Inductive Definitions and Type Theory
an Introduction

Thierry Coquand

Preliminary draft for the TYPES Summer School, August 1999

Introduction

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 non-trivial examples from programming, computer science, logic, and mathematics have been implemented using these.

This note contains four parts. The first one is an (still preliminary) historical presentation of some historical roots of Type Theory, written with Peter Dybjer. The second part presents some basic examples of inductive definition in type theory. This shows how to use the elegant notations designed in functional programming for representing and checking inductive proofs. The third part presents the notion of inductive definition in the setting of *rule set* [1] and show how this notion can be represented in type theory. The fourth part explains why inductive definitions play an important rôle in proof theory [4].

1 Some Historical Roots

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.

1.1 Prehistory

1.1.1 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.

These two formulations can be found also for more complicated cases. The fact that a relation $>$ is well-founded can be expressed in two a priori different forms: the fact that there is no infinite decreasing chain $x_0 > x_1 > \ldots$ or the fact that the following induction schema (*noetherian* induction principle) is valid

$$((\forall z)((\forall y < z)\phi(y)) \rightarrow \phi(z)) \rightarrow \phi(x).$$

1.1.2 Transfinite induction

A completely new kind of induction, transfinite induction, was discovered by Cantor. He was first motivated by his analysis of Fourier series: finding sets A such that if $\Sigma(a_n \cos\ nx + b_n \sin\ nx)$ converges to 0 for all x in A then $a_n = b_n = 0$. A first use of ordinals is in the definition of the Cantor-Bendixson index of a subset of the reals: define first the derivative X' of a set X as the set of non isolated point of X (in any interval containing a point of X' there is another point of X). We can then define X'' as the derivative of X' and $X^{(n+1)}$ as the derivative of $X^{(n)}$. The new step is that we can define $X^{(\omega)}$ as being $\bigcap X^{(n)}$ and then $X^{(\omega+1)}$ as being the derivative of $X^{(\omega)}, \ldots$. One is then lead to introduce arbitrary ordinals α and to define $X^{(\alpha)}$ by transfinite induction. This was used for instance to prove that a closed subset is either countable or has the cardinality of the continuum (a particular case of the continuum hypothesis)[1].

Transfinite induction gave rise to the idea of well-founded structures. There are examples, such as Paris-Harrington's version of Ramsey's theorem, or the proof of the accessibility of ϵ_0 which show that it is a new principle, not provable in Peano arithmetics. The notion of well-founded induction appeared later in algebra in the form of noetherian rings, and it is not a coincidence that one of the first non effective proof, by Hilbert, was about noetherian rings.

The set-theoretic notion of transfinite induction appears in type theory as generalized induction (as opposed to finitary induction). In the framework of Aczel's rule sets [1], they appear as "infinitary" rules, that are rules with an infinite number of premisses.

The book [4] contains a nice historical introduction to the notion of inductive definitions.

[1] Let us recall that, by an extension of this method, it is possible to verify the continuum hypothesis as well for Borel sets, and for analytic sets.

1.1.3 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 together with an intuitionistic logic is suggested in Aczel's survey paper [1].

1.1.4 Descriptive Set Theory and Measure Theory

After Cantor, an important use of transfinite induction occurs in measure theory in the work of Borel, in his definition of Borel sets of real numbers (to simplify in $(0, 1)$). An open interval (r, s) is a Borel set and its measure $\mu(r, s)$ is $s - r$. If we have a disjoint collection of Borel sets A_n its union A is a Borel sets, and $\mu(A) = \Sigma\mu(A_n)$. And finally if $A \subseteq B$ are two Borel sets, then the difference $B - A$ is a Borel set, and $\mu(B - A) = \mu(B) - \mu(A)$. This is a good example of a generalised induction definition, and of a definition of a function by recursion. [2]

A little later, Suslin made two remarkable discovery about subsets of real numbers: first there are Σ^1_1 sets of reals (projection of a plane Borel set on the line) that are not Borel sets, second if a set *and* its complement are Σ^1_1 then this set is a Borel set. This theorem is closely connected to Brouwer's bar induction.

1.2 Intuitionism and Proof Theory

Historically, the next important step is the work of Brouwer, who, in his intuitionistic analysis of topological notions, 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 to be 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. The representation in type theory of such ordinals is especially natural.

Brouwer made also an important use of inductive definitions in his direct characterisation of the topology of the Baire space, that is the space of all infinite sequences of integers. This space has for basic open the set U_σ of all sequences starting by a given finite sequence $\sigma = n_0 \ldots n_k$ An open in general is given by a family U_{σ_i} of basic open and Brouwer noticed that we can describe directly when we have $U_\sigma \subseteq \cup_i U_{\sigma_i}$ without ever talking about infinite sequences: this is the case iff

- either σ extends one σ_i
- or, inductively, *all* direct extensions σn are such that $U_{\sigma n} \subseteq \cup_i U_{\sigma_i}$.

[2] Of course in the definition of the measure, one should check that the measure does not depend on the presentation of the set.

1.2.1 Post Canonical Systems

The canonical systems were invented by Post [32] in an attempt to find the most general form of a formal system. An economical presentation is given in Martin-Löf's notes on constructive mathematics [21]. As shown for instance in Lorenzen's of Martin-Löf's book [20, 21], the use of canonical Post systems allows for a short and elegant presentation of the basic notions of recursivity.

1.2.2 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: the truth of $(\forall x)\phi(x)$ is defined as the truth of all instances $\phi(n)$ for $n = 0, 1, \ldots$ Tarski showed that this inductive definition cannot be reduced to a finitary induction definition.

1.2.3 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 [12]. 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 understanding 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. His system may be seen as a precursor of the usual presentation of Peano arithmetic as a first-order theory, but in such a presentation, the "open character" of the system has been lost.

This notion of primitive recursive functional was used in Gödel's Dialectica interpretation [9]. 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 [34, 14], has been historically an important step in the discovery of the deep analogy between functional systems and proof systems. One goal was to see if it was possible to understand Spector's consistency proof using only inductive definitions (with intuitionistic logic). These attempts are described in [4]. One first positive result was the reduction of some form of *impredicative* quantification to inductively defined objects by Takeuti. We describe below a simplified version of such a reduction.

1.2.4 Realizability

Recursive realizability was introduced by Kleene [16] in order to relate precisely two quite different approach to the notion of effective processes: the constructive functions in intuitionism, and the notion of recursive functions. Modified realizability was introduced by Kreisel in [17], as a tool to show the non derivability of Markov's principle in Heyting's arithmetic. The main difference is that extracted computations are programs of a programming language that has the normalisation property.

(Modified) realizability can be seen as a natural projection operation on formulae (that become types) and proofs. This is explained in [29].

What is important here is that realizability suggests strongly an elegant approach to rigourous programming: to prove (constructively) a proposition can be seen as a way to achieve the "hand-in-hand" development of programs together with proofs that they satisfy their specification, advocated by Dijkstra or Gries [7][3]. The book [2] contains a detailed discussion of realizability and of its potential interest for computer science.

1.2.5 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 [22]. He also proved a normalization theorem. This theory was an immediate precursor to intuitionistic type theory the first version of which appeared shortly afterwards.

1.3 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.

1.3.1 Data types, Constructors, Structural Induction

The definition of data types in terms of their constructors was presented by Burstall [5], who introduced in this paper also the quite elegant case notation for the definition of functions over data types. He also discussed the general method of structural induction for proving properties of programs which manipulate such data types. This was further analyzed in Hoare's paper on recursive data structures [13]. Earlier, McCarthy and Painter [27] had used structural induction for proving the correctness of a compiler for arithmetic expressions.

1.3.2 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 [10]. 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.

1.3.3 Operational and Natural Semantics

Natural Semantics [15, 6], inspired from previous works by Plotkin [31] and by Martin-Löf [25], can be seen as an illustration of the use of inductive definitions in computer science. The programs are

[3]Let us cite Gries [11]. "The study of program correctness proofs has led to the discovery and elucidation of methods of developing programs. Basically, one attempts to develop a program and its proof hand-in-hand, with the proof ideas leading the way."

seen as inductively defined relations between inductively defined objects, and the proofs are done by structural induction [6].

1.4 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 [18], 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 connective 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 seems so natural in a functional programming language, did not appear in the literature of functional systems before the 70s.

1.5 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 [33] who gives the following background [33][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* [24] and $funsplit$ [26].

On the logical side we first have Heyting's and Kolmogorov'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.

2 Some Examples

We give some basic examples of inductive definitions represented in type theory. For the complete schema, see [30, 8].

2.1 Natural Numbers

We introduce three basic objects

$N : Set, \quad 0 : N, \quad S : N \to N.$

These form the *introduction* rules. The elimination rule is

$$\begin{aligned} natrec : & (C : N \to Set) C\ 0 \to \\ & ((x : N) C\ x \to C\ (S\ x)) \to \\ & (x : N) C\ x \end{aligned}$$

and the computation rules are

$$\begin{aligned} & natrec\ C\ a\ g\ 0 = a \\ & natrec\ C\ a\ g\ (S\ n) = g\ n\ (natrec\ C\ a\ g\ n). \end{aligned}$$

It is remarkable that the non dependent version of the elimination rule

$$natrec : (C : Set) C \to (N \to C \to C) \to N \to C$$

gives *primitive recursion* and not only iteration.

The usual justification of the elimination and computation rules uses the intuition of objects of N as "finite" sequences of S ended "eventually" by a 0.

2.2 Ordinals

We suppose to have already introduced N and we introduce

$Ord : Set, \quad 0 : Ord, \quad S : Ord \to Ord, \ Lim : (N \to Ord) \to Ord.$

The elimination rule is

$$\begin{aligned} ordrec : & (C : Ord \to Set) C\ 0 \to \\ & ((x : Ord) C\ x \to C\ (S\ x)) \to \\ & ((u : N \to Ord)((n : N) C\ (u\ n)) \to C\ (Lim\ u)) \to \\ & (x : Ord) C\ x \end{aligned}$$

and the computation rules are

$$\begin{aligned} & ordrec\ C\ a\ g\ h\ 0 = a \\ & ordrec\ C\ a\ g\ h\ (S\ x) = g\ x\ (ordrec\ C\ a\ g\ \ h\ x) \\ & ordrec\ C\ a\ g\ h\ (Lim\ u) = h\ u\ (\lambda n\ ordrec\ C\ a\ g\ h\ (u\ n)). \end{aligned}$$

Intuitively an ordinal is a tree, that may be infinite, but always well-founded. It is harder to make intuitive sense of these elimination and computation rules: this justification is of a similar nature as Brouwer's bar induction, as noticed in Troelstra-van-Dalen [35].

Exercice: Define the addition and multiplication of ordinals. Define then the operation $\alpha \to \omega^\alpha$ and use this to define an element $\epsilon_0 : Ord$ which represents the limit of the sequences $u_0 = 0,\ u_{n+1} = \omega^{u_n}$.

Try then to define such an element *without* using the elimination rule over ordinals. This is possible, but much more difficult.

This exercice gives a good example of how the use of complex notions (elimination rules) can simplify a construction.

2.3 Accessibility

We have at least two ways of representing a transition system (set with a binary relation) in type theory. The first way is to give a set A with a binary relation $R : A \to A \to Set$. The second way is to give a dependent type $B : A \to Set$ with a function $d : (x : A)B\ x \to A$. Intuitively, $B\ a$ represents the possible transition from $a : A$ and $d\ a\ b : A$ the result of applying transition $b : B\ a$ to a.

We show how to express in type theory that, intuitively, there is no infinite transition starting from a. The parameters are $A : Set,\ B : A \to Set,\ d : (x : A)B\ x \to A$. We introduce $Acc : A \to Set$ with a constant $sup : (a : A)((b : B\ a)Acc\ (d\ a\ b)) \to Acc\ a$ and an elimination rule

$$\begin{aligned} accrec : & (C : A \to Set) \\ & ((x : A)((z : B\ x)C\ (d\ x\ z)) \to C\ x) \to \\ & (x : A)Acc\ x \to C\ x \end{aligned}$$

and the computation rule is

$$accrec\ C\ g\ x\ (sup\ x\ y) = g\ x\ (\lambda z\ accrec\ C\ g\ (d\ x\ z)\ (y\ z)).$$

Exercice: Given two transition systems A_1, B_1, d_1 and A_2, B_2, d_2 define their products in the following way: take $A = A_1 \times A_2$ and $B\ (a_1, a_2) = B_1\ a_1 + B_2\ a_2$ with $d\ (a_1, a_2)\ (i\ b_1) = d_1\ a_1\ b_1$ and $d\ (a_1, b_1)\ (j\ b_2) = d_2\ a_2\ b_2$. Prove then that $Acc\ a_1 \to Acc\ a_2 \to Acc\ (a_1, a_2)$.

Exercice: Given a transition system A, B, d and $U : A \to Set$ defines in type theory the relation $U|\ a$ (read "U is a *bar* for a") which holds iff $U\ a$ or, inductively, $U|\ (d\ a\ b)$ for all $b : B\ a$. Show that if U is *monotone* that is $U\ a$ implies $U\ (d\ a\ b)$ for all $b : B\ a$ then so is $U|$.

2.4 Type Theory and Functional Programming

Type Theory can be seen as a functional programming language with dependent types. Important notions of natural deductions get a concrete representation in type theory:

- introduction rules are represented as constructors,
- canonical proofs are represented by terms starting with a constructor,
- the method of producing a canonical proof from a proof is represented by head-reduction,
- inductively defined types and relations are represented as *data types*,
- doing a proof by induction over an object is seen as a recursive definition of a proof object.

All these points get a concrete illustration for the examples presented above. For instance, $Acc\ a$ can now be thought of as a data type.

What can be noticed is that we get in this way a logic with a natural computational semantics. As noticed by Gentzen (for natural deduction), this logic is *intuitionistic* and does not justify the law of excluded-middle.

The computational semantics of type theory not only ensures, but gives direct and elegant proofs of the following strong properties

- consistency: the absurd proposition $\perp$ is not provable,
- existence property: if a property P of natural numbers can be represented in type theory, and if $\exists x : N.P(x)$ has a proof in type theory, then, from this proof, we can effectively find a natural number n that satisfies the property P.

3 Rule Sets in Set Theory and Type Theory

3.1 Rule Set in Set Theory

We have a base set A. The elements of A are thought of as statements. A *rule* on a base set A is a pair (X, x) with $X \subseteq A$ and $x \in A$.

Let Φ be a set of rules on A.

An *axiom* of Φ is an element $x \in A$ such that $(\emptyset, x) \in \Phi$.

We say that a subset $P \subseteq A$ is Φ-*closed* (or is a *theory*) iff $(X, x) \in \Phi$ and $X \subseteq P$ imply $x \in P$. The intersection of all Φ-closed subset of A is itself a Φ-closed subset T of A, which is the least Φ-closed subset of A. Intuitively an element of T is *derivable* from axioms by applying the rules of Φ. More generally we can define $U \vdash x$ if $U \subseteq A$ by saying that x belongs to all Φ-closed subsets that contain U and this represents the there is a well-founded derivation tree of x where all the leaves are in U.

A rule set Φ is *deterministic* if for any $x \in A$ there is at most one $X \subseteq A$ such that $(X, x) \in \Phi$.

A rule set Φ is *finitary* iff whenever $(X, x) \in \Phi$ the set X is finite.

Proposition: If Φ is finitary and $U \vdash x$ then there exists $U_0 \subseteq U$ *finite* such that $U_0 \vdash x$.

The proof is by showing that the set of all consequences of a finite subset of U is a set containing U which is Φ-closed.

These notions can be seen as a generalisation of Frege-Dedekind's analysis of induction. We can explain in set-theoretic terms not only the idea of *finitary* proof tree (which can be reduced to the notion of finite tree) but also the idea of *arbitrary well-founded* proof tree. Notice however that this explanation uses the notion of arbitrary intersections over all subsets of a set.

3.2 Tree Set Constructors

The parameters are $A : Set,\ B : A \rightarrow Set,\ C : (x : A)B\ x \rightarrow Set$ and $d : (x : A)(y : B\ x)C\ x\ y \rightarrow A$. Intuitively, A is a set of statements, $B\ x$ is the set of all possible basic rules for deriving a statement $x : A$ while $C\ x\ y$ is an index set enumerating all the premisses of the rule y, these premisses being all $d\ x\ y\ z,\ z : C\ x\ y$. The predicate corresponding to accessibility is then $T\ x$ meaning intuitively that x is *derivable* in this formal system.

We introduce thus $T : A \rightarrow Set$ and $sup : (a : A)(b : B\ a)((c : C\ a\ b)T\ (d\ a\ b\ c)) \rightarrow T\ a$ and the elimination rule

$$\begin{aligned} treerec : &(P : A \rightarrow Set) \\ &((x : A)(y : B\ x)((z : C\ x\ y)P\ (d\ x\ y\ z)) \rightarrow P\ x) \rightarrow \\ &(x : A)T\ x \rightarrow P\ x \end{aligned}$$

and the computation rule is

$$accrec\ P\ g\ x\ (sup\ x\ y\ h) = g\ x\ (\lambda z\ treerec\ P\ g\ (d\ x\ y\ z)\ (h\ z)).$$

These rules express that T is the *smallest* predicate P on A which is what can be called a theory: if there is a rules, all premisses satisfy P, then the conclusion satisfies P. To see the complete dependent elimination rule see [28].

Exercice: Given a predicate $U : A \rightarrow Set$ over A represent as an inductive definition in type theory the relation $U \vdash a$. The predicate $T\ a$ corresponds to the special case where U is the empty predicate. Show then the transitivity of $\vdash$ in the following form: if $U \vdash a$ and $V \vdash x$ for all $x : A$ satisfying U then we have $V \vdash a$.

4 A Normalisation Proof of Gödel's System T

Gödel system T is one of the simplest type system with one inductive definition. In this section, we present a proof of normalisation for *head-reduction* and closed terms. This proof uses as a main tool one inductive definition, of being *reducible* or *computable* at type N.

In order to present a *complete* proof (in particular, to be completely precise about *bound* variables) we use a simple form of *explicit substitution*. The proof is formulated in such a way that it follows the following computational interpretation: an object u of type N is a (well-founded) method that will eventually gives 0 or $S\ u_1$ with u_1 method of type N. A method of type $\alpha \to \beta$ is a method that applied to a method of type α with gives a method of type β.

The goal of the argument is to prove that all closed terms are correct methods in this sense. For doing this, we have to explain what should be the meaning of an *open* term as well: it should correspond to an *hypothetical* method. Explicit substitutions are also used to explain this notion.

4.1 Gödel's system T

The types are inductively defined as being N or of the form $\alpha \to \beta$. A *context* $\Gamma, \Delta, \ldots$ is a set of typed variables $x_1 : \alpha_1, \ldots, x_n : \alpha_n$ with $x_1, \ldots, x_n$ distinct. The typing rules are the following

- $\Gamma \vdash x : \alpha$ if $x : \alpha$ is in Γ,
- $\Gamma \vdash 0 : N$,
- $\Gamma \vdash S\ t : N$ if $\Gamma \vdash t : N$,
- $\Gamma \vdash t\ t' : \beta$ if $\Gamma \vdash t : \alpha \to \beta$ and $\Gamma \vdash t' : \alpha$,
- $\Gamma \vdash \lambda x\ t : \alpha \to \beta$ if x does not appear in Γ and $\Gamma, x : \alpha \vdash t : \beta$,
- $\Gamma \vdash f_{a,b} : N \to \alpha$ if $\Gamma \vdash a : \alpha$ and $\Gamma \vdash b : N \to \alpha \to \alpha$.

We recall that $\alpha \to \alpha_1 \to \alpha_2$ should be read $\alpha \to (\alpha_1 \to \alpha_2)$. If we have $\Gamma \vdash t : \alpha$ then t represents an *hypothetical* method of type α with free variables Γ. If $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$ and u_i is a closed expression of type α_i then we say that the substitution $\gamma = (x_1 = u_1, \ldots, x_n = u_n)$ is a (closed) instantiation of Γ. We can then form the instantiation $t\gamma$ of the method t and we get a closed expression of type α. Recursively, the closed expressions are hence described by the following clauses:

- 0 is a closed expression of type N,
- $S\ u$ is a closed expression of type N, if u is a closed expression of type N,
- $u\ u'$ is a closed expression of type β, if u, u' are closed expressions of type $\alpha \to \beta$, α respectively,
- $t\gamma$ is a closed expression of type α if $\Gamma \vdash t : \alpha$ and γ is an instantiation of Γ.

We describe next the head-reduction on closed expressions:

- $0\gamma \to 0$,
- $(S\ t)\gamma \to S\ (t\gamma)$,

- $(\lambda x\ t)\gamma\ u \to t(\gamma, x = u)$,
- $u\ u' \to u_1\ u'$ if $u \to u_1$,
- $f_{a,b}\gamma\ 0 \to a\gamma$,
- $f_{a,b}\gamma\ (S\ u) \to b\gamma\ u\ (f_{a,b}\gamma\ u)$,
- $f_{a,b}\gamma\ u \to f_{a,b}\gamma\ u_1$ if $u \to u_1$.

As usual, let us write $\to^*$ for the reflexive transitive closure of $\to$. We define inductively the predicate $R_N\ u$ which expresses that the closed expression u is *reducible* or *computable* of type N:

$$\frac{u \to^* 0}{R_N\ u} \qquad\qquad \frac{u \to^* S\ u_1 \quad R_N\ u_1}{R_N\ u}$$

and we define recursively on the type α what it means to be reducible at any type: $R_{\alpha\to\beta}\ u$ means that $R_\alpha\ u'$ implies $R_\beta\ (u\ u')$.

4.2 All terms are computable

The proofs is divided in two lemmas.

Lemma 1: If $R_\alpha\ u'$ and $u \to u'$ then $R_\alpha\ u$.

Corollary: If $R_\alpha\ u'$ and $u \to^* u'$ then $R_\alpha\ u$.

Lemma 2: If $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash t : \alpha$ and $R_{\alpha_i}\ u_i$ then we have $R_\alpha\ t(x_1 = u_1, \dots, x_n = u_n)$.

Theorem: All closed expressions are reducible.

Let us prove the theorem, assuming first lemma 2. We do an induction on the form of a closed expression v:

- $v = 0$ is reducible by definition,
- $v = S\ u$, by induction hypothesis, u is reducible, hence so is v,
- $v = u\ u'$ by induction hypothesis, both u and u' are reducible, hence so is v,
- $v = t(x_1 = u_1, \dots, x_n = u_n)$, by induction hypothesis all $u_1, \dots, u_n$ are reducible, hence, by lemma 2, so is v.

Hence the theorem is easily proved if we have already lemma 2. The proof of lemma 1 and its corollary are left as *exercices*: to prove lemma 1, do first an induction on the type α and then in the base case where $\alpha = N$ an induction on the definition of R_N. A predicate on expressions that satisfies the statement of lemma 1 is called *saturated*.

We prove now the key lemma 2. This is proved by induction on t:

- $t = x_i$, then we have $t\gamma \to u_i$; by hypothesis u_i is reducible, and hence $t\gamma$ is reducible *using lemma 1*,
- $t = 0$, then we have $0\gamma \to 0$, and hence $R_N\ (0\gamma)$ by definition of R_N,
- $t = S\ t'$, then we have $(S\ t')\gamma \to S\ (t'\gamma)$, but $t'\gamma$ is reducible by induction hypothesis, and hence so is $t\gamma$,

- $t = \lambda x\ t'$ is of type $\alpha \to \beta$, then we have $(\lambda x\ t')\gamma\ u \to t'(\gamma, x = u)$, for any reducible u of type α. By induction hypothesis $t'(\gamma, x = u)$ is reducible, and hence, *by lemma 1*, $t\gamma\ u$ is reducible,

- $t = t_1\ t_2$, then $t\gamma \to t_1\gamma\ (t_2\gamma)$; by induction hypothesis, both $t_1\gamma$ and $t_2\gamma$ are reducible, hence $t_1\gamma\ (t_2\gamma)$ is reducible and so is $t\gamma$ *using lemma 1*,

- $t = f_{a,b}$, then we prove that $t\gamma\ u$ is reducible by induction on the proof of $R_N\ u$. If $u \to^* 0$ then $t\gamma\ u \to^* a\gamma$. Since $a\gamma$ is reducible by induction hypothesis, then so is $t\gamma\ u$ *by* the corollary of lemma 1. If $u \to^* S\ u_1$ with u_1 reducible then $t\gamma \to^* b\gamma\ u_1\ (t\gamma\ u_1)$. By induction hypothesis, $t\gamma\ u_1$ is reducible; also by induction hypothesis (on t) $b\gamma$ is reducible. Since u_1 is reducible we can conclude *by* the corollary of lemma 1.

This proof is a good example of a proof by induction. There are two induction going on there: one is an induction on the types, and the other is the inductive definition of R_N.

The advantage of such a proof is that it is rather direct to extend it to the case of other data types. For instance, if we have a type of ordinals $O = 0 \mid S\ O \mid L\ (N \to O)$ the definition of being reducible of type O will be

$$\frac{u \to^* 0}{R_O\ u} \qquad \frac{u \to^* S\ u_1 \qquad R_O\ u_1}{R_O\ u} \qquad \frac{u \to^* L\ f \qquad (\forall v) R_N\ v \to R_O\ (f\ v)}{R_O\ u}.$$

4.3 Comment on $\vdash$ versus $\to$

It is interesting to compare the use of $\vdash$ and the use of $\to$ in this argument. One can consider $\to$ as an *internalisation* of $\vdash$, while the application of lambda-calculus is an internalisation of the operation of instantiation. The similarity in these two operations appear also in the framework of dependent types: if we write $[x : A]B$ for the dependent product and $\Gamma[x : A]$ the operation of extending a context with a type A then the rule of product formation can be formulated as: $\Gamma \vdash [x : A]B$ iff $\Gamma[x : A] \vdash B$.

5 An Interpretation of Π^1_1 Quantification

Since the work of Russell and the incisive remarks of Poincaré, one of the main goal of mathematical logic is to analyse *impredicativity*. The problem is the definition of a set of natural numbers (or a real number) obtained by quantifying over all sets of natural numbers (resp. over all real numbers). One can argue that the collection of all sets of natural numbers (resp. all real numbers) is not given in advance. It can "change" for instance by adding a new definition of a set. There is then a "circularity" problem in a definition which uses universal quantification over subsets: the definition itself may introduce a new set, changing the range of quantification used in the definition. According to Poincaré's analysis, this is the root of the contradictions that were found in the beginning of set theory.

A classical example of such a situation is the definition of a lower bound of a set of real numbers, where a real numbers is defined as a Dedekind cut, that is a downward closed subset of rational numbers. Given a set C of such subsets, the lower bound of the set C is the intersection of all elements of C, and this intersection is defined by universal quantification over *all* sets of rational numbers.

Here we show how to explain this circularity in term of inductive definitions/well-founded trees. This explains why inductive definitions play a crucial rôle in proof theory [4].

The basic idea [19] is that it is possible to explain inductively when a formula $(\forall X)A(X)$ is valid, if $A(X)$ does not contain any universal quantification over predicates: introduce a new atomic predicate X, about which we know nothing, and then explain inductively when $A(X)$ is correct. For instance $(\forall X)X\ 5 \rightarrow X\ 5$ is correct since $X\ 5 \rightarrow X\ 5$ is intuitionistically correct, with X variable.

The problem with this explanation is that it explains when a formula $(\forall X)A(X)$ is valid, but does not explain what it means to assume a formula $(\forall X)A(X)$ that is what can be the meaning of $((\forall X)A(X)) \rightarrow \phi$. In order to solve this, we have to use a Kripke model.

5.1 Reformulation of the problem

We consider the following (infinitary, hence described by an inductive definition) language $H(X)$

$$\phi = 0 \mid 1 \mid \bigwedge_n \phi_n \mid \phi \rightarrow \psi \mid X\ n$$

where 0 stands for the false, 1 for the true, $\bigwedge_n \phi_n$ is an infinitary conjunction and where n is a natural number $n = 0, 1, \ldots$. In this language we can represent arbitrary arithmetic formulae, and even formulae depending of an extra predicate variable X.

Given an interpretation of X it is possible to give a meaning of a formula $A(X)$ of $H(X)$, but it is not possible in general to explain *predicatively* when a formula $A(X)$ is valid for every possible interpretation of X. In some cases, this is possible, because we can reinterpret $(\forall X)A(X)$ for instance if $A(X)$ is

$$(\bigwedge_l (\bigwedge_k k \prec l \rightarrow X\ k) \rightarrow X\ l) \rightarrow X\ n$$

then $(\forall X)A(X)$ is equivalent to say that n is accessible with respect to the relation $\prec$. But, in general, for complicated formulae $A(X)$ we cannot reinterpret $(\forall X)A(X)$.

We can interpret the language $H(X)$ more generally in any Kripke model. Given an aribitrary poset $P, \leq$ with a least element 0, let $Down(P)$ the collection of downward closed subsets of P containing 0. Given an interpretation D of X as a function $N \rightarrow Down(P)$, we give a meaning

$[\![\phi]\!]_{X=D}$ of a formula of $H(X)$ as an element of $Down(P)$ using the following operation on $Down(P)$:

$$X \Rightarrow Y = \{x \in P \mid (\forall x' \leq x)\ x' \in X \Rightarrow x' \in Y\}$$

we have

- $[\![0]\!] = \{0\},\ [\![1]\!] = P$,
- $[\![\bigwedge \phi_n]\!] = \cap [\![\phi_n]\!]$,
- $[\![\phi \to \psi]\!] = [\![\phi]\!] \Rightarrow [\![\psi]\!]$,
- $[\![X\ n]\!] = D(n)$.

The problem will be the same however: in general $\cap_D [\![\phi]\!]_{X=D}$ cannot be described predicatively.

The main point is now that there exists a *particular* poset $P_0, \leq$ such that for any formula $\phi(X)$ of $H(X)$, $\cap_D [\![\phi]\!]_{X=D}$ can be described predicatively.

5.2 An inductively defined Poset

To build this poset P_0, consider the languages $H(X_0, \ldots, X_k)$ like $H(X)$ but with finitely many atomic predicates $X_0, \ldots, X_k$ and define P_0 to be the union $\cup_k H(X_0, \ldots, X_k)$. An element of P_0 is a formula $\phi(X_0, \ldots, X_k)$. As for $H(X)$, for *any* poset P, we can give a meaning $[\![\phi]\!]_{X_i=E_i}$ of any formula $\phi \in P_0$ as an element of $Down(P)$ if we are given a meaning $E_i : N \to Down(P)$ of each X_i.

Furthermore, we can explain using only inductive definitions when a formula $\phi \in P_0$ is valid. This can be done for instance by describing inductively when a finite set of formulas Γ entails another formula ϕ:

- $\Gamma \vdash \phi$ if $\phi \in \Gamma$,
- $\Gamma \vdash \phi \to \psi$ if $\Gamma, \phi \vdash \psi$,
- $\Gamma \vdash 1$,
- $\Gamma \vdash \phi$ if $\Gamma \vdash 0$,
- $\Gamma \vdash \bigwedge \phi_n$ if $\Gamma \vdash \phi_n$ for all n.

We write now $\phi \leq \psi$ iff $\phi \to \psi$ is valid. This defines a poset P_0 with a least element 0. The fact that the relation $\leq$ is transitive follows from Gentzen's cut-elimination theorem: if $\Gamma \vdash \phi$ and $\Delta, \phi \vdash \psi$ then $\Gamma, \Delta \vdash \psi$, which is a good example of a theorem proved by induction.

It can then be proved that if $\phi \leq \psi$ holds then for any poset P and for any interpretation E_i of X_i we have $[\![\phi]\!]_{X_i=E_i} \subseteq [\![\psi]\!]_{X_i=E_i}$.

This expresses the *soundness* of Kripke models with respect to this logical calculus. The *completness* will be proved below.

The paper [23] presents a calculus of proofs for a calculus similar to the one given here.

5.3 Inductive Explanation of Π^1_1 quantification

Given a formula $A(X)$ of $H(X)$ we interpret now $(\forall X)A(X)$ as a downward closed set S of P_0. This is the set S of all formulae $\phi \in P_0$ such that $\phi \leq A(X_i)$ whenever X_i does not appear in ϕ. Notice that if $\phi \leq A(X_i)$ for *one* X_i not in ϕ, then we have $\phi \leq A(X_l)$ for all X_l not in ϕ. Of course, it is essential that this set is defined *without use* of universal quantification over predicates.

Lemma 1: S is a downward closed subset of H.

Proof: If $\phi \in S$ and $\psi \leq \phi$, let X_l be an atomic predicate not in ϕ neither in ψ. We have then $\phi \leq A(X_l)$ by hypothesis and hence $\psi \leq A(X_l)$. This implies $\psi \in S$ by the remark above. □

If $\phi \in H$ let $\downarrow(\phi)$ be the set of formulae $\leq \phi$. We let $D_k(n)$ be the set $\downarrow(X_k\ n)$.

Lemma 2: We have $\downarrow(\wedge\phi_n) = \cap\downarrow(\phi_n)$ and $\downarrow(\phi \rightarrow \psi) = \downarrow(\phi) \Rightarrow \downarrow(\psi)$.

Lemma 3: $[\![\phi]\!]_{X_k=D_k} = \downarrow(\phi)$.

Proof: This is true by definition for $\phi = X_k\ n$ and for $\phi = 0$ or $\phi = 1$.
It then holds by induction on the formula ϕ using lemma 2. □

It follows from this lemma that we have $\phi \leq \psi$ *iff* for all Kripke models and all interpretations of X_i, we have $[\![\phi]\!] \subseteq [\![\psi]\!]$.
This lemma is crucial in the argument, and holds only because of the use of ω-logic. [4]

Lemma 4: S is the intersection of all interpretations of $[\![A(X)]\!]_{X=D}$ where D varies over all possible functions $N \rightarrow Down(P_0)$.

Proof: Our claim is that S is the intersection of all $[\![A(X)]\!]_{X=D}$. First, if $\phi \in S$ then $\phi \leq A(X_i)$ for X_i not in ϕ. Hence $\phi \rightarrow A(X_i)$ is a valid formula, which holds in any model, in particular in the Kripke model over H. It follows that we have

$$[\![\phi]\!]_{X_k=D_k,X_i=D_i} = [\![\phi]\!]_{X_k=D_k,X_i=D} \subseteq [\![A(X_i)]\!]_{X_k=D_k,X_i=D} = [\![A(X_i)]\!]_{X_i=D}.$$

By lemma 3, we have $[\![\phi]\!]_{X_k=D_k} = \downarrow(\phi)$ and hence $\phi \in [\![A(X_i)]\!]_{X_i=D}$ as expected.
Conversely if $\phi \in [\![A(X)]\!]_{X=D}$ for all possible interpretation D, then in particular for X_i not in ϕ we have

$$\phi \in [\![A(X)]\!]_{X=D_i} = [\![A(X_i)]\!]_{X_i=D_i}.$$

Since $[\![A(X_i)]\!]_{X_i=D_i} = \downarrow(A(X_i))$ by lemma 2, this implies $\phi \leq A(X_i)$. □

5.4 Comprehension Axiom

Second-order arithmetic HA^2 is defined as a two sorted logic: with variables $n, m, x, y, \ldots$ over natural numbers and variables $X, Y, Z, \ldots$ over *sets* of natural numbers. We can form, besides the usual propositions of Heyting arithmetic, the atomic proposition $x \epsilon X$ and the comprehension axiom can be stated as

$$\exists X \forall x [x \epsilon X \equiv \phi(x)]$$

where $\phi(x)$ is *any* formula.

If we restrict $\phi(x)$ to a formula of the form $\forall X_1 \ldots \forall X_n \psi(X_1, \ldots, X_n, x)$ ψ without set quantification, we have Π^1_1-comprehension. We have built a model of Π^1_1-comprehension *without parameters*: the formula $\phi(x)$ has no free set variables.

[4] It follows from Gödel's incompleteness theorem that the impredicative quantification we deal with cannot be explained using only finitary inductive definitions.

5.5 How long can we go?

The first to obtain an explanation of Π^1_1 quantification in terms of inductive definitions was Takeuti, by a somewhat indirect method: he obtained a partial cut-elimination result generalising the one of Gentzen using ordinal diagrams, which can be proved well-founded using inductive definitions. The method we have used is inspired from the one in the book of Buchholz-Schütte [3], and from some remarks of Lorenzen.

We have explained only a special case of Π^1_1 quantification: in $(\forall X)A(X)$ the formula $A(X)$ cannot have any universal quantification over predicates. In order to cover a more general case, one needs to use iterated inductive definitions.

The program of reducing the circularity of quantifications over predicates is going on. The more recent progress in this direction can be elegantly described in terms of type theory: one considers type theory with W-types (iterated inductive definitions) with stronger and stronger *reflection principles*. These reflection principles are quite similar to large cardinal axioms in set theory. A simple explanation of Π^1_2 comprehension, generalising the one we have given is still a challenge!

References

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

[2] M. Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, 1985.

[3] W. Buchholz and K. Schutte. *Proof theory of impredicative subsystems of analysis.* Studies in Proof Theory. Monographs, 2. Bibliopolis, Naples., 1988.

[4] P. W. Buchholz W., Feferman S. and S. W. *Iterated Inductive Definitions and Subsystems of Analysis.* Springer, Berlin, 1981.

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

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

[7] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[8] P. Dybjer. Inductive families. *Formal Aspects of Computing 6*, pages 440–465, 1994.

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

[10] 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.

[11] D. Gries. *The Science of Programming.* Springer Texts and Monographs in Computer Science, 1981.

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

[13] C. Hoare. Recursive data structures. *International Journal of Computer and Information Sciences*, 4:105 – 124, 1975.

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

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

[16] S. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[17] G. Kreisel. Interpretation of analysis by mean of constructive functionals of finite type. In Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland, 1959.

[18] 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.

[19] K. Lorenzen. Logical reflection and formalism. *Journal of Symbolic Logic*, 23:241–249, 1958.

[20] K. Lorenzen. *Métamathématique.* Edition Gauthier-Villars, 1962.

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

[22] 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.

[23] P. Martin-Löf. Infinite terms and a system of natural deduction. *Composition Mathematica*, 34:93–103, 1972.

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

[25] 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.

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

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

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

[29] C. Paulin-Mohring. Extracting fω programs from proofs in the calculus of constructions. In *Sixteenth Annual ACM Symposium on Principles of Programing Languages.* ACM, 1989.

[30] C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings Typed λ-Calculus and Applications*, pages 328–245. Springer-Verlag, LNCS, March 1993.

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

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

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

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

[35] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics, an introduction.* North-Holland, 1988.