From constructivism to computer science

A.S. Troelstra*

My field is mathematical logic, with a special interest in constructivism, and I would not dare to call myself a computer scientist. But some computer scientists regard my work as a contribution to their field; and in this text I shall try to explain how this is possible, by taking a look at the history of ideas.

I want to describe how two interrelated ideas, connected with the constructivistic trend in the foundations of mathematics, developed within mathematical logic and ultimately diffused into computer science.

It will be seen that this development has not been a quite straightforward one. In the history of ideas it often looks as if a certain idea has to be discovered several times, by different people, before it really enters into the "consciousness" of science.

1 The proof interpretation

The first of these ideas is the so-called proof interpretation of intuitionistic logic, also known as the Brouwer–Heyting–Kolmogorov interpretation, since it is implicit in the writings of L.E.J. Brouwer, and made explicit by A.Heyting and, independently, by A. Kolmogorov.

The Dutch mathematician L.E.J. Brouwer (1881–1966) achieved international fame by his contributions to topology. However, his earliest interest was in the foundations of mathematics. His philosophy of mathematics, a special form of constructivism, which became known under the somewhat misleading name of intuitionism, generated a lot of interest, gave rise to a

*This paper is an expanded version of a lecture presented november 15, 1996, at the Technische Universität München, on the occassion of receiving the F.L. Bauer-Preis. I am indebted to K.R. Apt for detailed comments on an earlier draft of this paper.

heated debate, but attracted few actual followers. But even those who did not agree with Brouwer's ideas, such as D. Hilbert, were influenced by them and by the debate generated by Brouwer's views. Ideas developed in the context of intuitionism turned out to have a relevance which transcends the original setting.

Brouwer's ideas on the foundations of mathematics were embedded in a highly personal and rather extreme version of idealistic philosophy, but it would carry us too far to go into this here. The first exposition of these ideas on the foundations of mathematics is found in his thesis from 1907 ("On the Foundations of Mathematics").

Briefly, Brouwer viewed mathematics as the activity of building constructions in the mind (of an ideal mathematician); mathematics is about such mental constructions, not about objects in some outside reality. For Brouwer, there is no platonistic universe of abstract ideas existing somewhere quite independently of human cognition.

In his thesis, Brouwer had not yet realized the effect of his views on logic; but in a paper which appeared a year later, in 1908 ("On the unreliability of the logical principles") he did see the consequences. He demonstrated that from an intuitionistic point of view, we cannot assume that a mathematical statement is either true or false, independently of human knowledge; we can assert "A or not A" only in case we either have a proof of A or an argument showing that any attempt at constructing a proof of A must fail.

As a result, if A represents an open mathematical problem, such as Riemann's hypothesis, we cannot assert, intuitionistically, that A or not A, since we do not know how either to prove A or to refute A. Hence the so-called "principle of the excluded middle"

$$A \vee \neg A$$

is not an intuitionistic universal logical law. (This does not mean to say that $A \vee \neg A$ in our example is *false* since that would mean we could derive a contradiction from $A \vee \neg A$, not just that we are not able to prove it.)

Implicit in these considerations is another interpretation of logic. In the usual *classical logic*, the meaning of a logical operator c combining statements $A_1, A_2, \ldots, A_n$ into $\mathrm{c}(A_1, A_2, \ldots, A_n)$ is explained by describing how the *truth* or *falsity* of $\mathrm{c}(A_1, A_2, \ldots, A_n)$ depends on the truth or falsity of each of the components A_1, A_2, $\ldots$, A_n.

Intuitionistically, the meaning of c is described by explaining what is to be regarded as a *proof* of $c(A_1, A_2, \ldots, A_n)$, assuming one knows what counts as a proof of each of A_1, A_2,..., A_n.

This was made explicit, first by A. Heyting (1898–1966) in 1930 ([Hey31], and more fully in 1934 ([Hey34], and independently by A. Kolmogorov (1903–1987) in 1932 ([Kol32]). Actually, Heyting and Kolmogorov regarded their respective formulations as distinct. To convey the idea, let us consider the explanations of $\vee$ (disjunction, "or"), $\rightarrow$ (implication, "implies") and $\neg$ (negation, "not") in Heyting's formulation:

- A proof of $A \vee B$ is given by exhibiting either a proof of A or a proof of B.
- A proof of $A \rightarrow B$ is a construction f transforming any proof d of A into a proof $f(d)$ of B.
- A proof of $\neg A$ is a construction which transforms any possible proof of A into a contradiction, e.g. $0 = 1$.

One can take the notion of a falsehood or contradiction ($\perp$) as primitive; then $\neg A$ is the same as $A \rightarrow \perp$. A falsehood is simply a statement which cannot have a proof.

These explanations are actually rather vague, since they make use of unexplained primitive notions[1] such as "(constructive) proof" and "construction".

However, notwithstanding the vagueness, the explanations serve to justify the usual laws of intuitionistic logic, such as $A \vee B \rightarrow B \vee A$ and $A \rightarrow (B \rightarrow A)$, while at the same time making it clear why, for example, the principle of the excluded middle is not intuitionistically acceptable.

In Kolmogorov's interpretation, intuitionistic logic is seen as a calculus of problems; each proposition represents a problem. For example,

- Problem $A \vee B$ is solved by solving either A or B.
- $A \rightarrow B$ is solved if one can reduce the solution of problem B to the solution of problem A.

[1] Actually, by choosing an unintended, "non-standard" interpretation of these notions, one can even obtain that classical logic is valid!

- $\neg A$ is solved if one can show there is no solution of problem A.

Kolmogorov thought that his interpretation, in contrast to Heyting's explanations, also made sense classically. It should be noted, however, that with the concept of a (mathematical) problem Kolmogorov introduced a notion not present in standard classical logic (even if, on an informal level, mathematics abounds with problems). So one might say that Kolmogorov interprets intuitionistic logic in an extension of classical logic, namely logic enriched with the notion of "solving a problem".

There is a parallel here with a contribution of K. Gödel from 1933, where he describes an interpretation of intuitionistic logic into classical logic enriched with an extra propositional operator Bew, with intuitive meaning "A is provable" for "Bew(A)". As Gödel is careful to point out, "provable" should be taken to mean here "provable by any correct means", not "provable in a fixed formal system"; for the latter notion, some of the principles postulated by Gödel for Bew are incorrect, as follows from Gödel's incompleteness theorems. If one reads $\Box$ (necessity) for Bew, Gödel's principles for Bew yield a well-known and elegant axiomatization of the modal logic **S4**. Gödel mentions indeed Kolmogorov's paper in a footnote, but presumably his work is inspired by Heyting's explanations of the intuitionistic logical operators.

Returning to Kolmogorov's own formulation, one may observe that a problem is usually expressed by a mathematical statement, and "solving the problem A" then corresponds to "proving A". This led Heyting to the conclusion, many years later, that Kolmogorov's interpretation and his own were basically the same ([Hey58]).

2 Realizability

The vagueness of the basic notions of "constructive proof" and "construction" prevents immediate technical applications of the proof interpretation. For such applications we need to find a more concrete, perhaps unintended, "model" for these notions. Such a model need not be faithful to the informal interpretation, provided that it embodies some essential constructive principles.

A beautiful example of such a concrete version of the proof interpretation is S.C. Kleene's interpretation of statements of number theory, first published in 1945, but conceived in 1941.

The key idea is, that the mathematically interesting information contained in a constructive proof of a statement A is the information telling us how existential statements and disjunctions occurring in A are to be realized.

Kleene defines a notion $n \,\underline{\mathrm{r}}\, A$ ("n realizes A"), where A is an arithmetical formula, n a natural number. The intuition behind this notion is "n hereditarily encodes the information about the realization of existential quantifiers and disjunctions in A".

In the version of arithmetic we have in mind, the formulas are constructed from term equations $t = s$ using logical operators $\vee, \wedge, \rightarrow, \neg, \forall, \exists$. We write $n \bullet m$ for "the algorithm with code n is applied to argument m", and $(n)_0$, $(n)_1$ are the decodings of n when n is viewed as the code of a pair of natural numbers. Let us look at some clauses.

- $n \,\underline{\mathrm{r}}\, (t = s)$ holds if and only if $t = s$ is true.
- $n \,\underline{\mathrm{r}}\, (A \vee B)$ holds if either $(n)_0 = 0$ and $(n)_1 \,\underline{\mathrm{r}}\, A$ or $(n)_0 \neq 0$ and $(n)_1 \,\underline{\mathrm{r}}\, B$.
- $n \,\underline{\mathrm{r}}\, \exists x A(x)$ if $(n)_1 \,\underline{\mathrm{r}}\, A((n)_0)$.
- $n \,\underline{\mathrm{r}}\, (A \rightarrow B)$ if, whenever $m \,\underline{\mathrm{r}}\, A$, then $n \bullet m$ is defined and $n \bullet m$ realizes B.

This may be seen as a (fairly crude) modeling of the proof interpretation, where the notion of constructive proof is stripped down to "explicit information coded by a number concerning the realization of disjunctions and existential quantifiers". The *construction* in the proof interpretation of an implication is in this model interpreted as a (partial) algorithm, or a partial recursive function. Proofs of term equations $t = s$ contain no interesting information about realizing $\vee$ or $\exists$, hence it is irrelevant which numbers we take to realize $t = s$, once one knows that $t = s$ holds.

It looks at first sight as if Kleene's notion of realizability is straightaway inspired by the proof interpretation. However, the historical development is not quite that straightforward. According to Kleene's own remarks in a retrospective paper from 1971 ([Kle73]), he derived his initial inspiration from the finitistic reading of disjunctions and existential statements as "incomplete communications", explained in considerable detail in the standard work by Hilbert and P. Bernays (actually written by Bernays), "Grundlagen der Mathematik" the first volume of which had appeared in 1934.

To explain the situation, this requires a digression. In the first quarter of this century, Hilbert formulated with increasing precision his programme of saving classical mathematics from the paradoxes by codifying existing mathematical practice in a formal system, which then might be regarded as a combinatorial structure, and for which one could hope to prove the consistency (the impossibility of deriving something like $0 = 1$) by elementary combinatorial ("finitistic") means. The body of finitistic mathematics is assumed to be justified by its intuitive evidence and non-controversial nature. Brouwer claimed that this approach had been inspired by the distinction between language and metalanguage made in his thesis, and indeed I believe that Hilbert's programme is for some of its ideas indebted to Brouwer, although this has never been acknowledged in publications of the Hilbert school. (Needless to say, Hilbert's aims were completely different from those of Brouwer.) Initially it was believed that finitistic mathematics more or less cowincided with intuitionistic mathematics. However, as pointed out by Bernays in writing ([Ber35]) this was not correct: finitistic mathematics is more restrictive, not permitting general abstract notions such as "arbitrary number theoretic function". Hence also the proof interpretation as presented above is not immediately meaningful from a finitistic point of view.

Instead, as explained by Bernays in the first volume of [HB34], one could explain certain logical operators in statements as "incomplete communications" which had to be completed to obtain their finitistic meaning, for example:

- Completing the assertion $A \vee B$ means to assert either A or B.
- Completing the assertion $\exists x A(x)$ is done by providing some d in the range of x such that $A(d)$.
- The assertion $\neg\forall x A(x)$ is completed by providing a d such that $\neg A(d)$.
- To assert $\forall x A(x)$ we must have a method for establishing $A(d)$ for any d in the range of x.

Clearly, also on this explanation there are no grounds to assume that $\forall x A(x) \vee \neg\forall x A(x)$ holds generally. No general explanation of $A \to B$ is given except as $\neg A \vee B$.

So Kleene needed another idea for implication. In [Kle73] he wrote:

> The second clue which I consciously tried to use in 1941 was Heyting's "proof-interpretation" as it is rendered in [Hey34, p. 14]. [...] In my first trial of a definition of realizability, I handled implication in a way based on this idea. But the resulting definition did not "work" [...]. Thus Heyting's proof-interpretation failed to help me to my goal.
>
> In my second trial, a few weeks later, of a definition of realizability, I thought of an implication "$A \to B$" in terms of an effective process for converting information a to complete "A" into information b to complete "B". This idea "worked" when I took into account that the effective process need only lead to a number b when applied to a number a which does complete "A".

What is remarkable in this quotation is that Kleene did not view his clause for implication as an analogue of Heyting's clause for implication.

Nevertheless, from our present-day perspective, it seems obvious that realizability may be seen as a model of the proof interpretation, where the collection of proofs of a statement is modeled by a collection of realizing numbers of the statement; and one may think of realizing numbers as "skeletons" of proofs, in which only information concerning the disjunctions and the existential statements has been retained.

Many variants of realizability have been devised; one of these, $\underline{\mathsf{q}}$-realizability[2], enables us to extract from a constructive proof in, say, intuitionistic first-order arithmetic of

$$\forall y \exists x\, A(y, x)$$

an algorithm, given by a code number m, say, such that in intuitionistic first-order arithmetic

$$\forall y\, A(y, m\bullet y).$$

This may be used for a proofs–as–programs method, according to which algorithms are found by giving a constructive proof of a statement that is the specification of a problem.

[2] $\underline{\mathsf{q}}$-realizability may be described as realizability + truth, that is to say $x\,\underline{\mathsf{q}}\,A \to A$ always holds. This is achieved by strengthening the clause for implication $n\,\underline{\mathsf{q}}\,(A \to B)$ by adding "and $A \to B$".

An interesting aspect of Kleene's realizability is that it gives a classical interpretation to formalisms based on intuitionistic logic plus additional principles which are in formal contradiction with *classical logic*. For example, to first-order arithmetic based on intuitionistic logic one may consistently add the "intuitionistic form of Church's thesis":

> "For arbitrary arithmetical A, if $\forall n \exists m A(n, m)$ (n, m natural numbers), then there is an *algorithm* f such that $\forall n A(n, f(n))$."

This principle is easily seen to be incompatible with classical logic. Another interesting principle appears when we consider the extension to second-order arithmetic, i.e. a formalism where we can quantify over *sets* of natural numbers. Writing (t, s) for the number coding the pair t, s, each set Y may also be regarded as coding a set of pairs (a binary relation). Extra clauses for realizability include:

- $n \,\underline{\mathrm{r}}\, X(t)$ if $X^*(n, t)$, where X^* is assumed to be a set variable assigned 1–1 in a fixed manner to the set variable X. (To a variable set X we assign a variable set of pairs X^*, so that we may read $X^*(t, s)$ as "t realizes Xs".)
- $x \,\underline{\mathrm{r}}\, \forall X A(X)$ if $\forall X^*(x \,\underline{\mathrm{r}}\, A(X))$. More informally: a number n realizes a property $A(X)$ for all sets X if for all sets of pairs X^* (representing arbitrary realization predicates) n realizes $A(X)$.

A remarkable consequence of this definition is that the following highly non-classical *uniformity principle* is realized:

$$\forall X \exists n A(X, n) \rightarrow \exists n \forall X A(X, n)$$

Intuitively this principle says "if for each X there exists a number n such that $A(X, n)$ holds, then there is a single number n such that for all X we have $A(X, n)$". Classically this is obviously false: take $A(X, n)$ to be the property "$n = 0$ and X is empty, otherwise $n = 1$ and X is not empty". One may convince oneself of the plausibility of this principle from an intuitionistic point of view as follows.

Arbitrary sets of natural numbers are very "hazy", indefinite objects; there is not a single element for which we can with certainty determine whether it belongs to the set or not. (Think of the set which contains all natural numbers if the Riemann hypothesis is true, otherwise it is empty. Intuitionistically, being element of this set is a well defined mathematical property, but here is not a single number (at present) for which we can guarantee that it belongs to the set, or for which we know for certain that it does not belong to the set. So if we can find a natural number n for each set X such that $A(X, n)$, the only way we can imagine for assigning *always* such a very definite object as a natural number to something as indefinite as the arbitrary set, is in a trivial way: assign the same number to all sets.

The definition of realizability for second-order arithmetic first appears in [KT70], and the uniformity principle is explicitly formulated in [Tro73].

Higher-order arithmetic is a strengthening of second-order arithmetic, where we can quantify not only over numbers and sets of numbers, but also over sets of sets of numbers, over sets of sets of sets of numbers, etc. Realizability for second-order arithmetic is a first step towards realizability for higher-order arithmetic; and realizability for higher-order arithmetic has led towards algorithmic models of the polymorphic lambda calculus; we shall return to this later.

Another possible use one can make of realizability is in finding in a systematic way constructive equivalents to classical definitions.

Perhaps it is illustrative to give also an example of an aspect of the proof interpretation which has generated technical work and foundational discussion, without any decisive results. In other words, so far this has turned out to be a dead end.

G. Kreisel ([Kre62, Kre65]) proposed to refine the proof interpretation in the following way:

- the relation "p proves A" ought to be decidable;
- a proof of an implication $A \to B$ should consists of a pair (p, f), where f is a construction transforming proofs of A into proofs of B, and p is an argument showing that f has this property.

One reason for this sharpening is that one wants to explain "p proves A" for compound A in terms of notions which are conceptually essentially simpler. Hence the insistence on decidable proof predicates; then one may require that the p in the clause for implication above is a proof of an especially simple kind, not itself involving proofs of implications or universally quantified statements, but rather something like the verification of a term equation with free variables by a straightforward computation.

3 The work of Gentzen

Before I come to the history of the "formulas–as–types" idea, I should say a few words about the work of the German logician Gerhard Gentzen (1909–1945), which constitutes a link between the proof interpretation and formulas-as-types. In 1935 Gentzen's paper "Untersuchungen über das logische Schliessen" was published. The paper has had, and still has, an enormous impact, especially in proof theory, but also in subjects such as theorem-proving. Gentzen's motivation derived from the problems in the foundations of mathematics. In his paper he compared three styles of formalizing first-order (intuitionistic and classical) logic: Hilbert-style (as used in formulations of Hilbert's programme, and in the influential book [HA28]), natural deduction, and what we now often call "Gentzen systems". Gentzen was not the first to formulate natural deduction, but he did it for the full language containing all the usual logical operators $\vee, \wedge, \rightarrow, \neg, \forall, \exists$, and also for intuitionistic logic, in a convenient presentation. The Gentzen systems were new; the reason for introducing them was that they were technically more manageable than the other two.

In each of these formalizations, we may think of the formal proofs as trees, with the conclusion at the bottom (the root of the tree), and the axioms (in the case of Hilbert-style and Gentzen systems) or assumptions (in the case of natural deduction) at the top nodes (the leaves of the tree). The transition from premises, A, B say, to conclusion C is marked by a horizontal line, so

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

I shall not discuss the Gentzen systems here, since they are not directly relevant to our story. The natural deduction rules for implication logic will be exhibited later. The format of Hilbert-type systems is axiom schemata plus deduction rules; there are no open assumptions which may be discharged, as in natural deduction. For example, intuitionistic implication logic may be axiomatized by the following two schemata:

$$A \to (B \to A)$$
$$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$$

(that is to say, for all formulas A, B, C these are axioms), and a single rule, *modus ponens*,

$$\text{If } A \to B \text{ and } A, \text{ then } B.$$

Gentzen proved the equivalence (w.r.t. the set of derivable formulas) between the three formalizations; moreover, for the Gentzen systems he established a very important property ("Hauptsatz"): any proof of a statement of predicate logic could be transformed into a "direct proof", a proof without detours so to speak, of the same statement. In direct proofs only subformulas of the conclusion appeared, there were no detours via more complex statements. This property was proved by showing that the so-called Cut rule could always be eliminated from a proof, hence Gentzen's Hauptsatz is also known as "Cut elimination". The corresponding process of eliminating detours for natural deduction is usually called normalization, and this was proved in a direct way by D. Prawitz in 1965. Gentzen analyzed the natural deduction rules as follows.

> "The introductions represent, as it were, the 'definitions' of the symbols concerned, and the eliminations are no more, in the final analysis, than the consequences of these definitions"

Gentzen's idea may be readily understood by reference to the proof interpretation, and it may safely be assumed that Gentzen was aware of this, since he clearly showed to have grasped the essentials of the ongoing foundational debate. Applied to the rules for implication this yields

$$\dfrac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \to B}$$

If we have a way of getting from assumption A to conclusion B, we have in fact established $A \to B$ (without assumption A). $[A]$ indicates zero or more occurrences of an open hypothesis of the form A.

$$\frac{A \to B \quad A}{B}$$

If we have $A \to B$, this means that we have a method for getting from A to B; and we have an argument for A; hence B. So this rule may be said to be justified by the fact that the introduction rule for $\to$ tells us that an implication is given by 'a method for getting from A to B'.

4 Formulas-as-types

The expression "formulas–as–types" is widely used, but its precise meaning is not sharply delimited, since various authors differ in their interpretation. It seems to me that one may distinguish at least three aspects (or if you like, "variants") of the notion, as follows. More detailed explanations will follow afterwards. Summing up, one may distinguish the following aspects in the formulas–as–types idea (unexplained terms will be explained later on).

(A) Noting the existence of a parallel between "proof of a formula" and "element of a type (set)", and beyond this the observation that the implicational formulas provable in intuitionistic implication logic correspond exactly to the inhabited types in the basic theory of type assignment.

(B) A strengthening of the preceding: the isomorphism between simple type theory and natural deduction for minimal (= intuitionistic) implication logic, with β-reduction corresponding to normalization on the deduction side. This isomorphism may be extended to full predicate logic by a suitable extension of simple type theory.

(C) The exploitation of the similarities between "proof of a formula" and "element of a type" by giving a uniform presentation of rules for proving propositions and elementhood of types.

These three aspects of "formulas–as–types" may be distinguished by the abbreviations FAT(A), FAT(B) and FAT(C) respectively. Before discussing the history of this idea, I shall first try to explain formulas–as–types, version (B), more particularly for intuitionistic implication logic on the one hand, and simple type theory on the other.

The key idea is the following: intuitionistically, propositions are determined by the set of their proofs. We may say that a proof p belongs to a proposition A if p is a proof of A. Hence propositions and the proofs establishing them correspond to sets with their elements.

The analogy does not stop here: the operation of forming the set $A \to B$ of functions from A to B corresponds to the formation of an implication $A \to B$ with its proofs being constructions mapping the proofs from A to proofs of B. In a similar way (going for a moment beyond implication logic), the formation of a cartesian product $A \times B$ corresponds to the set of proofs of a conjunction $A \wedge B$, which are pairs of proofs of A and B respectively.

However, the sets in this analogy do not correspond so much to sets as we know them from axiomatic set theory, but to the types (which may be regarded as special sets) in the simple theory of types. In the simple theory of types we have a collection of unspecified basic types (denoted by type variables) and some operations for constructing new types from given ones; in particular the function-type constructor will always be present. The rules of simple type theory tell us what the objects are which belong to a given type.

In the formalism of simple type theory ($\boldsymbol{\lambda}_{\to}$ for short) we have type terms (expressions denoting types) and individual terms, expressions denoting elements of types. Each individual term has a fixed type, and all the subterms of an individual term have a fixed type as well. So individual terms in $\boldsymbol{\lambda}_{\to}$ always appear as $t\colon A$, where A is a type expression. This is called rigid typing. Simple type theory was first formulated by A. Church in 1940.

For function types, the canonical method for constructing an element is by abstraction: let $t[x\colon A]\colon B$ be a term describing the construction of an element of type B from an arbitrary element x of type A, then $\lambda x.t[x\colon A]\colon A{\to}B$ is the term t *as function of* x, an element of type $A \to B$. Since all elements of $A \to B$ are assumed to be functions, we of course also have application[3]: $t\colon A{\to}B$ applied to $s\colon A$ yields a term $(t\colon A{\to}B)(s\colon A)\colon B$.

$\boldsymbol{\lambda}_{\to}$ has standard set-theoretic models; in these models, the types are interpreted by sets, and terms $t\colon A$ by elements of the set interpreting A. If we write $[\![\ ]\!]$ for the interpretation function, and ρ for the environment interpreting the individual and type variables, then $[\![A \to B]\!]_\rho$ is the set of

[3]In practice, when discussing simple type theory, one often drops a lot of the type information in exhibiting terms; otherwise expressions soon become unreadable.

all set-theoretic functions from $[\![A]\!]_\rho$ to $[\![B]\!]_\rho$, etc.

The rules for constructing terms in the simple type theory clearly correspond to the natural deduction rules for introducing an implication and eliminating an implication. Therefore the following rule may be read either as a schematic rule of implication introduction, where the terms appearing at each node of the proof tree are nothing but a linear notation for the proof obtained hitherto, or as a schematic representation of the rule of term formation for function types.

$$\frac{\begin{array}{c}[x\colon A]\\ \vdots\\ t[x\colon A]\colon B\end{array}}{\lambda x\colon A.t[x\colon B]\colon (A \to B)}$$

Similarly for the elimination rule:

$$\frac{t\colon A{\to}B \qquad s\colon A}{(t\colon A{\to}B)(s\colon A)\colon B}$$

But the analogy goes still farther. Because of the meaning of lambda-abstraction, we have a rule of β-conversion:

$$(\lambda x.t[x])s = t[x/s]$$

(I have refrained from exhibiting the types, otherwise the notation gets so heavy.) Replacing in a term a subterm $(\lambda x.t)s$ by $t[x/s]$ is called a β-reduction step. If we replace a subterm of the form $\lambda x.t[x]s$ by $t[x/s]$, what corresponds to this in prooftrees? It means that we can replace a tree as on the left by the tree on the right, thereby removing a detour:

$$\frac{\dfrac{\begin{array}{c}[A]^x\\ \boxed{1}\\ B\end{array}}{A \to B}\,-x \qquad \begin{array}{c}\boxed{2}\\ A\end{array}}{\begin{array}{c}B\\ \boxed{3}\end{array}} \qquad\qquad\qquad \begin{array}{c}\boxed{2}\\ [A]\\ \boxed{1}\\ B\\ \boxed{3}\end{array}$$

The isomorphism as explained above covers only implication logic. But there is no difficulty in extending it to full predicate logic, by extending $\boldsymbol{\lambda}_{\to}$ with new type constructors, one constructor for each new logical operation, and new term-formation rules, one rule for each natural deduction rule.

H.B. Curry

The first clear expression of the formulas–as–types idea in the version FAT(A) was formulated by H.B. Curry in [CF58], with a hint of it in [Cur42]. Curry considered a calculus for assigning types to untyped lambda-terms[4]. The untyped terms are in a sense pure algorithms, without specified domain and range; or to put it otherwise, the lambda-calculus deals with a universe where everything can be both operation and argument. But if we specify a subdomain for an argument, what can we say about the range of an operator?

Curry first considered a system of type assignment corresponding on the logical side to a Hilbert-type axiomatization of intuitionistic implication logic. That is to say, he had two axioms assigning types to the constants K and S:

$$\begin{aligned}&\mathrm{K}\colon (A \to (B \to A)),\\ &\mathrm{S}\colon (A{\to}(B{\to}C)) \to ((A{\to}B){\to}(A{\to}C)),\end{aligned}$$

where A, B, C are arbitrary type expressions, together with an analogue of modus ponens: if $t\colon A \to B$ and $s\colon A$, then $ts\colon B$. In the lambda calculus we may define the constants as $\mathrm{K} \equiv \lambda xy.x$, $\mathrm{S} \equiv \lambda xyz.xz(yz)$. But inspired by Gentzens's paper from 1935 already mentioned, Curry also considered type-assignment systems corresponding to natural deduction and to Gentzen systems. Thus in the natural deduction formulation, Curry's rules are as follows:

$$\Gamma, t\colon A \vdash t\colon A$$

$$\frac{\Gamma, t\colon A{\to}\, B \qquad \Gamma \vdash s\colon A}{\Gamma \vdash ts\colon B} \qquad\qquad \frac{\Gamma, x\colon A \vdash t\colon B}{\Gamma \vdash \lambda x.t\colon A{\to}B}$$

(x not occurring free in Γ.) These rules look just like another way of writing the rules of implication logic as exhibited above; the Γ in $\Gamma \vdash t\colon A$ is a list of typed variables representing the open assumptions in the derivation of $t\colon A$. There is an important difference however. Here the types are not considered to be part of the lambda terms, and the subterms of the lambda terms have no types attached to them.

So Curry knew how to exploit the formulas-as-types parallel for his theories of type assignment. In particular he uses the idea of Cut elimination,

[4] Curry also considered combinatory logic, but in order not to complicate our story too much, we mostly concentrate on type assignment for the lambda calculus.

taken from Gentzen's work. One might also say that Curry proves indirectly, via the correspondence between natural deduction and Gentzen systems, a normalization theorem for natural deduction for implication logic; as far as I know, this is the first published proof (A.M. Turing had a proof of normalization for simple type theory as early as 1942, but this was only published by Gandy in 1980). However, Curry does not formulate the result as a result on intuitionistic implication logic. For Curry, it is a normal form theorem for typable lambda terms.

It should be pointed out that formulas–as-types has much more impact, and more interesting consequences, when formulated for natural deduction, than when it is formulated for Hilbert-type axiomatizations only[5].

A few years after [CF58] appeared, Curry published a standard textbook [Cur63], which contains a detailed treatment of natural deduction and Gentzen calculi, both for intuitionistic and classical logic. It is a remarkable fact, that in this book Curry does not discuss the formulas-as-types parallel[6].

Why did Curry not make use of the formulas-as-types parallel in his study of deduction? I can only guess. One possible reason is a technical one. As we have explained above, the parallel becomes an isomorphism, if one compares natural deductions on the one hand, with simple type theory (with rigid typing) on the other hand. But Curry was interested not in simple type theory, but in type-assignment systems; and then the parallel is not as simple and clear-cut, and more difficult to formulate precisely. Another, perhaps more important reason may be the following: on the deduction side, "formulas-as-types" did not answer any questions Curry was interested in. Summarizing, the insights FAT(A) are present in Curry's work, but not FAT(B).

D. Prawitz

[5]For combinatory logic the parallel with Hilbert-type axiomatizations of implication logic is the most natural one. But although there may be formulated a normalization theorem for Hilbert-type logic, inspired by the notion of weak reduction for combinatory logic, this is not as interesting and useful as the normalization for $\boldsymbol{\lambda}_{\rightarrow}$ and natural deduction. In particular, there is no manageable subformula property.

[6]In fact, just as in [Gen35], numbered assumption classes appear in Curry's book, but all examples are compatible with the assumption that open hypotheses are discharged as early as possible, a convention which simplifies life as long as one is only interested in derivability, not in the derivations themselves, but which leads to an imperfect match with simple type theory. Therefore it is not clear, from looking at the texts, whether Gentzen and Curry really were thinking of a natural deduction calculus isomorphic to simple type theory.

In 1965, D. Prawitz in a sense "rehabilitated" natural deduction. Gentzen considered natural deduction as the most basic type of formalization, since its rules correspond rather closely to steps in informal reasoning; but for technical work, more precisely, for obtaining metatheorems, he found the Gentzen systems LJ and LK more manageable. Literally, Gentzen says

> "In order to be able to enunciate and prove the Hauptsatz, I had to provide a logical calculus especially suited to the purpose. For this the natural calculus proved unsuitable. For, although it already contains the properties essential to the validity of the *Hauptsatz*, it does so only with respect to its intuitionist form, in view of the fact that the law of the excluded middle, as pointed out earlier, occupies a special position in relation to these properties."[7]

Prawitz' starting point was the formulation where open hypotheses are always discharged as early as possible, but he also described a version with "assumption classes" which may be said to be isomorphic to simple type theory. (He needed such a version for his treatment of the modal logic **S4**.) On the other hand, there is no mention of Curry's work in [CF58]. Prawitz showed that natural deductions could be *normalized*; in a normal proof, a logical constant which has just been introduced is never eliminated at the next rule. Thus normal proofs also have a claim to be called "direct": in this case too, detours via more complex formulas do not occur, all formulas involved are subformulas of the conclusion.

W.A. Howard

The first explicit formulation of "formulas-as-types" in the form FAT(B) is found in a paper by W.A. Howard ([How80]) which widely circulated since 1969, but which appeared in print only in 1980. Howard's aim was foundational: he wanted to develop a notion of construction suitable for the interpretation of intuitionistic mathematics[8]. Howard not only discusses formulas–

[7]The quote made me speculate whether perhaps Gentzen proved normalization (see below) for the intuitionistic case. He certainly did not publish it.

[8]Correspondence between G.Kreisel and Howard suggests that it was P. Martin-Löf who pointed out that the most pregnant expression of formulas–as–types, FAT(B), is obtained when the logic is based on natural deduction. (I owe this information to H.R. Jervell.) Cf. also the footnote in [How80] added by Howard in 1979. Apparently critical remarks

as–types for propositional logic, but also for intuitionistic arithmetic. By FAT(B), normalization in $\boldsymbol{\lambda}_{\to}$ is *the same* as in natural deduction[9].

N.G. de Bruijn

There is an independent discovery of the formulas-as-types idea in the form FAT(C), by N.G. de Bruijn ([Bru70]). De Bruijn and his group developed a language AUTOMATH, designed for the mechanical checking of mathematical proofs. De Bruijn's only clue was the intuitionistic interpretation of implication, as formulated by Heyting for his proof interpretation. For a long time the results on AUTOMATH were not very accessible, since a lot of it could be found in internal reports and Ph.D. theses only; but recently all the interesting material on AUTOMATH was brought together in one volume ([NGdV94]).

H. Läuchli

Independently also, H. Läuchli ([Läu70]) used the ideas of formulas–as–types for obtaining a completeness proof for intuitionistic predicate logic, relative to a notion which might be regarded as a version of realizability.

Martin-Löf's type theories

Formulas–as–types in the form FAT(B) and FAT(C) was the guiding idea behind the construction of an intuitionistic type theory by P. Martin-Löf. This type theory appeared in several versions. The first (inconsistent, impredicative) version dates from 1971, but the first published (consistent and predicative) version appeared in 1975. Martin-Löf's original aim was to give a foundation for constructive mathematics, but in 1979 ([ML82]) he proposed a version of his type theory as a language for programming. The formulas–as–types idea in these theories results in bringing together, for example, induction over the natural numbers and definition of arithmetical functions by recursion, as instances of the same set of rules for the type of the natural numbers. (The same idea applies also to other data types.)

There is a strong flavour of realizability in the Martin-Löf type systems; one may very well think of these systems as incorporating an abstract version of Kleene's realizability.

Thus, if $t\colon (\exists x\colon A)B(x)$, t must be a pair (t_0, t_1) such that $t_1\colon B(t_0)$ and $t_0\colon A$. It is this feature which makes it possible to use Martin-Löf's theories

by Kreisel concerning the usefulness of formulas–as–types w.r.t. the intended aim made Howard refrain from publishing his note.

[9]Howard states that he owes this insight to W.W. Tait, and refers to the paper [Tai65].

for proofs–as–programs (or proofs–as–algorithms); if $(\exists x\colon A)B(x)$ is a proposition describing a specification, a proof of $(\exists x\colon A)B(x)$ in a Martin–Löf type system appears as a deduction with conclusion

$$t\colon (\exists x\colon A)B(x)$$

and from t we can extract the second component t_0 for the required x. See also the next section.

5 From constructivism to computer science

In this final section I shall briefly describe two applications of the ideas discussed above, a theoretical one and a more practical one.

The polymorphic lambda calculus

The polymorphic lambda calculus, usually designed by $\boldsymbol{\lambda}2$ or as "system F" was first discovered by J.-Y. Girard in 1972 ([Gir72]), and rediscovered some years later by Reynolds ([Rey74]). $\boldsymbol{\lambda}2$ is a powerful extension[10] of $\boldsymbol{\lambda}_{\to}$. The simplest way of describing it is to say, relying on FAT(B), that it is isomorphic to a natural deduction formulation of intuitionistic second-order propositional logic with implication and universal quantifiers over propositions. The rules for implication are as before, but the rules for the quantifiers are (P, Q propositional variables)

$$\frac{A}{\forall Q\, A[P/Q]} \qquad \frac{\forall Q\, A}{A[Q/B]}$$

On the left, A has been derived with open assumptions not containing P free; on the right, $[Q/B]$ denotes substitution of formula B for the propositional variable Q. In terms of types, this means that we have an abstraction operator Λ over type variables corresponding to the rule of $\forall$-introduction, and an application of terms to types corresponding to the $\forall$-elimination. For example, $(\lambda x\colon A)(x\colon A)$ is the identity function for type A, and

$$t \equiv \Lambda P.(\lambda x\colon P)(x\colon P)$$

[10]In $\boldsymbol{\lambda}2$ all recursive functions which are provably total in classical or intuitionistic analysis, i.e., second-order arithmetic with impredicative comprehension, are representable by terms.

is the universal identity function, which specializes, when applied to type A, to $tA = (\lambda x{:}\, A)(x{:}\, A)$. Generally, $(\Lambda P.t)A = t[P/A]$. If we want to extend the set-theoretic models of $\boldsymbol{\lambda}_{\rightarrow}$ to $\boldsymbol{\lambda}2$, we have to pick a domain for the range of the propositional variables, and then the objects of type $\forall P\, A[P]$ are to be all functions assigning to each P in the propositional domain an object in type $A[P]$.

Reynolds showed in 1984 ([Rey84]) that such a set-theoretic model for $\boldsymbol{\lambda}2$ could not exist. However, Reynolds argument made essential use of *classical* logic. If we move to an alternative universe, with intuitionistic logic, there are other possibilities. One such solution is described in the paper by A.M. Pitts ([Pit87]), where it is shown that in the effective topos Eff we can construct a model for $\boldsymbol{\lambda}2$ based on a set-theoretic interpretation, in an intuitionistic universe. A topos may be seen as a category with extra structure, such that it is possible to interpret intuitionistic higher-order logic in it; one might say that toposes are just category-theoretic equivalents of models for intuitionistic higher-order logic. Eff is a topos which might be described as the category-theoretic version of Kleene realizability extended to intuitionistic higher-order logic; it was introduced by J.M.E. Hyland in 1980 ([Hyl82]). For the construction of a model of $\boldsymbol{\lambda}2$ in Eff it is crucial that a generalization of the Uniformity Principle (mentioned in the section on realizability) holds.

There are more nice examples of the fruitfulness of formulas–as–types in theory; for example, in [BCD83] where intersection types are added to a standard type-assignment system, the theory of (slightly extended) natural deduction proofs is used to prove conservativeness of the extended system over the original system, that is to say the extension does not prove new statements in the old language.

Formulas-as-types, programming, and proofs–as–algorithms

The formulas-as-types idea in the version FAT(C) has served as a source of inspiration in the construction of several languages designed with the proofs–as–algorithms paradigm in mind. A survey may be found in [NGdV94, pp. 8–12].

For example, there is the "calculus of constructions" by Coquand and Huet, combining ideas from Girard's $\boldsymbol{\lambda}2$ and AUTOMATH ([Coq85, CH88]). Luo ([Luo89]) designed the "extended calculus of constructions". For these languages implementations have been designed, such as LEGO and Coq, which can be used for interactive proof construction.

Close to AUTOMATH is LF ("Logical Framework") developed in Edinburgh ([HHP87]); this system has also been implemented.

As mentioned above, Martin-Löf proposed a version of his type theory as a functional programming language, embodying the programs–from–proofs paradigma. In [NPS90] this is elaborated in detail.

In practice, one has to compromise with the purity of the original idea of completely amalgamating propositions and types: it is not always useful to insist on "types = propositions". For example, if one considers a subset of a given type A, one is not always interested in the information which provides the proof that a given element of A belongs to the subset, but only in the truth of this statement. Thus one has to relax the rigid schema of introduction and elimination rules, in exchange for a more convenient manipulation of subsets. Similarly, in AUTOMATH, where one is using classical logic, one is not interested in the details of a proof of some proposition, so one introduces a principle of "irrelevance of proof", in fact a coarse equivalence relation on the proofs of propositions, which tells us that any two proofs of a proposition are to be regarded as equivalent (cf. Zucker's paper [Zuc77]). But for this principle one has to distinguish between types and propositions. So the formulas-as-types parallel is then relaxed; what remains from this idea is the general scheme for declaring what elements of a type (proposition are), with elimination rules as a counterpart to the introduction rules.

This illustrates that ideas taken from theory are not used as such, but in applications nearly always have to be adapted and modified to the purpose at hand.

There is also a system for interactive proof-construction based on Martin-Löf's ideas, namely Nuprl ([Con86]).

References

[BCD83] H. P. Barendregt, M. Coppo, and M. Dezani–Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48:931–940, 1983.

[Ber35] P. Bernays. Sur le platonisme dans les mathématiques (French). *L'Enseignement Mathématique*, 34:52–69, 1935. German translation in: P. Bernays, *Abhandlungen zur Philosophie der Mathematik*, Wissenschaftliche Buchgesellschaft, Darmstadt 1976, 62–78.

[Bro07] L. E. J. Brouwer. *Over de Grondslagen der Wiskunde (Dutch).* PhD thesis, Universiteit van Amsterdam, 1907. Translated in [Bro75, pp. 11–101].

[Bro08] L. E. J. Brouwer. De onbetrouwbaarheid der logische principes (Dutch). *Tijdschrift voor Wijsbegeerte*, 2:152–158, 1908. Translated in [Bro75, pp. 107–111].

[Bro75] L. E. J. Brouwer. *Collected Works, Vol. 1.* North-Holland Publ. Co., Amsterdam, 1975. Edited by A. Heyting.

[Bru70] N. G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics, 125, pages 29–61. Springer-Verlag, Berlin, Heidelberg, New York, 1970.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic I.* Studies in Logic and the Foundations of Mathematics. North-Holland Publ. Co., Amsterdam, 1958. 2nd edition 1968.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[Chu40] A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[Con86] R. L. Constable et. al. *Implementing Mathematics with the NuPRL Proof Development System.* Prentice–Hall, Englewood Cliffs, 1986.

[Coq85] T. Coquand. *Une théorie des constructions.* PhD thesis, Paris, Université Paris VII, 1985.

[Cur42] H. B. Curry. The combinatory foundations of mathematical logic. *The Journal of Symbolic Logic*, 7:49–64, 1942.

[Cur63] H. B. Curry. *Foundations of Mathematical Logic.* McGraw-Hill, New York, 1963. Also published by Dover, New York 1977.

[Gan80] R. O. Gandy. An early proof of normalization by A. M. Turing. In *[SH80]*, pages 453–455. Academic Press, 1980.

[Gen35] G. Gentzen. Untersuchungen über das logische Schliessen I, II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. Translation in *[Gen69]*, 68–131.

[Gen69] G. Gentzen. *Collected Works.* North-Holland Publ. Co., Amsterdam, 1969. English translation of Gentzen's works, edited by M.E. Szabo.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[Göd33] K. Gödel. Eine Interpretation des intuitionistischen Aussagenkalküls. *Ergebnisse eines mathematischen Kolloquiums*, 4:39–40, 1933. Also, with translation, in vol. 1 of K. Gödel's Collected Works, Oxford University Press, 1986, pp. 300–303.

[HA28] D. Hilbert and W. Ackermann. *Grundzüge der theoretischen Logik.* Springer-Verlag, Berlin, Heidelberg, New York, 1928.

[HB34] D. Hilbert and P. Bernays. *Grundlagen der Mathematik, Bd. I.* Springer-Verlag, Berlin, Heidelberg, New York, 1934. 2nd edition 1968.

[Hey31] A. Heyting. Die intuitionistische Grundlegung der Mathematik. *Erkenntnis*, 2:106–115, 1931.

[Hey34] A. Heyting. *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie.* Springer-Verlag, Berlin, Heidelberg, New York, 1934. Reprinted 1974.

[Hey58] A. Heyting. Intuitionism in mathematics. In R. Klibansky, editor, *Philosophy in the Mid-Century. A Survey*, pages 101–115. La Nuova Italia Editrice, Firenze, 1958.

[HHP87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings of the second Symposium on Logic in Computer Science, Ithaca, N.Y.* Computer Society of the IEEE, Washington, D.C., 1987.

[How80] W. A. Howard. The formulae-as-types notion of construction. In *[SH80]*, pages 480–490, 1980. Circulated as preprint since 1969.

[Hyl82] J. M. E. Hyland. The effective topos. In A. S. Troelstra and D. van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, pages 165–216. North-Holland Publ. Co., Amsterdam, 1982.

[Kle45] S. C. Kleene. On the interpretation of intuitionistic number theory. *The Journal of Symbolic Logic*, 10:109–124, 1945.

[Kle73] S. C. Kleene. Realizability: a retrospective survey. In *[MR73]*, pages 95–112, 1973.

[Kol32] A. N. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35:58–65, 1932.

[Kre62] G. Kreisel. Foundations of intuitionistic logic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Methodology and Philosophy of Science*, pages 198–210. Stanford University Press, Stanford, Cal., 1962.

[Kre65] G. Kreisel. Mathematical logic. In T. L. Saaty, editor, *Lectures on Mathematical Logic, Vol. III*, pages 95–195. John Wiley and Sons, New York, 1965.

[KT70] G. Kreisel and A. S. Troelstra. Formal systems for intuitionistic analysis. *Annals of Mathematical Logic*, 1:229–387, 1970.

[Läu70] H. Läuchli. An abstract notion of realizability for which intuitionistic predicate calculus is complete. In A. Kino, J. R. Myhill, and R. E. Vesley, editors, *Intuitionism and Proof Theory*, pages 227–234. North-Holland Publ. Co., Amsterdam, 1970.

[Luo89] Z. Luo. ECC: an extended Calculus of Constructions. In *Proceedings of the fourth Annual Symposium on Logic in Computer Science*, pages 386–395. IEEE Computer Society Press, Washington D.C., 1989.

[ML75] P. Martin-Löf. An intuitionistic theory of types: predicative part. In H. E. Rose and J. E. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118. North-Holland Publ. Co., Amsterdam, 1975.

[ML82] P. Martin-Löf. Constructive mathematics and computer programming. In L. J. Cohen, J. Los, H. Pfeiffer, and K. P. Podewski, editors, *Logic, Methodology and Philosophy of Science 6*, pages 153–175. North-Holland Publ. Co., Amsterdam, 1982.

[MR73] A. R. D. Mathias and H. Rogers, editors. *Cambridge Summer School in Mathematical Logic*. Springer-Verlag, Berlin, Heidelberg, New York, 1973.

[NGdV94] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer, editors. *Selected Papers on Automath*. North-Holland Publ. Co., Amsterdam, 1994.

[NPS90] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory. An Introduction.* Clarendon Press, Oxford, 1990.

[Pit87] A. M. Pitts. Polymorphism is set-theoretic, constructively. In D. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Category Theory and Computer Science*, Lecture Notes in Computer Science, pages 12–39. Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[Rey74] J. C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium, Proceedings. Colloque sur la Programmation*, Lecture Notes in Computer Science 19, pages 408–425. Springer-Verlag, Berlin, Heidelberg, New York, 1974.

[Rey84] J. C. Reynolds. Polymorphism is not set-theoretic. In *Semantics of Data Types*, Lecture Notes in Computer Science 173, pages 145–156. Springer-Verlag, Berlin, Heidelberg, New York, 1984.

[SH80] J. P. Seldin and J. R. Hindley, editors. *To H. B. Curry: Essays on Combinatory logic, Lambda Calculus and Formalism.* Academic Press, New York, 1980.

[Tai65] W. W. Tait. Infinitely long terms of transfinite type. In J. N. Crossley and M. A. E. Dummett, editors, *Formal Systems and Recursive Functions*, Studies in Logic and the Foundations of Mathematics, pages 176–185. North-Holland Publ. Co., Amsterdam, 1965.

[Tro73a] A. S. Troelstra. Chapters I–IV. In A. S. Troelstra, editor, *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, pages 1–323. Springer-Verlag, Berlin, Heidelberg, New York, 1973.

[Tro73b] A. S. Troelstra. Notes on intuitionistic second order arithmetic. In *[MR73]*, pages 171–205, 1973.

[Zuc77] J. Zucker. Formalization of classical mathematics in Automath. In *Colloque International de Logique, Clermont-Ferrand, France 1975*, pages 133–145. CNRS, Paris, 1977.