Inductive Definitions and Type Theory an Introduction

Thierry Coquand and Peter Dybjer

April 4, 1997

Abstract

Martin-Löf's type theory can be described as an intuitionistic theory of iterated inductive definitions developed in a framework of dependent types. It was originally intended to be a full-scale system for the formalization of constructive mathematics, but has also proved to be a powerful framework for programming. The theory integrates an expressive specification language (its type system) and a functional programming language (where all programs terminate). There now exist several proof-assistants based on type theory, and many non-trivial examples from programming, computer science, logic, and mathematics have been implemented using these.

In this series of lectures we shall describe type theory as a theory of inductive definitions. We emphasize its open nature: much like in a standard functional language such as ML or Haskell the user can add new types whenever there is a need for them. We discuss the syntax and semantics of the theory. Moreover, we present some examples of applications of the theory and discuss how they can be implemented by interacting with the window-based proof-assistant ALF.

1 What to read?

The present collection of five papers is intended to supplement the lectures. Here is a short description:

1. In this introduction we try to say something about the roots of Martin-Löf's type theory. While writing this historical introduction, we came to realize that the notion of inductive definitions permeates all of the history of proof theory and a large part of theoretical computer science. We must emphasize that this very preliminary account cannot even pretend to be a first approximation to a history of inductive definitions in mathematics and computer science and hope that anyone who knows more about this fascinating topic will share this knowledge with us.

   In this part we also give a brief account of inductive definitions in classical set theory using Aczel's *rule sets*. More information can be found in Aczel's article in the Handbook of Mathematical Logic [1]. Moreover, we indicate how rule sets can be employed for interpreting type-theoretic inductive definitions set-theoretically. For a detailed description of this interpretation we refer to Dybjer [7].

2. "Inductive families" by Peter Dybjer [6] describes a general schematic natural deduction formulation of inductive definitions in type theory. It also contains several examples which illustrate the generality of this schema.

3. “Pattern matching with dependent types” by Thierry Coquand [3] discusses an alternative to the traditional elimination rules for recursive function definitions in type theory.

4. “A user’s guide to ALF” (ALF is an acronym for “Another Logical Framework”) by Thorsten Altenkirch, Veronica Gaspes, Bengt Nordström, and Björn von Sydow describes how to use a proof editor for type theory. ALF supports pattern-matching with dependent types, a feature that has proved very useful in the context of formal proofs.

5. “Intuitionistic Model Construction and Normalization Proof” by Thierry Coquand and Peter Dybjer [4] describes an example of mechanical proof in ALF using type theory with inductive definitions and pattern-matching. The example is normalization in typed combinatory logic and lambda calculus and shows the power and elegance of the machinery of dependent types in the context of metamathematics.

The reader may also wish to consult the introductory text books on Martin-Löf’s type theory:

- “Intutitionistic Type Theory” by Per Martin-Löf [22].
- “Programming in Martin-Löf’s Type Theory An Introduction” by Bengt Nordström, Kent Petersson, and Jan Smith [24].
- “Type Theory and Functional Programming” by Simon Thompson [29].

Finally, we recommend the reading of Martin-Löf’s original papers. The following is a list of published papers on intuitionistic type theory and its precursor the intuitionistic theory of iterated inductive definitions.

- “Hauptsatz for the Intuitionistic Theory of Iterated Inductive Definitions” [17].
- “An Intuitionistic Theory of Types: Predicative Part” [19].
- “About Models for Intuitionistic Type Theories and The Notion of Definitional Equality” [18].
- “Constructive Mathematics and Computer Programming” [20].

2 Some Historical Roots

The importance of trying to put conceptual ideas in their historical context cannot be overestimated. Let us for instance cite Webb [31]: “But the more I tried to sort out and understand the arguments, to sift claim and counterclaim, the more I found that most of the central figures, however original they might have seemed, had really gotten key ideas from their teachers and predecessors, sometimes very obscure ones at that. And, most importantly, time after time I found that, *because* of my ignorance of these antecedants, *I had not, nor could have, really understood those ideas.*”

When looking for the roots of Intuitionistic Type Theory we must trace both main components: a functional component, based on λ-calculus, and an inductive component. This separation is quite similar in a programming language like ML, that incorporates both higher-order functions and concrete data types.

2.1 Prehistory

2.1.1 Induction on the natural numbers

Reasoning by induction on the natural numbers, though it had been used unconsciously for a long time, seems first to have been made explicit in the XVIIth century, in two different forms, by Pascal and Fermat. As formulated by Pascal and used for proving properties of numbers in his "triangle", to prove that a property P holds for all natural numbers, it is enough to prove that it holds for 1, and that it holds for $n+1$ whenever it holds for n. Fermat formulated it as a principle of "infinite descent" for establishing that a property is universally false: if a property P is such that $P(x)$ implies that there exists $y < x$ such that $P(y)$, then $P(x)$ never holds.

2.1.2 Transfinite induction

A completely new kind of induction, transfinite induction, was discovered by Cantor. He was first motivated by his analysis of trigonometry, but it also had an important rôle for general topology in the beginning. A typical instance is the definition of the Cantor-Bendixson index of a subset of the reals, where it is used for proving that a closed subset is either countable or has the cardinality of the continuum.

Transfinite induction gives rise to infinitely branching well-founded structures. There are examples, such as Paris-Harrington's version of Ramsey's theorem, which show that it is a genuinenly new principle (it is non-conservative over Peano arithmetic).

The set-theoretic notion of transfinite induction appears in type theory as generalized induction (as opposed to finitary induction).

2.1.3 Frege's and Dedekind's analysis

Frege and Dedekind independently discovered how inductive definitions can be explained in (impredicative) set theory. This explanation is the one currently used in mathematics. For instance the subgroup of a group G generated by a subset A is defined as the intersection of all subgroups of G containing A.

Skolem showed that a large part of mathematics can be developed without quantification over infinite sets. He essentially used what later came to be called primitive recursive arithmetic, where induction is only over non-quantified formulae.

The possibility of taking as primitive the notion of inductive definition in general is clearly expressed in the work of Lorenzen [15], but does not seem to have been formalised in detail. This possibility is suggested in [1], and one can see type theory as an attempt of actualisation.

2.2 Intuitionism

Historically, the next important step is the work of Brouwer, who, in his intuitionistic analysis of continuity, formulated the first examples of generalized inductive definitions. For instance, a countable ordinal is seen as an inductively generated object of three possible forms:

- the initial 1 ordinal,
- a successor ordinal $x+1$
- a limit ordinal $x_1 + x_2 + \ldots$, where (x_n) is a sequence of ordinals.

The principle of induction over this kind of objects was also explicitely formulated and considered primitive. To prove that $P(x)$ holds for any ordinal x, it is sufficient to show that $P(1)$ holds, that $P(x+1)$ holds if $P(x)$ holds, and that $P(x_1+x_2+\ldots)$ holds whenever $P(x_n)$ holds for all n.

2.3 Metamathematical Analysis

2.3.1 Post Canonical Systems

The canonical systems were invented by Post [26] in an attempt to find the most general form of a formal system, such as for example Principia Mathematica. An economical presentation is given in Martin-Löf's notes on constructive mathematics [16]. As shown for instance in [15, 16], the use of canonical Post systems allows for a short and elegant presentation of the basic notions of recursivity.

2.3.2 Tarski's Truth Definition

Natural examples of inductive definitions appeared in Tarski's work on truth definitions. First, one needed to give a precise definition of the syntax of the language. It is interesting that Tarski presented it as directly justified (though he points out that one may define it using an impredicative definition similar to the ones used in Principia Mathematica). Second, the truth definition was a complex example of a recursively defined function over this syntax.

2.4 Proof Theory

2.4.1 Functional Systems

Hilbert, in his study of the infinite, formulated the notion of higher-type primitive recursion (or primitive recursive functional). It was conjectured by Hilbert and shown by Ackermann that this gives a strict hierarchy of functions indexed by the complexity of the types: Ackermann's function can be defined by higher-type primitive recursion, but is not primitive recursive.

Closely connected to the consideration of these functional systems is the presentation of arithmetic given by Herbrand [11]. This presentation, which inspired the "Herbrand-Gödel" notion of general recursive functions, is quite interesting in the light of Martin-Löf's type theory. Indeed, Herbrand's presentation of arithmetic as an *open* system is similar to our presentation of type theory: he allows the addition of new function symbols and new computation rules, provided it is possible "to prove intuitionistically" that these computation rules are total.

This notion of primitive recursive functional was used in Gödel's Dialectica interpretation [8]. The goal was to give a consistency proof for arithmetic based on this notion. Spector extended this to a consistency proof of second-order arithmetic, by introducing the notion of bar-recursion.

The analysis of these systems seen as functional systems, and in particular, a meta-theoretical proof that all the functions defined in these systems are total, by Tait and Howard [28, 12], has been historically an important step in the discovery of the deep analogy between functional systems and proof systems.

2.4.2 Natural Deduction

Martin-Löf's type theory is formulated in the natural deduction style introduced by Gentzen. Reduction rules for natural deduction proofs were studied by Prawitz.

A general theory of inductive definitions in predicate logic in predicate logic was formulated by Martin-Löf [17]. He also proved a normalization theorem. This theory was an immediate precursor to intuitionistic type theory the first version of which appeared shortly afterwards.

2.5 Programming Languages

Inductively defined notions are permeate theoretical computer science too. One fundamental example is the notion of a language in the sense of formal language theory. But we shall focus on two aspects of programming languages here: inductive programming language constructs and inductive constructs for reasoning about programs.

2.5.1 Data types, Constructors, Structural Induction

The definition of data types in terms of their constructors were presented by Burstall [2]. He also discussed the general method of structural induction for provig properties of programs which manipulate such data types. Earlier, McCarthy and Painter [23] had used structural induction for proving the correctness of a compiler for arithmetic expressions.

2.5.2 Initial Algebras

In category theory inductive definitions are modelled by initial algebras. This is the basis for the algebraic approach to the semantics of data types, whereby one considers the initial algebra on a signature of constructor operations [9]. These ideas also influenced the design of programming languages such as OBJ, NPL, Hope, and Standard ML, where data types are defined by listing the constructors with their types.

2.5.3 Logic Programming

A natural interpretation of the definite Horn clauses of a logic program is as the introductory clauses of a (simulteneous) inductive definition of predicates. Hagiya and Sakurai [10] used this view to explain the foundation of logic programming, including negation as failure, and also pointed to the connection with Martin-Löf's theory of iterated inductive definitions.

2.5.4 Operational and Natural Semantics

Natural Semantics [13, 5], inspired from previous works by Plotkin [25] and by Martin-Löf [21], can be seen as an illustration of the use of inductive definitions in computer science. The programs are seen as inductively defined relations between inductively defined objects, and the proofs are done by structural induction [5].

2.6 The Curry-Howard Correspondence

In the 60s, people started to realize the deep analogies between functional systems and proof theoretical systems. For instance, the techniques developped by Tait for proving the totality of functions defined in Gödel's system T, could be used essentially to prove the normalisation property of systems of natural deduction. A typical example of this situation was the normalisation property for system F. Though it was found having in mind a termination theorem for a functional system, it was later realized that it can be applied, as by magic, to give a proof of Takeuti's conjecture, that was a conjecture about cut-elimination in a sequent calculus of second-order arithmetic.

These analogy were used by Prawitz in his study of "validity" of proofs. At about the same time, de Bruijn, Scott, and Martin-Löf showed that these analogy could be given a common expression in a general type theory with dependent types, unifying for instance the notion of proofs by induction and functions defined by induction.

Ideas closely related to the Curry-Howard correspondence had also appeared earlier in Lawvere's work on categorical logic from the 60s. His hyperdoctrines [14], which are categorical models of first order predicate logic, model propositions as objects and proofs as morphisms, and has a quite similar structure to type theory.

The notions of constructor and concrete data type turned out to be similar to the notions of introduction rule and logical constant respectively. Thus the close connection between functional systems studied in metamathematics and proof theory could be extended to a connection to functional programming languages. The notion of admissible rule corresponds closely to the notion of recursively defined functions. It should be noted for instance that the introduction of data types such as product or disjoint union, which seem so natural in a functional programming language, did not appear in the literature of functional systems before the 70s.

2.7 Rereading the history from a type-theoretic perspective

In the previous sections we have pointed to several traditions which have been brought together by the Curry-Howard isomorphism. It is interesting to reread this history in the light of the structure of type-theory. We note for example that there are several different levels of complexity of inductive definitions in type theory:

- First order datatypes and type constructors.
  - Simple type constructors such as disjoint union and cartesian product, and their logical analogues conjunction and disjunction.
  - Natural numbers.
  - Other simple datatypes such as lists and binary trees.
- Higher type constructors and transfinite types.
  - The function space construction and implication.
  - Brouwer ordinals.
- Dependent types.
  - Disjoint union and cartesian product of a family of types and their logical analogues the quantifiers.
  - The equality relation.
  - Other inductively defined relations, such as the transitive closure of a relation. Inductively defined families of types such as vectors (lists of a certain length).
- Universes. Metamathematical reflection.

There is also another kind of stratification coming from the distinction between ordinary primitive recursive definitions and primitive recursive functionals.

We may then structure the history by asking for some particular concept:

- Where does it come from? How has it been used informally?

- How was it explained/justified mathematically?
- When do its rules appear as part of a formal system?

In each case we have to look at both sides of the Curry-Howard isomorphism:

- Logical systems, induction principles.
- Programming languages and type systems, definition by recursion.

For example, analysing Π and Σ this way, we note that the cartesian product and disjoint union of a family of sets are well-known mathematical concepts with a clear set-theoretic definition. The formal type-theoretic rules for Π and Σ were used by Scott [27] who gives the following background [27][p 240]: "Next de Bruijn made good use of *cartesian products* of species (formation of function spaces) in connection with the *universal quantifier* - an idea also familiar to Lawvere and to a certain extent to Läuchli - and the author took this at once. Now dual to products (as Lawvere knows) are *disjoint sums* which must be used for the interpretation of the *existential quantifier* (cf. Kreisel - Goodman). These sums were not employed by de Bruijn, but it would be easy to add them to his system.". Later Martin-Löf introduced the eliminators *split* [19] and *funsplit* [22].

On the logical side we first have Heyting's and Kolmogov's explanations of the logical constants in terms of proofs and problems respectively. For example, according to Heyting, a proof of $\forall x.P[x]$ is a function that given an arbitrary element a returns a proof of $P[a]$, and a proof of $\exists x.P[x]$ is a pair consisting of an element a and a proof of $P[a]$. These explanations and also Gentzen's idea that logical constants are defined by their introduction rules can be read as saying that the logical constants are inductively defined sets of proofs. Gentzen also formulated the system NJ of natural deduction in intuitionistic logic. Rules for reduction of proofs in this system were given by Prawitz. Later Schroeder-Heister gave a generalized system of natural deduction with higher-level hypotheses. The formulation of $\forall$-elimination in this system inspired Martin-Löf's formulation of Π-elimination in terms of *funsplit*.

3 Inductive definitions in classical set theory

We here briefly review the interpretation of inductive definitions in classical set theory as given by Aczel [1]. Of particular interest is his notion of *rule set*, since it rather directly reflects the intuitive concept we have in mind. It is defined as follows.

A *rule* on a base set V in Aczel's sense is a pair of sets $\langle u, v\rangle$, often written

$$\frac{u}{v},$$

such that $u \subseteq V$ and $v \in V$.

Let Φ be a set of rules on V.

A set w is Φ-*closed* if

$$\frac{u}{v} \in \Phi \wedge u \subseteq w \supset v \in w.$$

There is a least Φ-closed set

$$\mathcal{I}(\Phi) = \bigcap \{w \subseteq V | w\ \Phi\text{-}closed\},$$

the set inductively defined by Φ.

Each rule set Φ on V generates a *monotone operator*

$$\phi(X) = \{v \in V | \exists \frac{u}{v} \in \Phi . u \subseteq X\}$$

on $\mathcal{P}(V)$, such that $\mathcal{I}(\Phi)$ is the least fixed point of ϕ.

This fixed point can also be characterized by transfinite iterations of ϕ. Define (non-constructively)

$$\begin{array}{rcll} \phi^0 & = & \emptyset & \\ \phi^{\lambda+1} & = & \phi(\phi^\lambda) & \\ \phi^\mu & = & \bigcup_{\lambda<\mu} \phi^\lambda & \qquad \text{for limit } \mu \end{array}$$

Then $\mathcal{I}(\Phi) = \bigcup_\lambda \phi^\lambda$. The smallest ordinal λ such that $\mathcal{I}(\Phi) = \phi^\lambda$ is called the ordinal of the inductive definition.

A rule set is *deterministic* provided it contains at most one rule with a given conclusion. For sets which are inductively defined by deterministic rule sets, functions can be defined by recursion on the way the elements are inducively generated.

3.1 Interpreting type theory in classical set theory

Martin-Löf's type theory can be interpreted in classical set theory in a rather direct way. 'The simplest interpretation of $\mathbf{ML_0}$ is in terms of a hierarchy within classical set theory, where Π, Σ, etc, correspond to the formation of cartesian products, disjoint unions etc. as already indicated above; function, i.e., elements of cartesian products are regarded as equal if for each argument their values are equal, etc.' (Troelstra [30, page 2]).

A full interpretation of type theory including a general schema for inductive definitions can be found in Dybjer [7]. Here we just give some basic examples.

Rule sets define *subsets* of a given set V. Since the inductive definitions in type theory are *fundamental*, no set V of which they are subsets is given a priori. So we must begin by choosing a set which is sufficiently large to include all the sets that we need for our interpretation. As long as all our type-theoretic inductive definitions are *small* it will be sufficient to let V be a Grothendieck universe of all small sets.

The type-theoretic natural numbers, inductively defined by the following rules

$$0 : N, \qquad \frac{n : N}{s(n) : N},$$

are for example interpreted in set theory by the set inductively defined by the following rule set on V:

$$\{\frac{\emptyset}{0}\} \cup \{\frac{\{n\}}{s(n)} | n \in V\}$$

Since rule sets can be infinitary, we can also interpret generalized induction. For example, the Brouwer ordinals

$$0 : \mathcal{O}, \qquad \frac{n : \mathcal{O}}{s(n) : \mathcal{O}}, \qquad \frac{\begin{array}{c}(x : N)\\ f(x) : \mathcal{O}\end{array}}{lim(f) : \mathcal{O}}.$$

generate the rule set

$$\{\frac{\emptyset}{0}\} \cup \{\frac{\{n\}}{s(n)} | n \in V\} \cup \{\frac{\{f(x) | x \in N\}}{lim(f)} | f \in N \to V\}$$

Here we have assumed that we have interpreted our constructors $0, s$ and lim appropriately.

With rule sets we can also interpret inductively defined families of sets, such as the family $N'(n)$ of n-element sets having the introduction rules:

$$\frac{n : N}{0'(n) : N'(s(n))}, \qquad \frac{n : N \qquad i : N'(n)}{s'(n,i) : N'(s(n))}.$$

(Note that $N'(n)$ in this *internal* sequence of finite sets corresponds to N_n in Martin-Löf's *external* sequence [19].) Here we create a rule set

$$\{\frac{\emptyset}{\langle s(n), 0'(n)\rangle} | n \in N\} \cup \{\frac{\{\langle n, i\rangle\}}{\langle s(n), s'(n,i)\rangle} | n \in N \wedge i \in V\}$$

for the relation $R_{N'} \subseteq N \times V$, such that $nR_{N'}i$ iff $i \in N'(n)$.

All inductively defined sets in type theory generate deterministic rule sets. Therefore we can interpret functions which are defined by primitive (or structural) recursion on a certain inductively defined set in type theory as set-theoretic functions defined on the corresponding set-theoretic inductively defined set. Given the computation rules for such a function it is easy to generate the corresponding rule set. For example, addition with the rules

$$\frac{m, n : N}{m + n : N}$$

$$\begin{array}{rcl} m + 0 & = & m, \\ m + s(n) & = & s(m+n), \end{array}$$

can be interpreted by the function inductively defined by the rule set

$$\{\frac{\emptyset}{\langle\langle m, 0\rangle, m\rangle} | m \in N\} \cup \{\frac{\langle\langle m, n\rangle, p\rangle}{\langle\langle m, s(n)\rangle, s(p)\rangle}\}.$$

There is also a form of primitive recursion associated with inductively defined families of sets. For example we can define a function ι which for $n : N$ injects $N'(n)$ into N by the rules

$$\frac{n : N \qquad i : N'(n)}{\iota(n,i) : N},$$

$$\begin{array}{rcll} \iota(s(n), 0'(n)) & = & 0 : N & (n : N), \\ \iota(s(n), s'(n,i)) & = & s(\iota(n,i)) : N & (n : N, i : N'(n)). \end{array}$$

This can be interpreted as the function defined inductively by the following rule set

$$\{\frac{\emptyset}{\langle\langle s(n), 0'(n)\rangle, 0\rangle} | n \in N\} \cup \{\frac{\langle\langle n, i\rangle, p\rangle}{\langle\langle s(n), s'(n,i)\rangle, s(p)\rangle}\}.$$

References

[1] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, 1977.

[2] R. M. Burstall. Proving properties of programs by structural induction. *Computer Journal*, (39):135–154, 1985.

[3] T. Coquand. Pattern matching with dependent types. In *Proceedings of The 1992 Workshop on Types for Proofs and Programs*, June 1992.

[4] T. Coquand and P. Dybjer. Intuitionistic model constructions and normalization proofs. Preliminary Proceedings of the 1993 TYPES Workshop, Nijmegen, 1993.

[5] J. Despeyroux. Proof of translation in natural semantics. In *Proceedings of the First ACM Conference on Logic in Computer Science*, pages 193–205, 1986.

[6] P. Dybjer. An inversion principle for Martin-Löf's type theory. In *Proceedings of the Workshop on Programming Logic*. Programming Methodology Group Report 54, Chalmers University of Technology and University of Göteborg, May 1989.

[7] P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.

[8] K. Gödel. *Collected Works, Volumes I and II.* Oxford University Press, 1986.

[9] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. *An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types.* Prentice Hall, 1978.

[10] M. Hagiya and T. Sakurai. Foundation of logic programming based on inductive definition. *New Generation Computing*, 2:59–77, 1984.

[11] J. Herbrand. On the consistency of arithmetic. In J. van Heijenoort, editor, *From Frege to Gödel*, pages 618–628. Harvard University Press.

[12] W. Howard. Functional interpretation of bar induction by bar recursion. *Compositio Mathematica*, 20:107 - 124.

[13] G. Kahn. Natural semantics. Technical Report 601, 1987.

[14] F. W. Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. In A. Heller, editor, *Applications of Categorical Algebra, Proceedings of Symposia in Pure Mathematics.* AMS, 1970.

[15] K. Lorentzen. *Métamathématique.* Edition Gauthier-Villars, 1962.

[16] P. Martin-Löf. *Notes on Constructive Mathematics.* Almqvist & Wiksell, Stockholm, 1968.

[17] P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 179–216. North-Holland, 1971.

[18] P. Martin-Löf. About models for intuitionistic type theories and the notion of definitional equality. In *Proceedings of the 3rd Scandinavian Logic Symposium*, pages 81–109, 1975.

[19] P. Martin-Löf. An intuitionistic theory of types: Predicative part. In *Logic Colloquium '73*, pages 73–118. North-Holland, 1975.

[20] P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science, VI, 1979*, pages 153–175. North-Holland, 1982.

[21] P. Martin-Löf. The domain interpretation of type theory, lecture notes. In K. Karlsson and K. Petersson, editors, *Workshop on Semantics of Programming Languages, Abstracts and Notes*, Chalmers University of Technology and University of Göteborg, August 1983. Programming Methodology Group.

[22] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[23] J. McCarthy and J. A. Painter. Correctness of a compiler for arithmetic expressions. In *Mathematical Aspects of Computer Science*, pages 33–41. AMS, 1967.

[24] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: an Introduction.* Oxford University Press, 1990.

[25] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN -19, Computer Science Depratment, Aarhus University, Aarhus, Denmark, 1981.

[26] E. Post. Absolutely unsolvable problems and relatively undecidable propositions. account of an anticipation. In M. Davis, editor, *The undecidable.* Raven Press, Hewlett, NY, 1965.

[27] D. S. Scott. Constructive validity. In *Symposium on Automatic Demonstration*, pages 237–275. Springer Lecture Notes in Mathematics 125, 1970.

[28] W. W. Tait. *Normal Derivability in Classical Logic.* Springer, 1968.

[29] S. Thompson. *Type Theory and Functional Programming.* Addison Wesley, 1991.

[30] A. S. Troelstra. On the syntax of Martin-Löf's type theories. *Theoretical Computer Science*, 51:1–26, 1987.

[31] J. Webb. *Mechanism, Mentalism and Metamathematics.* D. Reidel Publishing Company, 1980.