is instructive, as a somewhat crude first approximation, to replace types by ordinary sets in this correspondence. Define a set $E_{m,n}$, depending on $m, n \in \mathbb{N}$, by:

$$E_{m,n} = \begin{cases} \{0\} & \text{if } m = n \\ \\ \emptyset & \text{if } m \neq n. \end{cases}$$

Then $E_{m,n}$ is nonempty exactly when $m = n$. The set $E_{m,n}$ corresponds to the proposition $m = n$, and the number 0 is a proof-object (truth-maker) inhabiting the sets $E_{m,m}$.

Consider the proposition that m *is an even number* expressed as the formula $\exists n \in \mathbb{N}.m = 2n$. We can build a set of proof-objects corresponding to this formula by using the general set-theoretic sum operation. Suppose that A_n $(n \in \mathbb{N})$ is a family of sets. Then its disjoint sum is given by the set of pairs

$$(\Sigma n \in \mathbb{N})A_n = \{(n, a) : n \in \mathbb{N}, a \in A_n\}.$$

If we apply this construction to the family $A_n = E_{m,2n}$ we see that $(\Sigma n \in \mathbb{N})E_{m,2n}$ is nonempty exactly when there is an $n \in \mathbb{N}$ with $m = 2n$. Using the general set-theoretic product operation $(\Pi n \in \mathbb{N})A_n$ we can similarly obtain a set corresponding to a universally quantified proposition.

2.3 Dependent types

In Intuitionistic Type Theory there are primitive type formers Σ and Π for general sums and products, and I for identity types, analogous to the set-theoretic constructions described above. The *identity type* $I(N, m, n)$ corresponding to the set $E_{m,n}$ is an example of a *dependent type* since it depends on m and n. It is also called an *indexed family of types* since it is a family of types indexed by m and n. Similarly, we can form the general disjoint sum $\Sigma x : A.B$ and the general cartesian product $\Pi x : A.B$ of such a family of types B indexed by $x : A$, corresponding to the set theoretic sum and product operations above.

Dependent types can also be defined by primitive recursion. An example is the type of n-tuples A^n of elements of type A and indexed by $n : N$ defined by the equations

$$\begin{aligned} A^0 &= 1 \\ A^{n+1} &= A \times A^n \end{aligned}$$

where 1 is a one element type and $\times$ denotes the cartesian product of two types. We note that dependent types introduce computation in types: the defining rules above are computation rules. For example, the result of computing A^3 is $A \times (A \times (A \times 1))$.

2.4 Propositions as types in Intuitionistic Type Theory

With propositions as types, predicates become dependent types. For example, the predicate Prime(x) becomes the type of proofs that x is prime. This type *depends* on x. Similarly, $x < y$ is the type of proofs that x is less than y.

According to the Curry-Howard interpretation of propositions as types

$$\begin{aligned} \bot &= \emptyset \\ \top &= 1 \\ A \vee B &= A + B \\ A \wedge B &= A \times B \\ A \supset B &= A \to B \\ \exists x : A.B &= \Sigma x : A.B \\ \forall x : A.B &= \Pi x : A.B \end{aligned}$$

where $\Sigma x : A.B$ is the disjoint sum of the A-indexed family of types B and $\Pi x : A.B$ is its cartesian product. The canonical elements of $\Sigma x : A.B$ are pairs (a, b) such that $a : A$ and $b : B[x := a]$ (the type obtained by substituting all free occurrences of x in B by a). The elements of $\Pi x : A.B$ are (computable) functions f such that $f\, a : B[x := a]$, whenever $a : A$.

For example, consider the proposition

$$\forall m : N.\exists n : N.m < n \wedge \mathrm{Prime}(n) \tag{1}$$

expressing that there are arbitrarily large primes. Under the Curry-Howard interpretation this becomes the type $\Pi m : N.\Sigma n : N.m < n \times \mathrm{Prime}(n)$ of functions which map a number m to a triple $(n, (p, q))$, where n is a number, p is a proof that $m < n$ and q is a proof that n is prime. This is the *proofs as programs* principle: a constructive proof that there are arbitrarily large primes becomes a program which given any number produces a larger prime together with proofs that it indeed is larger and indeed is prime.

Note that the proof which derives a contradiction from the assumption that there is a largest prime is not constructive, since it does not explicitly give a way to compute an even larger prime. To turn this proof into a constructive one we have to show explicitly how to construct the larger prime. (Since proposition (1) above is a Π^0_2-formula we can for example use Friedman's A-translation to turn such a proof in classical arithmetic into a proof in intuitionistic arithmetic and thus into a proof in Intuitionistic Type Theory.)

3 Basic Intuitionistic Type Theory

We first present a core version of Intuitionistic Type Theory, closely related to the first version of the theory presented by Martin-Löf in 1972 (Martin-Löf 1998). In addition to the type formers needed for the Curry-Howard interpretation of typed intuitionistic predicate logic listed above, we have two types: the type N of natural numbers and the type U of small types.

The resulting theory can be shown to contain intuitionistic number theory **HA** (Heyting arithmetic), Gödel's System **T** of primitive recursive functions of higher type, and the theory $\mathbf{HA}^\omega$ of Heyting arithmetic of higher type.

This core Intuitionistic Type Theory is not only the original one, but perhaps the minimal version which exhibits the essential features of the theory. Later extensions with primitive identity types, well-founded tree types, universe hierarchies, and general notions of inductive and inductive-recursive definitions have increased the proof-theoretic strength of the theory and also made it more convenient for programming and formalization of mathematics. For example, with the addition of well-founded trees we can interpret the Constructive Zermelo-Fraenkel Set Theory **CZF** of Aczel (1977). However, we will wait until the next section to describe those extensions.

3.1 Judgments

In Martin-Löf (1996) a general philosophy of logic is presented where the traditional notion of judgment is expanded and given a central position. A judgment is no longer just an affirmation or denial of a proposition, but a general act of knowledge. When reasoning mathematically we make judgments about mathematical objects. One form of judgment is to state that some mathematical statement is true. Another form of judgment is to state that something is a mathematical object, for example a set. The logical rules give methods for producing correct judgments from earlier judgments. The judgments obtained by such rules can be presented in tree form

$$\dfrac{\dfrac{J_1 \quad J_2}{J_3}\ r_1 \qquad \dfrac{\dfrac{J_4}{J_5}\ r_5 \qquad J_6}{J_7}\ r_3}{J_8}\ r_4$$

or in sequential form

(1) J_1 axiom
(2) J_2 axiom
(3) J_3 by rule r_1 from (1) and (2)
(4) J_4 axiom
(5) J_5 by rule r_2 from (4)
(6) J_6 axiom
(7) J_7 by rule r_3 from(5) and (6)
(8) J_8 by rule r_4 from (3) and (7)

The latter form is common in mathematical arguments. Such a sequence or tree formed by logical rules from axioms is a *derivation* or *demonstration* of a judgment.

First-order reasoning may be presented using a single kind of judgment:

> the proposition B is true under the hypothesis that the propositions $A_1, \ldots, A_n$ are all true.

We write this *hypothetical judgment* as a so-called *Gentzen sequent*

$$A_1, \ldots, A_n \vdash B.$$

Note that this is a single judgment that should not be confused with the derivation of the judgment $\vdash B$ from the judgments $\vdash A_1, \ldots, \vdash A_n$. When $n = 0$, then the *categorical judgment* $\vdash B$ states that B is true without any assumptions. With sequent notation the familiar rule for conjunctive introduction becomes

$$\dfrac{A_1, \ldots, A_n \vdash B \quad A_1, \ldots, A_n \vdash C}{A_1, \ldots, A_n \vdash B \wedge C}\ (\wedge I).$$

3.2 Judgment forms

Martin-Löf type theory has four basic forms of judgments and is a considerably more complicated system than first-order logic. One reason is that more information is carried around in the derivations due to the identification of propositions and types. Another reason is that the syntax is more involved. For instance, the well-formed formulas (types) have to be generated simultaneously with the provably true formulas (inhabited types).

The four forms of *categorical* judgment are

- $\vdash A$ type, meaning that A is a well-formed type,

- $\vdash a : A$, meaning that a has type A,
- $\vdash A = A'$, meaning that A and A' are equal types,
- $\vdash a = a' : A$, meaning that a and a' are equal elements of type A.

In general, a judgment is *hypothetical*, that is, it is made in a context Γ, that is, a list $x_1 : A_1, \ldots, x_n : A_n$ of variables which may occur free in the judgment together with their respective types. Note that the types in a context can depend on variables of earlier types. For example, A_n can depend on $x_1 : A_1, \ldots, x_{n-1} : A_{n-1}$. The four forms of hypothetical judgments are

- $\Gamma \vdash A$ type, meaning that A is a well-formed type in the context Γ,
- $\Gamma \vdash a : A$, meaning that a has type A in context Γ,
- $\Gamma \vdash A = A'$, meaning that A and A' are equal types in the context Γ,
- $\Gamma \vdash a = a' : A$, meaning that a and a' are equal elements of type A in the context Γ.

Under the proposition as types interpretation

$$\vdash a : A \tag{2}$$

can be understood as the judgment that a is a proof-object for the proposition A. When suppressing this object we get a judgment corresponding to the one in ordinary first-order logic (see above):

$$\vdash A \text{ true}. \tag{3}$$

Remark 3.1. Martin-Löf (1994) argues that Kant's *analytic judgment a priori* and *synthetic judgment a priori* can be exemplified, in the realm of logic, by (2) and (3) respectively. In the analytic judgment (2) everything that is needed to make the judgment evident is explicit. For its synthetic version (3) a possibly complicated proof construction a needs to be provided to make it evident. This understanding of analyticity and syntheticity has the surprising consequence that "the logical laws in their usual formulation are all synthetic." (op. cit., p. 95). His analysis further gives: " [...] the logic of analytic judgments, that is, the logic for deriving judgments of the two analytic forms, is complete and decidable, whereas the logic of synthetic judgments is incomplete and undecidable, as was shown by Gödel." (op. cit., p. 97). The decidability of the two analytic judgments ($\vdash a : A$ and $\vdash a = b : A$) hinges on the metamathematical properties of type theory: strong normalization and decidable type checking.

Sometimes also the following forms are explicitly considered to be judgments of the theory:

- Γ context, meaning that Γ is a well-formed context.
- $\Gamma = \Gamma'$, meaning that Γ and Γ' are equal contexts.

Below we shall abbreviate the judgment $\Gamma \vdash A$ type as $\Gamma \vdash A$ and Γ context as $\Gamma \vdash$.

3.3 Inference rules

When stating the rules we will use the letter Γ as a meta-variable ranging over contexts, $A, B, \ldots$ as meta-variables ranging over types, and $a, b, c, d, e, f, \ldots$ as meta-variables ranging over terms.

The first group of inference rules are general rules including rules of assumption, substitution, and context formation. There are also rules which express that equalities are equivalence relations. There

are numerous such rules, and we only show the particularly important *rule of type equality* which is crucial for computation in types:

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A = B}{\Gamma \vdash a : B}$$

The remaining rules are specific to the type formers. These are classified as formation, introduction, elimination, and equality rules.

3.4 Intuitionistic predicate logic

We only give the rules for Π. There are analogous rules for the other type formers corresponding to the logical constants of typed predicate logic.

In the following $B[x := a]$ means the term obtained by substituting the term a for each free occurrence of the variable x in B (avoiding variable capture).

Π-formation.

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi x : A.B}$$

Π-introduction.

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x.b : \Pi x : A.B}$$

Π-elimination.

$$\frac{\Gamma \vdash f : \Pi x : A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\, a : B[x := a]}$$

Π-equality.

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x.b)\, a = b[x := a] : B[x := a]}$$

This is the rule of β-conversion. We may also add the rule of η-conversion:

$$\frac{\Gamma \vdash f : \Pi x : A.B}{\Gamma \vdash \lambda x.f\, x = f : \Pi x : A.B}.$$

Furthermore, there are congruence rules expressing that operations introduced by the formation, introduction, and elimination rules preserve equality. For example, the congruence rule for Π is

$$\frac{\Gamma \vdash A = A' \qquad \Gamma, x : A \vdash B = B'}{\Gamma \vdash \Pi x : A.B = \Pi x : A'.B'}.$$

3.5 Natural numbers

As in Peano arithmetic the natural numbers are generated by 0 and the successor operation s. The elimination rule states that these are the only possible ways to generate a natural number.

We write $f(c) = R(c, d, xy.e)$ for the function which is defined by primitive recursion on the natural number c with base case d and step function $xy.e$ (or alternatively $\lambda xy.e$) which maps the value y for the previous number $x : N$ to the value for $s(x)$. Note that R is a new variable-binding operator: the variables x and y become bound in e.

N-formation.

$$\Gamma \vdash N$$

N-introduction.

$$\Gamma \vdash 0 : N \qquad \frac{\Gamma \vdash a : N}{\Gamma \vdash s(a) : N}$$

N-elimination.

$$\frac{\Gamma, x : N \vdash C \quad \Gamma \vdash c : N \quad \Gamma \vdash d : C[x := 0] \quad \Gamma, y : N, z : C[x := y] \vdash e : C[x := s(y)]}{\Gamma \vdash R(c, d, yz.e) : C[x := c]}$$

N-equality (under appropriate premises)

$$\begin{aligned} R(0, d, yz.e) &= d : C[x := 0] \\ R(s(a), d, yz.e) &= e[y := a, z := R(a, d, yz.e)] : C[x := s(a)] \end{aligned}$$

The rule of N-elimination simultaneously expresses the type of a function defined by primitive recursion and, under the Curry-Howard interpretation, the rule of mathematical induction: we prove the property C of a natural number x by induction on x.

Gödel's System T is essentially Intuitionistic Type Theory with only the type formers N and $A \to B$ (the type of functions from A to B, which is the special case of $(\Pi x : A)B$ where B does not depend on $x : A$). Since there are no dependent types in System T the rules can be simplified.

3.6 The universe of small types

Martin-Löf's first version of type theory (Martin-Löf 1971) had an axiom stating that there is a type of all types. This was proved inconsistent by Girard who found that the Burali-Forti paradox could be encoded in this theory.

To overcome this pathological impredicativity, but still retain some of its expressivity, Martin-Löf introduced in 1972 a universe U of small types closed under all type formers of the theory, except that it does not contain itself (Martin-Löf 1998). The rules are:

U-formation.

$$\Gamma \vdash U$$

U-introduction.

$$\frac{\Gamma \vdash \emptyset}{}$$

$$\frac{\Gamma \vdash 1}{}$$

$$\frac{\Gamma \vdash A : U \qquad \Gamma \vdash B : U}{\Gamma \vdash A + B : U}$$

$$\frac{\Gamma \vdash A : U \qquad \Gamma \vdash B : U}{\Gamma \vdash A \times B : U}$$

$$\frac{\Gamma \vdash A : U \qquad \Gamma \vdash B : U}{\Gamma \vdash A \to B : U}$$

$$\frac{\Gamma \vdash A : U \qquad \Gamma, x : A \vdash B : U}{\Gamma \vdash \Sigma x : A.B : U}$$

$$\frac{\Gamma \vdash A : U \qquad \Gamma, x : A \vdash B : U}{\Gamma \vdash \Pi x : A.B : U}$$

$$\Gamma \vdash N : U$$

U-elimination.

$$\frac{\Gamma \vdash A : U}{\Gamma \vdash A}$$

Since U is a type, we can use N-elimination to define small types by primitive recursion. For example, if $A : U$, we can define the type of n-tuples of elements in A as follows:

$$A^n = R(n, 1, xy.A \times y) : U$$

This type-theoretic universe U is analogous to a Grothendieck universe in set theory which is a set of sets closed under all the ways sets can be constructed in Zermelo-Fraenkel Set Theory. The existence of a Grothendieck universe cannot be proved from the usual axioms of Zermelo-Fraenkel Set Theory but needs a new axiom.

In Martin-Löf (1975) the universe is extended to a countable hierarchy of universes

$$U_0 : U_1 : U_2 : \cdots .$$

In this way each type has a type, not only each small type.

3.7 Propositional identity

Above, we introduced the equality judgment

$$\Gamma \vdash a = a' : A. \tag{4}$$

This is usually called a "definitional equality" because it can be decided by normalizing the terms a and a' and checking whether the normal forms are identical. However, (4) is not a proposition (type) and we thus cannot prove such equalities by induction. For this reason we need to introduce propositional identity types. For example, the identity type for natural numbers $I(N, m, n)$ can be defined by U-valued primitive recursion. We can then express and prove the Peano axioms. Moreover, extensional equality of ufnctions can be defined by

$$I(N \to N, f, f') = \Pi x : N.I(N, f\, x, f'\, x).$$

3.8 The axiom of choice is a theorem

The following form of the axiom of choice is an immediate consequence of the BHK-interpretation of the intuitionistic quantifiers, and is easily proved in Intuitionistic Type Theory:

$$(\Pi x : A.\Sigma y : B.C) \to \Sigma f : (\Pi x : A.B).C[y := f\, x]$$

The reason is that $\Pi x : A.\Sigma y : B.C$ is the type of functions which map elements $x : A$ to pairs (y, z) with $y : B$ and $z : C$. The choice function f is obtained by returning the first component $y : B$ of this pair.

It is perhaps surprising that Intuitionistic Type Theory directly validates an axiom of choice, since this axiom is often considered problematic from a constructive point of view. A possible explanation for this state of affairs is that the above is an axiom of choice for *types*, and that types are not in general appropriate constructive approximations of sets in the classical sense. For example, we can represent a real number as a Cauchy sequence in Intuitionistic Type Theory, but the set of real numbers is not the type of Cauchy sequences, but the type of Cauchy sequences up to equiconvergence. More generally, a set in Bishop's constructive mathematics is represented by a type (commonly called "preset") together with an equivalence relation.

If A and B are equipped with equivalence relations, there is of course no guarantee that the choice function, f above, is extensional in the sense that it maps equivalent element to equivalent elements. This is the failure of the *extensional axiom of choice*, see (Martin-Löf 2009) for an analysis.

4 Extensions

4.1 The logical framework

The above completes the description of a core version of Intuitionistic Type Theory close to that of (Martin-Löf 1998).

In 1986 Martin-Löf proposed a reformulation of intuitionistic type theory; see (Nordström, Peterson and Smith 1990) for an exposition. The purpose was to give a more compact formulation, where λ and Π are the only variable binding operations. It is nowadays considered the main version of the theory. It is also the basis for the Agda proof assistant. The 1986 theory has two parts:

- the theory of types (the logical framework);
- the theory of sets (small types).

Remark 4.1. Note that the word "set" in the logical framework does not coincide with the way it is used in Bishop's constructive mathematics. To avoid this confusion, types together with equivalence relations are usually called "setoids" or "extensional sets" in Intuitionistic Type Theory.

The logical framework has only two type formers: $\Pi x : A.B$ (usually written $(x : A)B$ or $(x : A) \to B$ in the logical framework formulation) and U (usually called Set). The rules for $\Pi x : A.B$ ($(x : A) \to B$) are the same as given above (including η-conversion). The rules for U (Set) are also the same, except that the logical framework only stipulates closure under Π-type formation.

The other small type formers ("set formers") are introduced in the theory of sets. In the logical framework formulation each formation, introduction, and elimination rule can be expressed as the typing of a new constant. For example, the rules for natural numbers become

$$\begin{array}{rcl} N & : & \mathrm{Set}, \\ 0 & : & N, \\ s & : & N \to N, \\ R & : & (C : N \to \mathrm{Set}) \to C\,0 \to ((x : N) \to C\,x \to C\,(s\,x)) \to (c : N) \to C\,c. \end{array}$$

where we have omitted the common context Γ, since the types of these constants are closed. Note that the recursion operator R has a first argument $C : N \to \mathrm{Set}$ unlike in the original formulation.

Moreover, the equality rules can be expressed as equations

$$\begin{array}{rcl} R\,C\,d\,e\,0 & = & d : C\,0 \\ R\,C\,d\,e\,(s\,a) & = & e\,a\,(R\,C\,d\,e\,a) : C\,(s\,a) \end{array}$$

under suitable assumptions.

In the sequel we will present several extensions of type theory. To keep the presentation uniform we will however *not* use the logical framework presentation of type theory, but will use the same notation as in section 2.

4.2 A general identity type former

As we mentioned above, identity on natural numbers can be defined by primitive recursion. Identity relations on other types can also be defined in the basic version of Intuitionistic Type Theory presented in Section 2.

However, Martin-Löf (1975) extended Intuitionistic Type Theory with a uniform primitive identity type former I for all types. The rules for I express that the identity relation is inductively generated by the proof of reflexivity, a canonicial constant called r. (Note that r was coded by the number 0 in the introductory presentation of proof-objects in 1.3.) The elimination rule for the identity type is a generalization of identity elimination in predicate logic and introduces an elimination constant J. (We here show the formulation due to Paulin (1993) rather than the original formulation of Martin-Löf (1975).) The inference rules are the following.

I-formation

$$\frac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash a' : A}{\Gamma \vdash I(A, a, a')}$$

I-introduction.

$$\frac{\Gamma \vdash A \quad \Gamma \vdash a : A}{\Gamma \vdash r : I(A, a, a)}$$

I-elimination.

$$\frac{\Gamma, x : A, y : I(A, a, x) \vdash C \quad \Gamma \vdash b : A \quad \Gamma \vdash c : I(A, a, b) \quad \Gamma \vdash d : C[x := a, y := r]}{\Gamma \vdash J(c, d) : C[x := b, y := c]}$$

I-equality (under appropriate assumptions).

$$J(r, d) \quad = \quad d$$

By constructing a model of type theory where types are interpreted as *groupoids* (categories where all arrows are isomorphisms) Hofmann and Streicher (1998) showed that it cannot be proved in Intuitionistic Type Theory that all proofs of $I(A, a, b)$ are identical. This may seem as an incompleteness of the theory and Streicher suggested a new axiom K from which it follows that all proofs of $I(A, a, b)$ are identical to r.

The I-type is often called the *intensional identity type*, since it does not satisfy the principle of function extensionality. Intuitionistic Type Theory with the intensional identity type is also often called *Intensional Intuitionistic Type Theory* to distinguish it from *Extensional Intuitionistic Type Theory* which will be presented in section 6.1.

4.3 Well-founded trees

A type of well-founded trees of the form $Wx : A.B$ was introduced in (Martin-Löf 1982) (and in a more restricted form by Scott 1970). Elements of $Wx : A.B$ are trees of varying and arbitrary branching: varying, because the branching type B is indexed by $x : A$ and arbitrary because B can be arbitrary. The type is given by a *generalized inductive definition* since the well-founded trees may be infinitely branching. We can think of $Wx : A.B$ as the free term algebra, where each $a : A$ represents a term constructor $sup\, a$ with (possibly infinite) arity $B[x := a]$.

W-formation.

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash Wx : A.B}$$

W-introduction.

$$\frac{\Gamma \vdash a : A \qquad \Gamma, y : B[x := a] \vdash b : Wx : A.B}{\Gamma \vdash sup(a, y.b) : Wx : A.B}$$

We omit the rules of W-elimination and W-equality.

Adding well-founded trees to Intuitionistic Type Theory increases its proof-theoretic strength significantly (Setzer 1995).

4.4 Iterative sets and CZF

An important application of well-founded trees is Aczel's (1978) construction of a type-theoretic model of Constructive Zermelo Fraenkel Set Theory. To this end he defines the type of iterative sets as

$$V = Wx : U.x.$$

Let $A : U$ be a small type, and $x : A \vdash M$ be an indexed family of iterative sets. Then $sup(A, x.M)$, or with a more suggestive notation $\{M \mid x : A\}$, is an iterative set. To paraphrase: an iterative set is a set of iterative sets.

Note that an iterative set is a data-structure in the sense of functional programming: a possibly infinitely branching well-founded tree. Different trees may represent the same set. We therefore need to define a notion of extensional equality between iterative sets which disregards repetition and order of elements. This definition is formally similar to the definition of bisimulation of processes in process algebra. The type V up to extensional equality can be viewed as a constructive type-theoretic model of the cumulative hierarchy,

See the article on *Set Theory: Constructive and Intuitionistic ZF* by (Crosilla) for further infromation about CZF.

4.5 Inductive definitions

The notion of an inductive definition is fundamental in Intuitionistic Type Theory. It is a primitive notion and not, as in set theory, a derived notion where an inductively defined set is defined impredicatively as the smallest set closed under some rules. However, in Intuitionistic Type Theory inductive definitions are considered predicative: they are viewed as being built up from below.

The inductive definability of types is inherent in the meaning explanations of Intuitionistic Type Theory which we shall discuss in the next section. In fact, Intuitionistic Type Theory can be described briefly as a theory of inductive, recursive, and inductive-recursive definitions based on a framework of lambda calculus with dependent types.

We have already seen the type of natural numbers and the type of well-founded trees as examples of types given by inductive definitions; the natural numbers is an example of an ordinary finitary inductive definition and the well-founded trees of a generalized possibly infinitary inductive definition. The introduction rules describe how elements of these types are inductively generated and the elimination and equality rules describe how functions from these types can be defined by structural recursion on the way these elements are generated. According to the propositions as types principle, the elimination rules are simultaneously rules for proof by structural induction on the way the elements are generated.

The type formers $0, 1, +, \times, \to, \Sigma$, and Π which interpret the logical constants for intuitionistic predicate logic are examples of degenerate inductive definitions. Even the identity type (in Intensional Intuitionistic Type Theory) is inductively generated; it is the type of proofs generated by the reflexivity axiom. Its elimination rule expresses proof by pattern matching on the proof of reflexivity.

The common structure of the rules of the type formers can be captured by a general schema for inductive definitions (Dybjer 1991). This general schema has many useful instances, for example, the type $List(A)$ of lists with elements of type A has the following introduction rules:

$$\Gamma \vdash nil : List(A) \qquad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash as : List(A)}{\Gamma \vdash cons(a, as) : List(A)}$$

Other useful instances are types of binary trees and other trees such as the infinitely branching trees of the Brouwer ordinals of the second and higher number classes.

The general schema does not only cover inductively defined types, but also inductively defined families of types, such as the identity relation. The above mentioned type A^n of n-tuples of type A was defined above by primitive recursion on n. It can also be defined as an inductive family with the following introduction rules

$$\Gamma \vdash nil : A^0 \qquad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash as : A^n}{\Gamma \vdash cons(a, as) : A^{s\,n}}$$

The schema for inductive types and families is a type-theoretic generalization of a schema for iterated inductive definitions in predicate logic (formulated in natural deduction) presented by Martin-Löf (1971b). This paper immediately preceded Martin-Löf's first version of Intuitionistic Type Theory. It is both conceptually and technically a forerunner to the development of the theory.

It is an essential feature of proof assistants such as Agda and Coq that it enables users to define their own inductive types and families by listing their introduction rules (the types of their constructors). This is much like in typed functional programming languages such as Haskell and the different dialects of ML. However, unlike in these programming languages the schema for inductive definitions in Intuitionistic Type Theory enforces a restriction amounting to well-foundedness of the elements of the defined types.

4.6 Inductive-recursive definitions

We already mentioned that there are two main definition principles in Intuitionistic Type Theory: the inductive definition of types (sets) and the (primitive, structural) definition of functions by recursion on

the way the elements of such types are inductively generated. Usually, the inductive definition of a set comes first: the formation and introduction rules make no reference to the elimination rule. However, there are definitions in Intuitionistic Type Theory for which this is not the case and we simultaneously inductively generate a type and a function from that type defined by structural recursion. Such definitions are simultaneously *inductive-recursive*.

The first example of such an inductive-recursive definition is an alternative formulation *à la Tarski* of the universe of small types. Above we presented the universe formulated *à la Russell*, where there is no notational distinction between the element $A : U$ and the corresponding type A. For a universe à la Tarski there is such a distinction, for example, between the element $\hat{N} : U$ and the corresponding type N. The element $\hat{N}$ is called the *code* for N.

The elimination rule for U à la Tarski is:

$$\frac{\Gamma \vdash a : U}{\Gamma \vdash T(a)}$$

This expresses that there is a function T which maps a code a to its corresponding type $T(a)$. The equality rules defines this correspondence. For example,

$$T(\hat{N}) = N.$$

We see that U is inductively generated with one introduction rule for each small type former, and T is defined by recursion on these small type formers. The simultaneous inductive-recursive nature of this definition becomes apparent in the rules for Π for example. The introduction rule is

$$\frac{\Gamma \vdash a : U \qquad \Gamma, x : T(a) \vdash b : U}{\Gamma \vdash \hat{\Pi} x : a.b : U}$$

and the corresponding equality rule is

$$T(\hat{\Pi} x : a.b) = \Pi x : T(a).T(b)$$

Note that the introduction rule for U refers to T, and hence that U and T must be defined simultaneously.

There are a number of other universe constructions which are defined inductive-recursively: universe hierarchies, superuniverses (Palmgren 1998, Rathjen, Griffor and Palmgren 1998), and Mahlo universes (Setzer 2000). These universes are analogues of certain large cardinals in set theory: inaccessible, hyperinaccessible, and Mahlo cardinals.

Other examples of inductive-recursive definitions include an informal definition of computability predicates used by Martin-Löf in an early normalization proof of Intuitionistic Type Theory (Martin-Löf 1998). There are also many natural examples of "small" inductive-recursive definitions, where the recursively defined (decoding) function returns an element of a type rather than a type.

A large class of inductive-recursive definitions, including the above, can be captured by a general schema (Dybjer 2000) which extends the schema for inductive definitions mentioned above. As shown by Setzer, Intuitionistic Type Theory with this class of inductive-recursive definitions is very strong proof-theoretically (Dybjer and Setzer 2003). However, as proposed in recent unpublished work by Setzer, it is possible to increase the strength of the theory even further and define universes such as an *autonomous Mahlo universe* which are analogues of even larger cardinals.

5 Meaning explanations

The consistency of Intuitionistic Type Theory relative to set theory can be proved by model constructions. Perhaps the simplest method is an interpretation whereby each type-theoretic concept is given

its corresponding set-theoretic meaning, as outlined in section 1.3. For example the type of functions $A \to B$ is interpreted as the set of all functions in the set-theoretic sense between the set denoted by A and the set denoted by B. To interpret U we need a set-theoretic universe which is closed under all (set-theoretic analogues of) the type constructors. Such a universe can be proved to exist if we assume the existence of an inaccessible cardinal κ and interpret U by V_κ in the cumulative hierarchy.

Alternatives are realizability models, and for intensional type theory, a model of terms in normal forms. The latter can also be used for proving decidability of the judgments of the theory.

Mathematical models only prove consistency relative to classical set theory (or whatever other meta-theory we are using). Is it possible to be convinced about the consistency of the theory in a more direct way, so called *simple minded consistency* (Martin-Löf 1984)? In fact, is there a way to explain what it *means* for a judgment to be correct in a direct *pre-mathematical* way? And given that we know what the judgments mean can we then be convinced that the inference rules of the theory are valid?

An answer to this problem was proposed by Martin-Löf 1979 in the paper *Constructive mathematics and computer programming* (Martin-Löf 1982) and elaborated later on in numerous lectures and notes, see for example, Martin-Löf (1984, 1987). These meaning explanations for Intuitionistic Type Theory are also referred to as the *direct semantics*, *intuitive semantics*, *informal semantics*, *standard semantics*, or the *syntactico-semantical* approach to meaning theory.

This meaning theory follows the Wittgensteinian meaning-as-use tradition. The meaning is based on rules for building objects (introduction rules) of types and computation rules (elimination rules) for computing with these objects. A difference from much of the Wittgensteinian tradition is that also higher order types like $N \to N$ are given meaning using rules.

In spite of the *pre-mathematical* nature of this meaning theory, its technical aspects can be captured as a mathematical model construction similar to Kleene's *realizability* interpretation of intuitionistic logic, see the next section. The realizers here are the terms of type theory rather than the number realizers used by Kleene.

5.1 Computation to canonical form

The meaning of a judgment is explained in terms of the computation of the types and terms in the judgment. These computations stop when a canonical form is reached. These are the canonical forms used in lazy functional programming (for example in the Haskell language).

For the purpose of illustration we consider meaning explanations only for three type formers: $N, \Pi x : A.B$, and U. The context free grammar for the terms of this fragment of Intuitionistic Type Theory is as follows:

$$\begin{array}{rcl} a & ::= & 0 \mid s(a) \mid \lambda x.a \mid N \mid \Pi x : a.a \mid U \\ & & R(a, a, xx.a) \mid a\, a. \end{array}$$

The canonical terms are generated by the following grammar:

$$v ::= 0 \mid s(a) \mid \lambda x.a \mid N \mid \Pi x : a.a \mid U,$$

where a ranges over arbitrary, not necessarily canonical, terms. Note that $s(a)$ is canonical even if a is not.

Judgments are interpreted in terms of the relation $a \Rightarrow v$ between *closed* terms a and canonical forms (values) v given by the following computation rules:

$$\frac{c \Rightarrow 0 \quad d \Rightarrow v}{R(c, d, xy.e) \Rightarrow v} \qquad \frac{c \Rightarrow s(a) \quad e[x := d, y := R(a, d, xy.e)] \Rightarrow v}{R(c, d, xy.e) \Rightarrow v}$$

$$\frac{f \Rightarrow \lambda x.b \quad b[x := a] \Rightarrow v}{f\, a \Rightarrow v}$$

in addition to the rule

$$v \Rightarrow v$$

stating that a canonical term has itself as value.

5.2 The meaning of categorical judgments

A categorical judgment is a judgment where the context is empty and there are no free variables.

The meaning of the categorical judgment $\vdash A$ is that A has a canonical type as value. In our fragment this means that either of the following holds:

- $A \Rightarrow N$,
- $A \Rightarrow U$,
- $A \Rightarrow \Pi x : B.C$ and furthermore that $\vdash B$ and $x : B \vdash C$.

The meaning of the categorical judgment $\vdash a : A$ is that a has a canonical term of the canonical type of A as value. In our fragment this means that either of the following holds:

- $A \Rightarrow N$ and either $a \Rightarrow 0$ or $a \Rightarrow s(b)$ and $\vdash b : N$,
- $A \Rightarrow U$ and either $a \Rightarrow N$ or $a \Rightarrow \Pi x : b.c$ where furthermore $\vdash b : U$ and $x : b \vdash c : U$,
- $A \Rightarrow \Pi x : B.C$ and $a \Rightarrow \lambda x.c$ and $x : B \vdash c : C$.

The meaning of the categorical judgment $\vdash A = A'$ is that A and A' have the same canonical types as values. In our fragment this means that either of the following holds:

- $A \Rightarrow N$ and $A' \Rightarrow N$,
- $A \Rightarrow U$ and $A' \Rightarrow U$,
- $A \Rightarrow \Pi x : B.C$ and $A' \Rightarrow \Pi x : B'.C'$ and furthermore that $\vdash B = B'$ and $x : B \vdash C = C'$.

The meaning of the categorical judgment $\vdash a = a' : A$ is explained in a similar way.

It is a tacit assumption of the meaning explanations that the repeated computations of canonical forms is well-founded. For example, a natural number is the result of finitely many computations of the successor function s ended by 0. A computation which results in infinitely many computations of s is not a natural number in Intuitionistic Type Theory. (However, there are extensions of type theory, for example, Partial Type Theory, and Non-Standard Type Theory, where such infinite computations can occur, see section 6.3. To justify the rules of such theories the present meaning explanations do not suffice.)

5.3 The meaning of hypothetical judgments

According to Martin-Löf (1982) the meaning of a hypothetical judgment is reduced to the meaning of the categorical judgments by substituting the closed terms of appropriate types for the free variables. For example, the meaning of

$$x_1 : A_1, \ldots, x_n : A_n \vdash a : A$$

is that the categorical judgment

$$\vdash a[x_1 := a_1, \ldots, x_n := a_n] : A[x_1 := a_1, \ldots, x_n := a_n]$$

is valid whenever the categorical judgments

$$\vdash a_1 : A_1, \ldots, \vdash a_n[x_1 := a_1, \ldots, x_{n-1} := a_{n-1}] : A_n[x_1 := a_1, \ldots, x_{n-1} := a_{n-1}]$$

are valid.

6 Mathematical models

6.1 Categorical models

Hyperdoctrines. Curry's correspondence between propositions and types was extended to predicate logic in the late 1960s by Howard (1980) and de Bruijn (1970). At around the same time Lawvere developed related ideas in categorical logic. In particular he proposed the notion of a *hyperdoctrine* (Lawvere 1970) as a categorical model of (typed) predicate logic. A hyperdoctrine is an indexed category $P : T^{op} \to \mathbf{Cat}$, where T is a category where the objects represent types and the arrows represent terms. If A is a type then the *fibre* $P(A)$ is a category of propositions depending on a variable $x : A$. The arrows in this category are proofs $Q \vdash R$ and can be thought of as proof-objects. Moreover, since we have an indexed category, for each arrow t from A to B, there is a reindexing functor $P(B) \to P(A)$ representing substitution of t for a variable $y : B$. The category $P(A)$ is assumed to be cartesian closed and conjunction and implications are modelled by products and exponentials in this category. The quantifiers $\exists$ and $\forall$ are modelled by the left and right adjoints of the reindexing functor. Moreover, Lawvere added further structure to hyperdoctrines to model identity propositions (as left adjoints to a diagonal functor) and a comprehension schema.

Contextual categories, categories with attributes, and categories with families. Lawvere's definition of hyperdoctrines preceded Intuitionistic Type Theory but did not model all aspects of it. Nevertheless it influenced Scott's (1970) work on *Constructive Validity*, a somewhat preliminary precursor of Intuitionistic Type Theory. After Martin-Löf (1972) had presented a more definite formulation of the theory, the first work on categorical models was presented by Cartmell in 1978 with his notions of category with attributes and contextual category (Cartmell 1986). However, we will not define these structures here but instead the closely related *categories with families* (Dybjer 1996) which are formulated so that they directly model a variable-free version of a formulation of Intuitionistic Type Theory with explicit substitutions (Martin-Löf 1995).

A category with families is a functor $T : C^{op} \to \mathbf{Fam}$, where **Fam** is the category of families of sets. The category C is the category of contexts and substitutions. If Γ is an object of C (a context), then $T(\Gamma)$ is the family of terms of type A which depend on variables in Γ. If γ is an arrow in C representing a substitution, then the arrow part of the functor represents substitution of γ in types and terms. A category with families also has a terminal object and a notion of context comprehension, reminiscent of Lawvere's comprehension in hyperdoctrines. The terminal object captures the rules for empty contexts and empty substitutions. Context comprehension captures the rules for extending contexts and substitutions, and has projections capturing weakening and assumption of the last variable.

Categories with families are algebraic structures which model the general rules of dependent type theory, those which come before the rules for specific type formers, such as Π, Σ, identity types, universes, etc. In order to model specific type-former corresponding extra structure needs to be added.

Locally cartesian closed categories. From a categorical perspective the above-mentioned structures may appear somewhat special and ad hoc. A more regular structure which gives rise to models of Intuitionistic Type Theory are the locally cartesian closed categories. These are categories with a terminal object, where each slice category is cartesian closed. It can be shown that the pullback functor has a left and a right adjoint, representing Σ- and Π-types, respectively. Locally cartesian closed categories correspond to Intuitionistic Type Theory with extensional identity types and Σ and Π-types (Seely 1984, Clairambault and Dybjer 2014). It should be remarked that the correspondence with Intuitionistic Type Theory is somewhat indirect, since a coherence problem, in the sense of category theory, needs to be solved. The problem is that in locally cartesian closed categories type substituion is represented by pullbacks, but these are only defined up to isomorphism, see Curien 1993 and Hofmann 1994.

6.2 Set-theoretic model

Intuitionistic Type Theory is a possible framework for constructive mathematics in Bishop's sense. Such constructive mathematics is compatible with classical mathematics: a constructive proof in Bishop's sense can directly be understood as a proof in classical logic. A formal way to understand this is by constructing a set-theoretic model of Intuitionistic Type Theory, where each concept of type theory is interpreted as the corresponding concept in Zermelo-Fraenkel Set Theory. For example, a type is interpreted as a set, and the type of functions in $A \to B$ is interpreted as the set of all functions in the set-theoretic sense from the set representing A to the set representing B. The type of natural numbers is interpreted as the set of natural numbers. The interpretations of identity types, and Σ and Π-types were already discussed in the introduction. And as already mentioned, to interpret the type-theoretic universe we need an inaccessible cardinal.

Model in CZF. It can be shown that the interpretation outlined above can be carried out in Aczel's constructive set theory CZF. Hence it does not depend on classical logic or impredicative features of set theory.

6.3 Realizability models

The set-theoretic model can be criticized on the grounds that it models the type of functions as the set of all set-theoretic functions, in spite of the fact that a function in type theory is always computable, whereas a set-theoretic function may not be.

To remedy this problem one can instead construct a *realizability model* whereby one starts with a set of *realizers*. One can here follow Kleene's numerical realizability closely where functions are realized by codes for Turing machines. Or alternatively, one can let realizers be terms in a lambda calculus or combinatory logic possibly extended with appropriate constants. Types are then represented by sets of realizers, or often as partial equivalence relations on the set of realizers. A partial equivalence relation is a convenient way to represent a type with a notion of "equality" on it.

There are many variations on the theme of realizability model. Some such models, such as Aczel's Frege structures (Aczel 1980) tacitly assume set theory as the metatheory, whereas others explictly assume a constructive metatheory (Smith 1984).

Realizability models are also models of the extensional version of Intuitionistic Type Theory (Martin-Löf 1982) which will be presented in 7.1 below.

6.4 Model of normal forms and type-checking

In Intuitionistic Type Theory each type and each well-typed term has a normal form. A consequence of this normal form property is that all the judgments are decidable: for example, given a correct context Γ, a correct type A and a possibly ill-typed term a, there is an algorithm for deciding whether $\Gamma \vdash a : A$. This type-checking algorithm is the key component of proof-assistants for Intensional Type Theory, such as Agda.

The correctness of the normal form property can be expressed as a model of normal forms, where each context, type, and term are interpreted as their respective normal forms.

7 Variants of the theory

7.1 Extensional Type Theory

In Extensional Intuitionistic Type Theory (Martin-Löf 1982) the rules of I-elimination and I-equality for the general identity type are replaced by the following two rules:

$$\frac{\Gamma \vdash c : I(A, a, a')}{\Gamma \vdash a = a' : A} \qquad \frac{\Gamma \vdash c : I(A, a, a')}{\Gamma \vdash c = r : I(A, a, a')}$$

The first causes the distinction between propositional and judgmental equality to disappear. The second forces identity proofs to be unique. Unlike the rules for the intensional identity type former, the rules for extensional identity types do not fit into the schema for inductively defined types mentioned above.

These rules are however justified by the meaning explanations in Martin-Löf (1982). This is because the categorical judgment

$$\vdash c : I(A, a, a')$$

is valid iff $c \Rightarrow r$ and the judgment $\vdash a = a' : A$ is valid.

However, these rules make it possible to define terms without normal forms. Since the type-checking algorithm relies on the computation of normal forms of types, it no longer works for Extensional Type Theory.

On the other hand, certain constructions which are not available in Intensional Type Theory are possible in Extensional Type Theory. For example, function extensionality

$$(\Pi x : A.I(B, f\,x, f'\,x)) \to I(\Pi x : A.B, f, f')$$

is a theorem.

Another example is that W-types can be used for encoding other inductively defined types in Extensional Type Theory. For example, the Brouwer ordinals of the second and and higher number classes can be defined as special instances of the W-type (Martin-Löf 1984). More generally, it can be shown that all inductively defined types which are given by a *strictly positive type operator* can be represented as instances of well-founded trees (Dybjer 1997).

7.2 Univalent Foundations and Homotopy Type Theory

Univalent foundations refers to Voevodsky's programme for a new foundation of mathematics based on Intuitionistic Type Theory and employing ideas from homotopy theory. Here every type A is considered as a space, and the identity type $I(A, a, b)$ is the space of paths from point a to point b in A. The notion of ordinary set can then be thought of as a discrete space A where all paths in $I(A, a, b)$ are trivial loops. The origin of these ideas was the remarkable discovery by Hofmann and Streicher (1995) that the axioms of Intensional Type Theory do not force all paths to be trivial. This was shown by a model construction where each type is interpreted as a groupoid. Iterated identity types, e.g.

$$I(I(A, a, b), f, g)$$

represent higher homotopies. Deeper connections between identity types, and notions from homotopy theory and higher categories were subsequently discovered by Awodey and Warren (2009), Lumsdaine (2010), van den Berg and Garner (2011). Voevodsky realized that the whole type theory could be modelled by a well-known category studied in homotopy theory, namely the Kan simplicial sets. Inspired by this model he introduced the crucial *univalence axiom* for a small universe U of types, which states that the substitution map associated with the J-operator

$$I(U, a, b) \longrightarrow T(a) \cong T(b)$$

is an equivalence. Equivalence here refers to a general notion of equivalence of higher dimensional objects, as in the sequence *equal elements, isomorphic sets, equivalent groupoids, biequivalent bigroupoids*, etc. The univalence axiom expresses that "everything is preserved by equivalence", thereby realizing the informal categorical slogan that all categorical constructions are preserved by isomorphism, and its generalization, that all constructions of categories are preserved by equivalence of categories, etc.

The axiom of univalence was originally justified by Voevodsky's simplical set model. This model is however not constructive and (Bezem, Coquand and Huber 2014) has more recently proposed a model in cubical sets.

Although univalent foundations concern preservation of mathematical structure in general, strongly inspired by category theory, applications within homotopy theory are particularly actively investigated. Intensional type theory extended with the univalence axiom and so called higher inductive types is therefore also called "Homotopy Type Theory". We refer to Coquand's article on *Type Theory* (link) for further details.

7.3 Partial and Non-Standard Type Theory

Intuitionistic Type Theory is not intended to model Brouwer's notion of *free choice sequence*, although lawlike choice sequences can be modelled as functions from N. However, there are extensions of the theory which incorporate such choice sequences: namely *Partial Type Theory* and *Non-Standard Type Theory* (Martin-Löf 1990). The types in Partial Type Theory can be interpreted as Scott domains (Martin-Löf 1986, Palmgren and Stoltenberg-Hansen 1990, Palmgren 1991). In this way a type N which contains an infinite number ∞ can be interpreted. However, in Partial Type Theory all types are inhabited by a least element $\bot$, and thus the propositions as types principle is not maintained. Non-Standard Type Theory incorporates non-standard elements, such as an infinite number $\infty : N$ without inhabiting all types.

7.4 Impredicative Type Theory

The inconsistent version of Intuitionistic Type Theory of Martin-Löf 1971 was based on the strongly impredicative axiom that there is a type of all types. However, Coquand and Huet 1985 showed with their Calculus of Constructions, that there is a powerful impredicative but consistent version of type theory. In this theory the universe U (usually called *Prop*) is closed under the following formation rule for cartesian product of families of types:

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B : U}{\Gamma \vdash \Pi x : A.B : U}$$

This rule is more general than the rule for constructing small cartesian products of families of small types in Intuitionistic Type Theory, since we can now quantify over arbitrary types A, including U, and not just small types. (See Coquand and Huet 1988.)

The motivation for this theory was that inductively defined types and families of types become definable in terms of impredicative quantification. For example, the type of natural numbers can be defined as the type of Church numerals:

$$N = \Pi X : U.X \to (X \to X) \to X : U$$

This is an impredicative definition, since it is a small type which is constructed by quantification over all small types. Similarly we can define an identity type by impredicative quantification:

$$I(A, a, a') = \Pi X : A \to U.X\, a \to X\, a' : U$$

This is Leibniz' definition of equality: a and a' are equal iff they satisfy the same properties (ranged over by X).

Unlike in Intuitionistic Type Theory, the function type in impredicative type cannot be interpreted set-theoretically in a straightfoward way, see Reynolds 1984.

7.5 Proof assistants

In 1979 Martin-Löf wrote the paper "Constructive Mathematics and Computer Programming" where he explained that Intuitionistic Type Theory is a programming language which can also be used as a formal foundation for constructive mathematics. Shortly after that, interactive proof systems which help the user to derive valid judgments in the theory, so called proof assistants, were developed.

One of the first systems was the NuPrl system (PRL Group 1986), which is based on an extensional type theory similar to Martin-Löf (1982).

Systems based on versions of intensional type theory go back to the type-checker for the impredicative Calculus of Constructions was written around 1984 by Coquand and Huet. This led to the Coq system, which is based on the Calculus of Inductive Constructions (Paulin-Mohring 1993), a theory which extends the Calculus of Construction with primitive inductive types and families. The encodings of the pure Calculus of Constructions were found to be inconvenient, since the full elimination rules could not be derived and instead had to be postulated. We also remark that the Calculus of Inductive Constructions has a subsystem, the Predicative Calculus of Inductive Constructions, which follows the principles of Martin-Löf's Intuitionistic Type Theory.

Agda is another proof assistant which is based on the logical framework formulation of Intuitionistic Type Theory, but adds numerous features inspired by practical programming languages (Norell 2008). It is an intensional theory with decidable judgments and a type-checker similar to Coq's. However, in contrast to Coq it is based on Martin-Löf's predicative Intuitionistic Type Theory.

There are several other systems based either on the Calculus of Constructions (Lego, Matita) or on Intuitionistic Type Theory (Epigram, Idris); see (Pollack 1994, Asperti *et al.* 2011, McBride and McKinna 2004, Brady 2011).

Bibliography

References

[1] Peter Aczel. The type theoretic interpretation of constructive set theory. *Logic Colloquium '77 (Proc. Conf., Wrocław, 1977),* pp. 55 – 66, North-Holland, Amsterdam-New York, 1978.

[2] Peter Aczel. Frege Structures and the Notions of Proposition, Truth and Set. In: *The Kleene Symposium.* Studies in Logic and the Foundations of Mathematics 101. pp. 31– 59, North-Holland 1980.

[3] Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen ans Enrico Tassi. The Matita interactive theorem prover. *Automated deduction – CADE-23,* pp. 64 – 69, Lecture Notes in Computer Science, vol. 6803, Springer, Heidelberg, 2011.

[4] Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society* 146(2009), pp. 45 – 55.

[5] Errett Bishop. *Foundations of constructive analysis.* McGraw-Hill Book Co., New York-Toronto, Ont.-London 1967.

[6] Benno van den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society* 102(2011), pp. 370 – 394.

[7] Marc Bezem, Thierry Coquand and Simon Huber. *A model of type theory in cubical sets.* Preprint 2014.

[8] E. Brady. Idris — systems programming meets full dependent types. In: *Programming Languages meets Program Verification (PLPV 2011),* pp. 43 – 54, 2011.

[9] N. G. de Bruijn, The mathematical language AUTOMATH, its usage, and some of its extensions. *1970 Symposium on Automatic Demonstration (Versailles, 1968) pp. 29:61.* Lecture Notes in Mathematics, Vol. 125. Springer, Berlin.

[10] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic* 32 (1986), pp. 209 – 243.

[11] Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and Martin-Löf type theories. *Mathematical Structures in Computer Science* 24(6) (2014)

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

[13] Pierre-Louis Curien. Substitution up to isomorphism, *Fundamentae Informatica*, 1993.

[14] H.B. Curry and R. Feys. *Combinatory Logic.* North-Holland 1958.

[15] Peter Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics, in *Logical Frameworks*, editors Gerard Huet and Gordon Plotkin, pp 280-306, Prentice Hall 1991 .

[16] Peter Dybjer. Internal Type Theory. In: *Types for Proofs and Programs* Lecture Notes in Computer Science Volume 1158, pp. 120 – 134 Springer 1996.

[17] Peter Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theoretical Computer Science* 176 (1997), pp. 329 – 335.

[18] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic* 65 (2000), pp. 525 – 549.

[19] Peter Dybjer and Anton Setzer. Induction-recursion and initial algebras. *Annals of Pure and Applied Logic* 124 (2003) 1-47.

[20] Georges Gonthier. Formal proof of the four-color theorem. *Notices American Mathematical Society* 55 (2008), pp. 1382 –1393.

[21] Martin Hofmann. Interpretation of Type Theory in Locally Cartesian Closed Categories, *Proceedings of CSL, Springer LNCS*, 1994.

[22] Martin Hofmann. Syntax and semantics of dependent types. *Semantics and logics of computation (Cambridge, 1995),* 79130, Publ. Newton Inst., 14, Cambridge Univ. Press, Cambridge, 1997.

[23] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995),* 83-111, Oxford Logic Guides, 36, Oxford Univ. Press, New York, 1998.

[24] William A. Howard. The Formulae-as-Types Notion of Construction. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* J.P. Seldin and J.R. Hindley (red.), Academic Press 1980, pp. 479 – 490.

[25] Bart Jacobs. *Categorical logic and type theory.* Studies in Logic and the Foundations of Mathematics, 141. North-Holland Publishing Co., Amsterdam, 1999.

[26] F. William Lawvere. Equality in Hyperdoctrines and Comprehension Schema as an Adjoint Functor. In: *Proceedings of the American Mathematical Society Symposium on Pure Mathematics XVII* pp. 1 – 14, 1970.

[27] Peter LeFanu Lumsdaine. Weak omega-categories from intensional type theory. *Typed Lambda Calculus and Applications* 6(2010), pp. 1 – 19.

[28] Per Martin-Löf. *An intuitionistic theory of types.* Unpublished preprint, 1971.

[29] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In *Proceedings of the 2nd Scandinavian logic symposium,* ed. J.E. Fenstad. Amsterdam: North-Holland Publishing Company, 1971, pp. 179 – 216.

[30] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In Logic colloquium '73, ed. H.E. Rose and J. Shepherdson. Amsterdam: North-Holland Publishing Company, 1975, pp. 73 – 118.

[31] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, methodology and philosophy of science VI, Proceedings of the 1979 international congress at Hannover, Germany,* ed. L.J. Cohen, J. Los, H. Pfeiffer and K.-P. Podewski. Amsterdam: North- Holland Publishing Company, 1982, pp. 153 – 175.

[32] Per Martin-Löf. *Intuitionistic type theory: Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980.* Napoli: Bibliopolis 1984.

[33] Per Martin-Löf. Unifying Scott's theory of domains for denotational semantics and intuitionistic type theory (abstract). In: *Atti del Congresso 'Logica e Filosofia della Scienza, oggi', San Gimignano, 7 – 11 December 1983, Vol. I – Logica.* CLUEB, Bologna, 1986.

[34] Per Martin-Löf. Truth of a proposition, evidence of a judgment, validity of a proof. *Synthese* 73(1987), pp. 407 – 420.

[35] Per Martin-Löf. Mathematics of infinity. In *COLOG-88,* ed. P. Martin-Löf and G. Mints. Berlin: Springer, 1990, pp. 146 – 197.

[36] Per Martin-Löf. Analytic and synthetic judgments in type theory. In *Kant and contemporary epistemology*, ed. P. Parrini, 1992, pp 87–99. Dordrecht: Kluwer, 1994.

[37] Per Martin-Löf. *The substitution calculus.* Notes from a seminar, Stockholm 1995.

[38] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic* 1996(1), pp. 11–60.

[39] Per Martin-Löf. An intuitionistic theory of types. In *Twenty-five years of constructive type theory,* ed. G. Sambin and Jan M. Smith. Oxford: Clarendon Press, 1998, pp. 127 – 172. (Written in 1972 but unpublished.)

[40] Per Martin-Löf. 100 years of Zermelo's Axiom of Choice: What was the problem with it? In *Logicism, intuitionism, and formalism: What has become of them?,* ed. S. Lindström, E. Palmgren, K. Segerberg and V. Stoltenberg-Hansen. Dordrecht: Springer, 2009, pp. 209 – 219.

[41] Conor McBride and James McKinna. The view from the left. *Journal of Functional Programming* 14 (2004), pp. 69 – 111.

[42] Bengt Nordström, Kent Petersson and Jan M. Smith. *Programming in Martin-Löf's type theory. An introduction. Oxford University Press*, New York, 1990.

[43] Ulf Norell. Dependently Typed Programming in Agda. *Advanced Functional Programming* 2008: 230-266

[44] Erik Palmgren. *On Fixed Point Operators, Inductive Definitions and Universes in Martin-Löf Type Theory.* Doctoral dissertation in mathematics, Uppsala University 1991.

[45] Erik Palmgren. On universes in type theory. In: In *Twenty-five years of constructive type theory,* ed. G. Sambin and Jan M. Smith. Oxford: Clarendon Press, 1998, pp. .

[46] Erik Palmgren and Viggo Stoltenberg-Hansen. Domain interpretations of Martin-Löf's partial type theory, *Annals of Pure and Applied Logic* 48 (1990), 135-196.

[47] Christine Paulin-Mohring. Inductive Definitions in the system Coq - Rules and Properties. *TLCA* 1993: 328-345

[48] PRL Group. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Engelwood Cliffs, NJ, 1986.

[49] Randy Pollack. *The theory of LEGO.* PhD Thesis, Edinburgh 1994.

[50] Aarne Ranta. *Type-theoretical Grammar.* Oxford University Press, 1994.

[51] Michael Rathjen, Edward R. Griffor and Erik Palmgren. Inaccessibility in constructive set theory and type theory. *Annals of Pure and Applied Logic* 94 (1998), pp. 181 – 200.

[52] John C. Reynolds. Polymorphism is not Set-Theoretic. *Semantics of Data Types* 1984.

[53] Dana S. Scott. Constructive Validity. In: M. Laudet, D. Lacombe, L. Nolin and M. Schützenberg (eds.) *Symposium on Automatics Demonstration (Versailles, December 1968.* Lecture Notes in Mathematics, vol. 125, Springer 1970, pp. 237 – 275.

[54] Robert A. G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society* 95 (1984), pp. 33 – 48.

[55] Anton Setzer. Well-ordering proofs for Martin-Löf type theory. *Annals of Pure and Applied Logic* 92 (1998), pp. 113 – 159.

[56] Anton Setzer. Extending Martin-Löf type theory by one Mahlo-universe. *Archive for Mathematical Logic* 39 (2000), pp. 155 – 181.

[57] Jan M. Smith. An interpretation of Martin-Löf's type theory in a type-free theory of propositions. *Journal of Symbolic Logic* 49 (1984), pp. 730 – 753.

[58] Göran Sundholm. On the Philosophical Work of Per Martin-Löf. In: P. Dybjer, S. Lindström, E. Palmgren, G. Sundholm (Editors) *Epistemology versus Ontology: Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf.* Springer 2012, pp. xxiii-xxiv.

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

Academic Tools

[Auto-inserted by SEP staff]

Other Internet Resources

- Put URL for website 1 here
- Put URL for website 2 here

Related Entries

- Constructive Mathematics (Bridges and Palmgren): has a short introduction to Martin-Löf type theory as one of several approaches to constructive mathematics
- Intuitionistic Logic (Moschovakis): intuitionistic number theory with Kripke and realizability semantics
- Intuitionism in the Philosophy of Mathematics (Iemhoff):
- Type Theory (Coquand): universes, Girard's paradox, impredicative type theory, univalent foundations
- Church's Type Theory (Andrews):
- Set Theory: Constructive and Intuitionistic ZF (Crosilla)
- Set Theory (Jech)
- Zermelo's Axiomatization of Set Theory (Hallet)
- The Development of Proof Theory (von Plato)
- The Development of Intuitionistic Logic (van Atten)
- Proof-Theoretic Semantics (Schroeder-Heister)

type theory — church type theory — entry3