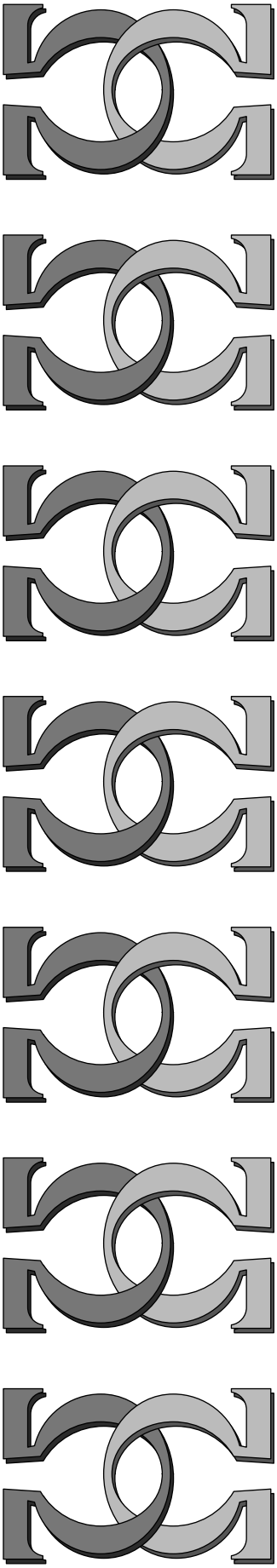

CDMTCS
Research
Report
Series

Constructive Mathematics, in Theory and Programming Practice

Douglas Bridges

Steve Reeves
Department of Mathematics
University of Waikato

CDMTCS-068
November 1997

Centre for Discrete Mathematics and
Theoretical Computer Science

Constructive Mathematics, in Theory and Programming Practice

Douglas Bridges and Steve Reeves
University of Waikato, Hamilton, New Zealand

November 24, 1997

Abstract

The first part of the paper introduces the varieties of modern constructive mathematics, concentrating on Bishop's constructive mathematics (BISH). It gives a sketch of both Myhill's axiomatic system for BISH and a constructive axiomatic development of the real line $\mathbf{R}$. The second part of the paper focusses on the relation between constructive mathematics and programming, with emphasis on Martin-Löf's theory of types as a formal system for BISH.

1 What is Constructive Mathematics?

The story of modern constructive mathematics begins with the publication, in 1907, of L.E.J. Brouwer's doctoral dissertation *Over de Grondslagen der Wiskunde* [18], in which he gave the first exposition of his philosophy of *intuitionism* (a general philosophy, not merely one for mathematics). According to Brouwer, mathematics is a creation of the human mind, and precedes logic: the logic we use in mathematics grows from mathematical practice, and is not some *a priori* given before mathematical activity can be undertaken.

It is not difficult to see how, with this view of mathematics as a strictly creative activity, Brouwer came to the view that the phrase "there exists" should be interpreted strictly and uniquely as "there can be constructed" or, in more modern parlance, "we can compute". In turn, this interpretation of existence led Brouwer to reject the unbridled use of the **Law of Excluded Middle (LEM)**, $P \vee \neg P$, in mathematical arguments. For example, consider the following statement, the **Limited Principle of Omniscience (LPO):**

$$\forall \mathbf{a} \in \{0,1\}^{\mathbf{N}} \ (\mathbf{a} = \mathbf{0} \vee \mathbf{a} \neq \mathbf{0}) \,. \tag{1}$$

Here, $\mathbf{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers, $\{0,1\}^{\mathbf{N}}$ is the set of all

binary sequences $\mathbf{a} \equiv (a_0, a_1, a_2, \ldots)$,

$$\begin{array}{rcl} \mathbf{a} = \mathbf{0} & \Leftrightarrow & \forall n \ (a_n = 0), \\ \mathbf{a} \neq \mathbf{0} & \Leftrightarrow & \exists n \ (a_n = 1). \end{array}$$

According to Brouwer's analysis, a proof of statement (1) would, for any $\mathbf{a} \in \{0,1\}^{\mathbf{N}}$,

- either demonstrate that each term of the sequence $\mathbf{a}$ equals 0
- or else construct (compute) a certain natural number N, and show that $a_N = 1$.

To see the power of such a proof, if it were available, we need only realise that, applied to the sequence $\mathbf{a}$ defined by

$$a_n = \begin{cases} 0 & \text{if } 2n+4 \text{ is a sum of two primes} \\ \\ 1 & \text{otherwise,} \end{cases}$$

it would, at least in principle, enable us to solve the Goldbach Conjecture[1]. The intervention of the Goldbach Conjecture here is not essential: were that conjecture to be resolved today, we could replace it in our example by any one of a host of open problems of mathematics, including the twin prime conjecture, the conjecture that there are no odd perfect numbers, and the Riemann Hypothesis. A Brouwerian proof of (1) would provide a method of literally incredible power and wide applicability; for this reason, Brouwer would not accept as valid mathematical principles either (1) or LEM, from which (1) is trivially deducible. In turn, he could not accept any classical proposition that constructively entails LEM, LPO, or some other manifestly nonconstructive principle.

It is important to stress here that, for Brouwer,

- mathematics precedes logic, which arises out of intuitionistic mathematical practice, and
- a careful, introspective analysis of the meaning of mathematical existence leads to the rejection of certain consequences of LEM, such as LPO, and therefore of LEM itself.

Passing over the intervening years, in which Brouwer struggled, perhaps too aggressively, to overcome the antipathy of Hilbert and his followers to intuitionistic mathematics,[2] we arrive at 1930, when Heyting, a former student of Brouwer, published axioms for the intuitionistic propositional and predicate

[1] This conjecture, first stated in a letter from Christian Goldbach to Euler in 1742, states that every even integer > 2 is a sum of two primes.

[2] See [45] for more details of the history of that period.

calculi. These axioms, which we shall describe shortly, have led to substantial developments in intuitionistic logic, but for the Brouwerians were of lesser importance than the mathematical activity from which they were abstracted.

From the 1940s there also grew, in the former Soviet Union, a substantial group of analysts, led by A.A. Markov, who practised what was essentially recursive mathematics using intuitionistic logic. Although this group accomplished much, the strictures of the recursive function theoretic language in which its mathematics was couched did not encourage its acceptance by the wider community of analysts, and perhaps also hindered the production of positive constructive analogues of traditional mathematical theories. An excellent reference for the work of the Markov School is [28].

By the mid–1960s it appeared that constructive mathematics was at best a minor activity, with few positive developments to show in comparison with the prodigious advances in traditional mathematics throughout the century. Indeed, many mathematicians were virtually ignorant of Brouwer's work outside classical topology, and those who knew something about it probably shared Bourbaki's view:

> The intuitionistic school, of which the memory is no doubt destined to remain only as an historical curiosity, would at least have been of service by having forced its adversaries, that is to say definitely the immense majority of mathematicians, to make their position precise and to take more clearly notice of the reasons (the ones of a logical kind, the others of a sentimental kind) for their confidence in mathematics. ([7], p. 38)

This situation changed dramatically with the publication, in 1967, of Errett Bishop's *Foundations of Constructive Analysis* [3]. Here was a major young mathematician, already holding a formidable reputation among functional analysts and experts in several complex variables, who had turned away from traditional mathematics to become a powerful advocate of a radical constructive approach. Moreover, the breadth and depth of mathematics in his monograph were breathtaking: starting with traditional calculus, Bishop gave a constructive development of a large part of twentieth century analysis, including the Stone-Weierstrass Theorem, the Hahn-Banach and separation theorems, the spectral theorem for selfadjoint operators on a Hilbert space, the Lebesgue convergence theorems for abstract integrals, Haar measure and the abstract Fourier transform, ergodic theorems, and the elements of Banach algebra theory. At a stroke, he refuted the long-held belief summarised in the famous words of Hilbert:

> Taking the principle of excluded middle from the mathematician would be the same, say, as proscribing the telescope to the astronomer or to the boxer the use of his fists. [25]

Although Bishop's work led to a renewed interest in constructive mathematics, especially among logicians and computer scientists (see the second part

of this paper), it would be idle to suggest that he convinced any but a few mathematicians to take up his challenge to work systematically within a constructive framework. Nevertheless, there have been substantial developments in Bishop-style constructive analysis since 1967, and, contrary to Bishop's expectations ([5], pp. 27-28), modern algebra has also proved amenable to a natural, thoroughgoing, constructive treatment [33].

Bishop's development **(BISH)** was based on a primitive, unspecified notion of **algorithm** and on the properties of the natural numbers:

> The primary concern of mathematics is number, and this means the positive integers. We feel about number the way Kant felt about space. The positive integers and their arithmetic are presupposed by the very nature of our intelligence and, we are tempted to believe, by the very nature of intelligence in general. The development of the positive integers from the primitive concept of the unit, the concept of adjoining a unit, and the process of mathematical induction carries complete conviction. In the words of Kronecker, the positive integers were created by God. ([3], p. 2)

By not specifying what he meant by an algorithm, Bishop gained two significant advantages over other approaches to constructivism.

- He was able to develop the mathematics in the style of normal analysis, without the cumbersome linguistic restrictions of recursive function theory.
- His results and proofs were formally consistent with Brouwer's intuitionistic mathematics (**INT**), recursive constructive mathematics (**RUSS**), and classical (that is, traditional) mathematics **(CLASS)**: every theorem proved in Bishop is also a theorem, with the same proof, in INT, RUSS, and CLASS.

Now, one point at which BISH is open to criticism is its lack of precision about the notion of algorithm (although it is precisely that lack of precision that allows it to be interpreted in a variety of models). But that criticism can be overcome by looking more closely at what we actually do, as distinct from what Bishop may have thought he was doing, when we prove theorems in BISH: in practice, we are doing *mathematics with intuitionistic logic*, and we observe from our experience that *the restriction to that logic always forces us to work in a manner that, at least informally, can be described as algorithmic.* The original algorithmic motivation for our approach led us use intuitionistic logic, which, in turn, seems to produce only arguments that are entirely algorithmic in character. In other words, *algorithmic mathematics appears to be equivalent to mathematics that uses only intuitionistic logic.*[3] If that is the case—and all

[3]Is this Bishop's "secret still on the point of being blabbed" ([3], epigraph)?

the evidence of our experience suggests that it is—then we can carry out our mathematics using intuitionistic logic on any reasonably defined mathematical objects, not just some special class of so–called "constructive" objects.

To emphasise this point, which may come as a surprise to readers expecting here some version of hard-core constructivism, our experience of doing constructive mathematics suggests that we are

- dealing with normal mathematical objects, and
- working only with intuitionistic logic, and not the classical logic of normal mathematical practice.

This view, more or less, appears to have first been put forward by Richman ([40], [41]). It does not, of course, reflect the way in which Brouwer, Heyting, Markov, Bishop, and other pioneers of constructive mathematics regarded their activities. Indeed, it is ironic that, having first become interested in constructivism through the persuasive writings of Bishop, in which, as with Brouwer, the use of what became identified as intuitionistic logic was derived from an analysis of his perception of meaningful mathematical practice, we have been led, through our practice of Bishop-style mathematics, to a view that perhaps it is the logic that determines the kind of mathematics that we are doing.

Note that this is a view of the *practice* of constructive mathematics, and is certainly compatible with a more radical constructive philosophy of mathematics, such as Brouwer's intuitionism, in which the *objects* of mathematics are mental constructs. Thus, in saying that constructive mathematics deals with "normal mathematical objects", we have not precluded the possibility that the radical constructivist view of the nature of those objects may hold; the viewpoint we have adopted is an epistemological, rather than ontological, one

From now on, when we speak of "normal mathematical objects", we have in mind the kind of things that are handled by either **Heyting arithmetic**—the Peano axioms plus intuitionistic logic—or, at a higher level, a formal system such as intuitionistic set theory (IZF), Myhill's constructive set theory (CST), or Martin-Löf's type theory (the last two of which are discussed later in this paper). When working in any axiomatic system, we must take care to use only intuitionistic logic, and therefore to ensure that we do not adopt a classical axiom that implies LEM or some other nonconstructive principle. For example, in IZF we cannot adopt the common classical form of the axiom of foundation,

$$\forall x \exists y \, (y \in x \wedge y \cap x = \emptyset) \, ,$$

since it entails LEM ([35], [14]).

A rather different approach to a constructive theory of sets (based on A.P. Morse's beautiful classical development [34]), in which each statement can be read either as one in intuitionistic predicate calculus or as one about sets, was

developed in [13]. In this approach there is a universal class U, and the members of U correspond to those objects whose existence has been established constructively. An outline of this theory can be found in [14].

We now look a little more closely at intuitionistic logic. To illustrate how Heyting arrived at his axioms, note that in order to prove that either the equation $f(n) = 0$ or the equation $g(n) = 0$ has a solution, where f, g are functions on the natural numbers, it is not enough for the intuitionist to prove the impossibility of neither having a solution: such a proof would not enable him to find a solution of either equation. Thus we are led to the constructive interpretation of disjunction: (P or Q) holds if and only if either we have a proof of P or we have a proof of Q.

Similar consideration of all the logical connectives

$$\vee \text{(or)}, \wedge \text{(and)}, \Rightarrow \text{(implies)}, \neg \text{ (not)}$$

in the light of constructive mathematical practice leads to the following axioms for the **intuitionistic propositional calculus:**

1. $P \Rightarrow (P \wedge P)$
2. $(P \wedge Q) \Rightarrow (Q \wedge P)$
3. $(P \Rightarrow Q) \Rightarrow (P \wedge R \Rightarrow Q \wedge R)$
4. $(P \Rightarrow Q) \Rightarrow ((Q \Rightarrow R) \Rightarrow (P \Rightarrow R))$
5. $Q \Rightarrow (P \Rightarrow Q)$
6. $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$
7. $P \Rightarrow (P \vee Q)$
8. $(P \vee Q) \Rightarrow (Q \vee P)$
9. $((P \Rightarrow R) \wedge (Q \Rightarrow R)) \Rightarrow ((P \vee Q) \Rightarrow R)$
10. $\neg P \Rightarrow (P \Rightarrow Q)$
11. $((P \Rightarrow Q) \wedge (P \Rightarrow \neg Q)) \Rightarrow \neg P$

To use these axioms we also need one rule of inference, **modus ponens:** from P and $(P \Rightarrow Q)$ we infer Q. To obtain axioms for the classical propositional calculus, we need only add LEM to the foregoing intuitionistic ones.

A **first-order language** consists of the connectives used above, together with the quantifiers $\exists$ (there exists) and $\forall$ (for each), a list of variables and constants, and a list of predicate symbols. Each predicate symbol has an associated positive integer, giving the number of places it has. We need the notion of a **well-formed formula**, introduced recursively as follows.

> If P is an n-place predicate, and $a_1, \ldots, a_n$ are variables or constants, then $P(a_1, \ldots, a_n)$ is a well-formed formula.
>
> If A and B are well-formed formulae, then so are $A \vee B$, $A \wedge B$, $A \Rightarrow B$, and $\neg A$.
>
> If A is a well-formed formula, and x is a variable, then $\exists x\, A$ and $\forall x\, A$ are well-formed formulae.

We denote by $A(x/t)$ the result of replacing every occurrence of the variable x in A by t; here, t can be either a variable or a constant. An occurrence of the variable x in A is **bound** if it appears in a subformula of the form $\forall xB$ or $\exists xB$; otherwise, the occurrence of x in A is **free.** Let x be a variable, t a variable or constant, and A a formula; we say that t is **free for** x **in** A if no free occurrence of x in A is in a subformula of A of the form $\forall tB$.

We obtain the **intuitionistic predicate calculus** by adding to the axioms of the intuitionistic propositional calculus those in the following list, together with the rule of inference known as **generalisation:** from A infer $\forall x\, A$.

1. $\forall x\, (A \Rightarrow B) \Rightarrow (A \Rightarrow \forall xB)$ if x is not free in A
2. $\forall x\, (A \Rightarrow B) \Rightarrow (\exists xA \Rightarrow B)$ if x is not free in B
3. $\forall x\, A \Rightarrow A(x/t)$ if t is free for x in A
4. $A(x/t) \Rightarrow \exists xA$ if t is free for x in A.

There is are model theories for this logic—Kripke models and Beth models. These models are often useful for showing that classical results, such as LPO, cannot be derived within Heyting arithmetic; see [19] and Chapter 7 of [17].

To carry out the development of mathematics, as distinct from logic, constructively, Bishop also requires the notions of **set** and **function.**

> A set is not an entity which has an ideal existence: a set exists only when it has been defined. To define a set we prescribe, at least implicitly, what we (the constructing intelligence) must do in order to construct an element of the set, and what we must do to show that two elements of the set are equal. ([3], p. 2)

There are two points to emphasise in this quotation. First, Bishop does not require that the property characterising a set be decidable. (Under the recursive interpretation, to do so would be to restrict oneself to recursive subsets of the natural numbers, which would patently destroy the viability of the theory.) Secondly, Bishop requires the equality relation between elements of a set to be a part of the definition of the set, provided that it satisfies the usual rules for an equivalence relation:

- $x = x$,

- $x = y \Rightarrow y = x$,
- $((x = y) \wedge (y = z)) \Rightarrow x = z$.

In particular, this means that we cannot form such objects as the union of two sets unless the sets come with equality relations that are compatible in the obvious sense; normally, this means that the two sets will themselves be given as subsets of a third set from which their equality relations are induced.

In general, Bishop is not interested in intensional equality (identity) of objects. For example, he defines a **real number** as a sequence (x_n) of rational numbers that is **regular**, in the sense that

$$|x_m - x_n| \leq \frac{1}{m} + \frac{1}{n}$$

for all $m, n > 1$; he then defines two real numbers $(x_n), (y_n)$ to be **equal** if

$$|x_n - y_n| \leq \frac{2}{n}$$

for all $n > 1$. So he works directly with Cauchy sequences, rather than, as would the classical mathematician, with equivalence classes of Cauchy sequences. This is akin to the standard practice of calling the fractions $\frac{1}{2}$ and $\frac{17}{34}$ "equal", rather than "equivalent".

Having dealt with sets, Bishop turns to functions:

> in order to define a function from a set A to a set B, we prescribe a finite routine which leads from an element of A to an element of B, and show that equal elements of A give rise to equal elements of B. ([3], p. 2)

The notion defined by dropping from this definition the last clause, about preservation of equality, is called an **operation**. In the first part of this paper we shall have little to say about operations, but they will have more significance in the second part, when we discuss Martin-Löf's theory of types.

The notions of positive integer, set, and function are the foundation stones of BISH:

> Building on the positive integers, weaving a web of ever more sets and more functions, we get the basic structures of mathematics: the rational number system, the real number system, the euclidean spaces, the complex number system, the algebraic number fields, Hilbert space, the classical groups, and so forth. Within the framework of these structures most mathematics is done. Everything attaches itself to number, and every mathematical statement ultimately expresses the fact that if we perform certain computations within the set of positive integers, we shall get certain results. ([3], pp. 2-3)

The constructivists' rejection[4] of LPO has some significant consequences even at the level of the real number line $\mathbf{R}$. For example, we cannot expect to prove constructively that

$$\forall x \in \mathbf{R}\ (x = 0 \vee x \neq 0)\,,$$

where $x \neq 0$ means $|x| > 0$. (Here we are anticipating some elementary constructive properties of $\mathbf{R}$.) For if we could prove this statement, then, given any binary sequence $\mathbf{a}$ and applying it to the real number whose binary expansion is $0 \cdot a_1 a_2 a_3 \cdots$, we could prove LPO.

Among other classical propositions that imply LPO are

- The **law of trichotomy:** $\forall x \in \mathbf{R}\ (x < 0 \vee x = 0 \vee x > 0)\,.$
- The **least-upper-bound principle:** each nonempty subset of $\mathbf{R}$ that is bounded above has a least upper bound.
- Every real number is either rational or irrational. (To see this, consider a decreasing binary sequence (a_n) and the real number $\sum_{n=1}^{\infty} a_n/n!$.)

Another classically trivial principle that is rejected in BISH is the **Lesser Limited Principle of Omniscience (LLPO)**

$$\begin{aligned}\forall \mathbf{a} \ \in\ \{0,1\}^{\mathbf{N}}\,(\forall m \forall n\,(a_m = a_n = 1 \Rightarrow m = n) \Rightarrow \\ \forall n\,(a_{2n} = 0) \vee \forall n\,(a_{2n+1} = 0))\end{aligned}$$

—in other words, if (a_n) is a binary sequence with at most one term equal to 1, then either $a_{2n} = 0$ for all n or else $a_{2n+1} = 0$ for all n.
Among the classical propositions that entail LLPO and are therefore regarded as essentially nonconstructive are

- $\forall x \in \mathbf{R}\ (x > 0 \vee x \not> 0)\,.$
- If $x, y \in \mathbf{R}$ and $xy = 0$, then $x = 0$ or $y = 0$.
- The **Intermediate Value Theorem:** If $f : [0,1] \to \mathbf{R}$ is a continuous function with $f(0) < 0 < f(1)$, then there exists $x \in (0,1)$ such that $f(x) = 0$.

For more on LPO, LLPO, and related matters, we refer the reader to Chapter 1 of [17].

[4]There is another reason for rejecting LPO in the constructive setting: its recursive interpretation is provably false within recursive function theory, even with classical logic (see [17], Chapter 3). So if we want BISH to remain consistent with a recursive interpretation, we must not allow LPO to be used therein.

It would be wrong to get the impression that constructive mathematics only deals with negative results. For example, there are several constructive substitutes for the Intermediate Value Theorem, each of which can be successfully applied to most of the functions that arise in practice in analysis; see [6] (pages 40-41 and 63), and [17] (pages 54-58). Indeed, the major effort of Bishop and his followers has been directed at obtaining positive constructive substitutes for classical results and theories.

2 Myhill's Constructive Set Theory

In this section we outline Myhill's constructive set theory (CST—see [36]), providing a formal foundation for BISH. Although this is one of several formal systems intended to capture the spirit and method of BISH ([20], [21]), it is one which we understand that Bishop himself held in some regard.

CST is based on intuitionistic predicate logic with identity. The variables are of three basic kinds: **numbers**, **sets**, and **functions**. The seven primitive notions are

- three constants
  - 0 (**zero**)
  - s (**successor**)
  - **N** (**the set of natural numbers**);
- two one-place predicates
  - $\mathcal{M}(a)$ (a is a set)
  - $\mathcal{F}(a)$ (a is a function);
- a two-place predicate
  - $a \in b$ (a is an element of the set b);
- a three-place predicate
  - $V(a, b, c)$ (the function a is defined for the argument b and has the corresponding value c)

The last of these predicates enables us to handle partial functions whose domains are not necessarily decidable. In practice, we would normally write $a(b) = c$ rather than $V(a, b, c)$.

The axioms of CST fall into several groups, the first of which clarifies the nature of the basic objects.

A1 Everything is a number, a function, or a set: $a \in \mathbf{N} \vee \mathcal{F}(a) \vee \mathcal{M}(a)$

A2 Numbers are not functions: $a \in \mathbf{N} \Rightarrow \neg\mathcal{F}(a)$

A3 Functions are not sets: $\mathcal{F}(a) \Rightarrow \neg\mathcal{M}(a)$

A4 Sets are not numbers: $\mathcal{M}(a) \Rightarrow \neg\,(a \in \mathbf{N})$

A5 Only numbers have successors: $V(s,a,b) \Rightarrow a \in \mathbf{N}$

A6 Only functions have values: $V(a,b,c) \Rightarrow \mathcal{F}(a)$

A7 Only sets have members: $a \in b \Rightarrow \mathcal{M}(b)$

A8 A function has at most one value for a given argument: $V(a,b,c) \wedge V(a,b,d) \Rightarrow c = d$

The second group of axioms is **Peano's axioms** for the natural numbers.

B1 $0 \in \mathbf{N}$

B2 $a \in \mathbf{N} \Rightarrow \exists y\,(V(s,a,y) \wedge y \in \mathbf{N})$

B3 $\neg V(s,a,0)$

B4 $V(s,a,c) \wedge V(s,b,c) \Rightarrow a = b$

B5 $(P(0) \wedge \forall x \forall y\,((P(x) \wedge V(s,x,y)) \Rightarrow P(y))) \Rightarrow \forall x\,(x \in \mathbf{N} \Rightarrow P(x))$, where $P(x)$ is a one-place predicate.

The next axiom embodies the principle that if for each element x of a set A there exists a unique element y of a set B such that $P(x,y)$, then y is obtained from x by a function from A to B. Before stating this axiom we introduce a convenient shorthand:

$$\mathrm{dom}(z) = a \text{ stands for } \forall x\,(x \in a \Leftrightarrow \exists y\, V(z,x,y))$$

Now we have what Myhill calls an **axiom of nonchoice:**

C1 $(\mathcal{M}(a) \wedge \forall x \in a\, \exists! y \in b\, P(x,y)) \Rightarrow$
$\exists f\,(\mathcal{F}(f) \wedge \mathrm{dom}(f) = a \wedge \forall x \in a\, \exists y \in b\,(V(f,x,y) \wedge P(x,y)))$

In addition, we have the **axiom of dependent choice**:

C2 $(t \in a \wedge \forall x\ \in a\, \exists y\, P(x,y)) \Rightarrow$
$\exists f\ (\mathcal{F}(f) \wedge \mathrm{dom}(f) = \mathbf{N} \wedge V(f,0,t) \wedge$
$\forall x \in \mathbf{N}\, \exists y \in a\, \exists z \in a\ (V(f,x,y) \wedge V(f,s(x),z) \wedge P(y,z)))\,,$

where P is a two-place predicate. It is not hard to derive from this last axiom the **principle of countable choice**:

$$(\forall x \in \mathbf{N}\, \exists y \in a\, P(x,y)) \Rightarrow \exists f\ (\mathcal{F}(f) \wedge \mathrm{dom}(f) = \mathbf{N} \wedge \forall x \in \mathbf{N}\, \exists y \in A\, V(f,x,y))\,.$$

These three choice principles appear to be sufficient[5] for the development of analysis in [3] and [6]. The full axiom of choice, on the other hand, cannot be allowed in constructive mathematics, since, as Goodman and Myhill have shown [22], it entails the law of excluded middle.

There appears to be a conflict here with Bishop's remark ([3], p. 9) that

> the axiom of choice ... is not a real source of nonconstructivity in classical mathematics. A choice function exists in constructive mathematics, because a choice *is implied by the very meaning of existence.*

Indeed, it is true that if to each element x of a set A there corresponds an element y of set B such that the property $P(x,y)$ holds, then it is implied by the meaning of existence in constructive mathematics that there is a finite routine for computing an appropriate $y \in B$ from a given $x \in A$; but this computation may depend not only on the value a *but also on the information that shows that* a *belongs to the set* A. The computation of the value at a of a function f from A to B would depend only on a, and not on the proof that a belongs to A; in other words, a function is **extensional.** So Bishop's remark is correct if he admits functions whose value depends on both a and a proof that $a \in A$, but is not correct if, as Myhill does, one only admits extensional functions.

Of course, the axiom of choice will hold for us if the set A is one for which no computation is necessary to demonstrate that an element belongs to it; Bishop calls such sets **basic sets.** For Myhill and Bishop, $\mathbf{N}$ is a basic set, a belief reflected in their acceptance of the principle of countable choice.

Returning to Myhill's axioms, we now have a group that reflects the usual types of axiom found in classical set theories. The first two of these show that the domain and range of a function are sets.

D1 $\mathcal{F}(f) \Rightarrow \exists X\, \forall x\ (x \in X \Leftrightarrow \exists y\, V(f,x,y))$

D2 $\mathcal{F}(f) \Rightarrow \exists X\, \forall x\ (x \in X \Leftrightarrow \exists y\, V(f,y,x))$

Axiom D2 acts like the standard axiom of replacement in classical set theory, since it implies that

$$\mathcal{F}(f) \Rightarrow \exists X\, \forall y\ (y \in X \Leftrightarrow \exists x \in A\, V(f,x,y))$$

[5] It appears, however, that there may be many places in the development of BISH where substantial results are provable without the principles of countable choice or dependent choice; see for example, [39].

—in other words, that the set $\{f(x) : x \in A \cap \mathrm{dom}(f)\}$ exists.

Next we have the **mapping set axiom:**

D3 $\exists X \, \forall f \; (f \in X \Leftrightarrow \mathcal{F}(f) \wedge \mathrm{dom}(f) = A \wedge \mathrm{ran}(f) \subset B)$,

where

$$\forall x \; (x \in \mathrm{ran}(f) \Leftrightarrow \exists y \, V(f, y, x))$$

and

$$S \subset B \Leftrightarrow \forall x \; (x \in S \Rightarrow x \in B).$$

The mapping set axiom if a weak substitute for the standard **power set axiom,**

$$\exists Y \; \forall s \; (s \in Y \Leftrightarrow s \subset X),$$

to which Myhill and others have raised serious constructive objections; see pages 351–352 and 364-365 of [36]. The power set axiom is used implicitly in the chapter on measure theory in [6], but, as Myhill points out on pages 354-355 of his paper [36], the power set axiom can easily be avoided in constructive measure theory.

Myhill's next axiom, asserting the existence of the **pair set** $\{a, b\}$ formed from two objects a and b, can actually be deduced from the one following it, C1, and D2, but we shall not do this:

D4 $\exists X \, \forall x \; (x \in X \Leftrightarrow x = a \vee x = b)$.

The existence of the **ordered pair** (a, b), defined as the function f with domain $\{0, 1\}$ such that $f(0) = a$ and $f(1) = b$, can also be deduced from the axioms.

For the next axiom we define the notion of a **restricted formula** as follows. Atomic formulae are restricted; propositional combinations of restricted formulae are restricted; if P is restricted and τ is a parameter or $\mathbf{N}$, then $\forall x \in \tau \, P(x)$ and $\exists x \in \tau \, P(x)$ are restricted. We now have the **axiom of predicative separation:**

D5 $\exists X \, \forall x \; (x \in X \Leftrightarrow x \in A \wedge P(x))$, where every bound variable of P is restricted to a set.

The purpose of the restriction condition is to ensure that the condition defining a set only refers to sets that have already been defined—in other words, to avoid circularity in the definition of sets.

The last axiom of this group is that of **union:**

D6 $(\forall x \in A \, \mathcal{M}(x)) \Rightarrow \exists X \, \forall x \; (x \in X \Leftrightarrow \exists Y \; (x \in Y \wedge Y \in A))$

Finally, we have two **axioms of extensionality** for functions and sets:

E1 $\mathcal{F}(a) \wedge \mathcal{F}(b) \Rightarrow (a = b \Leftrightarrow (\mathrm{dom}(a) = \mathrm{dom}(b)) \wedge$
$\forall x \in \mathrm{dom}(A) \, \forall y \, (V(a, x, y) \Leftrightarrow V(b, x, y)))$

E2 $A = B \Leftrightarrow \forall x\ (x \in A \Leftrightarrow x \in B)$

We believe that Myhill's axiomatic system captures well the spirit of Bishop's approach to constructive mathematics, based, as it is, on the notions of natural number, set, and function. However, we shall not attempt in this paper to use the axioms to formalise any parts of BISH.

3 The Constructive Real Line

Although the derivation of the algebraic and order properties of the real line $\mathbf{R}$ using Bishop's definitions of real number, equality of real numbers, positive, and nonnegative is reasonably smooth, it is instructive (and perhaps paedogogically advantageous) to produce a constructive axiomatic development of $\mathbf{R}$. These axioms are intended to capture the idea that a real number, whatever it may be, is something that can be arbitrarily closely approximated by rational numbers. (In Bishop's formal construction, referred to above, that approximation is done by means of regular Cauchy sequences of rational numbers.)

Our starting point is to assume the existence of a set $\mathbf{R}$ with

- a binary relation $>$ (**greater than**)
- a corresponding **inequality relation** $\neq$ defined by

$$x \neq y \text{ if and only if } (x > y \text{ or } y > x)$$

- binary operations $(x, y) \mapsto x + y$ (**addition**) and $(x, y) \mapsto xy$ (**multiplication**)
- distinguished elements 0 (**zero**) and 1 (**one**) with $0 \neq 1$
- a unary operation $x \mapsto -x$
- a unary operation $x \mapsto x^{-1}$ on the set of elements $x \neq 0$.

The elements of $\mathbf{R}$ are called **real numbers.** We identify the sets $\mathbf{N}$ of natural numbers, $\mathbf{N}^+$ of positive integers, $\mathbf{Z}$ of integers, and $\mathbf{Q}$ of rational numbers with the usual subsets of $\mathbf{R}$: for example, we identify $\mathbf{N}^+$ with $\{n1 : n \in \mathbf{N}^+\}$.

We say that a real number x is **positive** if $x > 0$, and **negative** if $-x > 0$. We define the relation $\geq$ (**greater than or equal to**) by

$$x \geq y \text{ if and only if } \forall z\ (y > z \Rightarrow x > z),$$

and we define the relations $<$ and $\leq$ in the usual way, calling x **nonnegative** if $x \geq 0$. Two real numbers x, y are **equal** if $x \geq y$ and $y \geq x$, in which case we write $x = y$. Note that this notion of equality satisfies the usual properties of an equivalence relation.

We assume that all the foregoing relations and operations are **extensional**; for example, to say that the relation $>$ is extensional means that if $x > y$, $x = x'$, and $y = y'$, then $x' > y'$. We also assume that they satisfy a number of axioms, falling into three groups, the first of which deals with the basic algebraic properties of $\mathbf{R}$.

R1. $\mathbf{R}$ is a **Heyting field**: For all $x, y, z \in \mathbf{R}$,

$$\begin{aligned} x + y &= y + x, \\ (x + y) + z &= x + (y + z), \\ 0 + x &= x, \\ x + (-x) &= 0, \\ xy &= yx, \\ (xy)\, z &= x\,(yz), \\ 1x &= x, \\ xx^{-1} &= 1 \text{ if } x \neq 0, \text{ and} \\ x(y + z) &= xy + xz. \end{aligned}$$

Of course, we also denote x^{-1} by $\frac{1}{x}$ or $1/x$.

It is natural to ask whether, for the existence of x, it suffices to have $\neg(x = 0)$. The answer is provided by a well known example which shows that the statement

$$\forall x \in \mathbf{R}\,(\neg(x = 0) \Rightarrow \exists y \in \mathbf{R}\ (xy = 1))$$

is equivalent to **Markov's Principle (MP):**

$$\forall \mathbf{a} \in \{0,1\}^{\mathbf{N}}\,(\neg\,(\mathbf{a} = \mathbf{0}) \Rightarrow \mathbf{a} \neq \mathbf{0})$$

—that is, if (a_n) is a binary sequence such that $\neg\forall n\ (a_n = 0)$, then there exists n such that $a_n = 1$. (See [17], Ch. 1, Problem 8). Since Markov's Principle is a form of unbounded search, it is not accepted by the majority of constructive mathematicians (although it is clearly true in classical mathematics).

We now have the second group of axioms.

R2. Properties of $>$.

1. $\neg(x > y \text{ and } y > x)$
2. $(x > y) \Rightarrow \forall z\ (x > z \vee z > y)$
3. $\neg(x \neq y) \Rightarrow x = y.$
4. $(x > y) \Rightarrow \forall z\,(\,x + z > y + z)$

5. $(x > 0 \wedge y > 0) \Rightarrow xy > 0$

The second of these axioms is a substitute for the law of trichotomy, and can be justified heuristically as follows. Given that $x > y$, and given any real number z, approximate $\frac{1}{2}(x+y)$ and z to within $\frac{1}{8}(x-y)$ by rational numbers p and q respectively. Using rational arithmetic, we can decide whether $q \leq p$ or $q > p$. In the first case we have

$$\begin{aligned} z &< q + \tfrac{1}{8}(x-y) \\ &\leq p + \tfrac{1}{8}(x-y) \\ &< \tfrac{1}{2}(x+y) + \tfrac{1}{8}(x-y) + \tfrac{1}{8}(x-y) \\ &= x. \end{aligned}$$

In the second case a similar argument shows that $z > y$.

In connection with axiom **R2**(3), note that the statement

$$\forall x, y \in \mathbf{R}\ (\neg\,(x = y) \Rightarrow x \neq y)$$

is equivalent to Markov's Principle ([17], Ch. 1, Problem 8).

Our last two axioms describe special properties of $>$ and $\gg$. For the second of these we need to know that the notions **bounded above, bounded below,** and **bounded** are defined as in classical mathematics; and that, for example, if S is a nonempty subset of $\mathbf{R}$ that is bounded above, then its **least upper bound,** if it exists, is the unique real number b such that

- b is an upper bound of S, and
- for each $b' < b$ there exists $s \in S$ such that $s > b'$.

(Note that **nonempty** means **inhabited**—that is, we can construct an element of the set in question.)

R3 Special properties of $>$.

1. **Axiom of Archimedes:** For each $x \in \mathbf{R}$ there exists $n \in \mathbf{Z}$ such that $x < n$.
2. **The Least-upper-bound Principle:** Let S be a nonempty subset of $\mathbf{R}$ that is bounded above relative to the relation $\gg$, such that for all real numbers α, β with $\alpha < \beta$, either β is an upper bound of S or else there exists $s \in S$ with $s > \alpha$; then S has a least upper bound.

The first of these two axioms would seem to require no justification; but the second is a little harder to motivate. To do so, consider the following attempt to construct the least upper bound of a set S that is bounded above. Let $s_0 \in S$ and let b_0 be an upper bound for S. Having constructed $s_n \in S$ and an upper

bound b_n for S, consider $t \equiv \frac{1}{2}(s_n + b_n)$: if t is an upper bound for S, set $s_{n+1} = s_n$ and $b_{n+1} = t$; if t is not an upper bound for S, then choose $s_{n+1} \in S$ such that $s_{n+1} > t$, and set $b_{n+1} = b_n$. This gives an inductive construction of a sequence (s_n) in S and a sequence (b_n) of upper bounds for S, such that for each $n > 1$,

$$[s_n, b_n] \subset [s_{n-1}, b_{n-1}]$$

and

$$0 < b_n - s_n < 2^{-n}(b_0 - s_0).$$

Our intuition of the real number system now suggests that the sequence (s_n) and (b_n) converge to a common limit that is the required least upper bound.

Viewed constructively, this argument breaks down because we cannot decide whether or not t is a least upper bound for S. However, if S has the additional property in the hypothesis of axiom **R3**(2), then we can modify the unsuccessful classical attempt as follows. Having found s_n and b_n, consider the two numbers

$$\begin{aligned} t_1 &\equiv \tfrac{2}{3}s_n + \tfrac{1}{3}b_n, \\ t_2 &\equiv \tfrac{1}{3}s_n + \tfrac{2}{3}b_n. \end{aligned}$$

Since $t_1 < t_2$, either t_2 is an upper bound for S, in which case we set $s_{n+1} = s_n$ and $b_{n+1} = t_2$; or else there exists $s_{n+1} \in S$ such that $s_{n+1} > t_1$, in which case we set $b_{n+1} = b_n$. This gives an inductive construction of a sequence (s_n) in S and a sequence (b_n) of upper bounds for S, such that for each $n > 1$,

$$[s_n, b_n] \subset [s_{n-1}, b_{n-1}]$$

and

$$0 < b_n - s_n < \left(\tfrac{2}{3}\right)^n (b_0 - s_0).$$

Again, our intuition leads us to expect that the sequences (s_n) and (b_n) will approach a common limit, which will be the least upper bound of S. Thus we have the heuristic motivation for our axiom **R3**(2).

While not exactly routine, it is nevertheless not too hard to derive from these axioms the properties of $\mathbf{R}$ that Bishop establishes directly from his definitions. In particular, we can prove that $\mathbf{R}$ is **complete**, in the usual sense that each Cauchy sequence of real numbers has a limit in $\mathbf{R}$ [15].

4 A Case Study: Approximation Theory

To illustrate Bishop's mathematics in practice, we now consider some constructive aspects of approximation theory. This will require of the reader some familiarity with some basic classical notions of the theory of metric and normed spaces.

A subset V of a metric space (X, ρ) is **located** if

$$\rho(x, V) \equiv \inf\{\rho(x, v) : v \in V\}$$

exists (is computable!) for each $x \in X$. It is relatively straightforward to show that finite-dimensional subspaces of a normed space are located. But we cannot expect to prove that every linear subset of $\mathbf{R}$ is located. To see this, take any real number a and consider the linear subset

$$\mathbf{R}a = \{ax : x \in \mathbf{R}\}$$

of $\mathbf{R}$. If $\mathbf{R}a$ is located, then we can compute $\rho(1, \mathbf{R}a)$. By axiom **R2**(2), either $\rho(1, \mathbf{R}a) > 0$ or $\rho(1, \mathbf{R}a) < 1$. In the first case it is absurd that $a \neq 0$, so $a = 0$, by **R2**(3). In the second, choosing x such that $|1 - ax| < 1$, we see that $|ax| > 0$, so $a \neq 0$. (It is an elementary deduction from our axioms for $\mathbf{R}$ that if $xy \neq 0$, then $x \neq 0$ or $y \neq 0$; see [15].

Let Y be a located subset of the metric space (X, ρ), and a and element of X. We say that $b \in Y$ is a **best approximation** to a in Y if $\rho(a, b) = \rho(a, X)$; and that Y is **proximinal** in X if each $x \in X$ has a best approximation in Y. The fundamental theorem of classical approximation theory says that

Each finite-dimensional subspace of a real normed space is proximinal.

The classical proofs of this theorem depend on the theorem that a continuous, real-valued function on a compact space attains its infimum, a result that implies LLPO. In fact, as is shown in [11], it is not just the proofs, but the theorem itself, that is nonconstructive. So it is a serious problem to find a good constructive substitute for that theorem.

To this end, we say that an element a of a metric space X has **at most one best approximation** in the subset Y of X if

$$\max\{\rho(a, y),\ \rho(a, y')\} > \rho(a, Y)$$

whenever y, y' are distinct points of Y; and that Y is **quasiproximinal** if each $x \in X$ with at most one best approximation in Y has a (unique) best approximation in Y. Clearly, a proximinal subspace is quasiproximinal. Classically, it can be shown that proximinal and quasiproximinal are equivalent concepts: for if a given $x \in X$ has no best approximation in a quasiproximinal subspace Y, then it has at most one, and therefore exactly one, best approximation in Y, which is absurd.

The following constructive version of the fundamental theorem of approximation theory was proved in [9]:

Each finite-dimensional subspace of a real normed space is quasiproximinal.

The tricky part of the proof is a lemma dealing with a strong version of the case where the dimension is 1; the rest is a careful induction over the dimension of the subspace. The result itself is an ideal constructive substitute for the classical fundamental theorem, in that it is classically equivalent to that theorem. It illustrates a common phenomenon: namely, that classical unique existence often translates into constructive existence. It also covers **Chebyshev approximation,** where X is the Banach space of continuous functions on the closed interval $[0, 1]$ and Y is the subspace spanned by the monomials $1, x, x^2, \ldots, x^n$ [8]. However, the existence, continuity, and strong unicity of the best Chebyshev approximation can be proved constructively without using the Fundamental Theorem [11].

Now, there is a famous algorithm for constructing best Chebyshev approxi–mations—the *Remes algorithm.* Does that not provide a constructive existence proof? It does not. Inspection reveals that the classical proof of the convergence of the Remes algorithm is nonconstructive: at one crucial step it shows that a sequence converges by assuming the contrary and deducing a contradiction [27]. It is really quite remarkable that such an important classical algorithm is presented without estimates of its rate of convergence! Fortunately, a more careful description and analysis of the algorithm leads to a constructive proof of its convergence [10].

We should be realistic about what such a proof has achieved. In order to handle the convergence of the Remes algorithm in even the most pathological cases, the estimates produced by the constructive proof are, of necessity, extremely rough. There remains, however, the possibility that a deeper constructive analysis will produce convergence estimates that can be used in practical applications of the algorithm.

5 Intuitionism and Computer Science

The first explicit, direct use of intuitionistic logic in connection with computer science was the paper *Constructive Mathematics and Computer Programming* (later reprinted as [32]), which was read by Per Martin-Löf at the 6th International Congress for Logic, Methodology and Philosophy of Science in Hannover in August 1979. This paper followed the first expositions of Martin-Löf's ideas in [29] and in some lecture notes, made by Sambin during a course in 1980, published as [31]. (It is interesting to note that Bishop foresaw the possibility of using constructive mathematics as a basis for programming; he suggested in [4] using Gödel's theory of computable functionals of finite type.)

In his series of papers Martin-Löf first develops the philosophical and formal basis for his constructive set theory, or **constructive type theory,** and then points out and exploits the identity between mathematics and programming. In this very clear sense Martin-Löf's work shows the truth of the statement made in an earlier section, namely that *algorithmic mathematics*—that is, computer

science—*appears to be equivalent to mathematics that uses only intuitionistic logic*. We now expand on this point and make clear that the apparent equivalence is real.

Martin-Löf explains the equivalence in a table in [30], some of which runs:

Programming	**Mathematics**
program, procedure, algorithm	function
input	argument
output, result	value
$\vdots$	$\vdots$
$a : A$	$a \in A$
$\vdots$	$\vdots$
record $s1 : T1;\ s2 : T2$ *end*	$T1 \times T2$
$\vdots$	$\vdots$

and he says (in the same paper):

> the whole conceptual apparatus of programming mirrors that of modern mathematics (set theory, that is, not geometry) and yet is supposed to be different from it. How come? The reason for this curious situation is, I think, that the mathematical notions have gradually received an interpretation, the interpretation which we refer to as classical, which makes them unusable for programming. Fortunately, I do not need to enter the philosophical debate as to whether the classical interpretation of the primitive logical and mathematical notions ... is sufficiently clear, because this much at least is clear, that if a function is defined as a binary relation satisfying the usual existence and unicity conditions, whereby classical reasoning is allowed in the existence proof ... then a function cannot be the same thing as a computer program ... Now it is the contention of the intuitionists...that the basic mathematical notions, above all the notion of function, ought to be interpreted in such a way that the cleavage between mathematics, classical mathematics, that is, and programming that we are witnessing at present disappears. In the case of the mathematical notions of function and set, it is not so much a question of providing them with new meanings as of restoring old ones ...

6 A computational view of proof

In this section we expand on some of the ideas mentioned in the above quote, and, making comments as appropriate, give the complete version of the foregoing

table. In this way we hope to give a good, fairly non-technical view of the effects of constructive mathematics on modern computer science thinking.

One large difference between mathematics and computer science that will quickly become clear is that computer scientists, while "all" that they are doing is algorithmic mathematics, have to spend most of their time dealing with a very formalised world. This is simply because, in the end, they have to produce programs, which are of course nothing more than rather large and very complicated formal objects. Whereas a mathematician, when communicating with other mathematicians, can rely on knowledge, intuition, insight and all those human processes that make up our ability to reason intelligently, the computer scientist has to produce an object that instructs a machine. Every last detail must be explicit; machines, after all, have no intelligence and so cannot be relied on to fill in the gaps in the programs that instruct them. So, since computer scientists spend much of their time producing formal objects, it should not be surprising that they create formal systems within which to work and within which their programs can be built.

Bearing this in mind, we might adapt the characterisation of computer science given above to: *computer science is equivalent to completely formalised mathematics that uses only intuitionistic logic.*

All we have said is by way of preparation for the reader, who must be in the right frame of mind for accepting the need for formalisation and for being patient when we appear to spend inordinate amounts of time and space getting the details of a formalisation correct. We do this not out of any narrowness of view or inability to think; rather we do it because we know that we are forced to do by the nature of the end product.

Now we start building the formal system, based on Martin-Löf's work, within which, later on, we create our programs. Our plan is to begin with a standard logical system (which can be seen as merely a different presentation of Heyting arithmetic) and gradually build on this, all the while mirroring to some extent the underlying logic in Section 1, until we arrive at a system that is expressive enough for our task of constructing programs.

The main difference between the formal parts of Section 1 and what we are about to do is that we use a natural–deduction presentation of the system. In doing this we are not only presenting the system just as Martin-Löf did, but we are following what has (thanks precisely to Martin-Löf's work as taken up by theoretical computer scientists) become a standard way of elegantly presenting a language and its associated logic.

First, we need to introduce some technical terms. Since we will have to distinguish carefully between a proof (in the sense of a witness to the fact that some proposition has been proved) and the record of the construction of that proof we introduce two terms: a **proof object**—that is, a witness to the fact that some proposition has been proved; and a **derivation**—the record of the construction of a proof object. We will see many examples of this use of language later.

A **judgement** comes in two basic forms: either it is a relation between proof objects and propositions, or else it states a property of some propositions. In the first basic form there are two cases, the first of which records that the mentioned proof object is a witness to the mentioned proposition. We write this as

$$a : A$$

which we read as a **is in** A, or a **proves** A, or a **witnesses** A. (These are all somewhat imprecise statements, but they are all commonly used, convenient ways of stating a common situation.) The second case records that two proofs objects are equal and that they witness that a proposition has been proved. We write this as

$$a = b : A$$

The second basic form of a judgement also has two cases, the first of which records that a certain proposition is well-formed. For reasons which we address later, this is written as

$$A\ prop$$

The second case records that two propositions are equal, and is written

$$A = B$$

Finally, these basic forms of judgement are generalised to make them hypothetical judgements by allowing finite lists of hypotheses to appear; so the general judgement has the form

$$a : A[x_1 : A_1, x_2 : A_2, ..., x_n : A_n]$$

where

- the x_i are distinct variables,
- the A_i are propositions such that if x_j is in A_i then $j < i$, and
- $a : A$ is any of the three other possible forms.

These form contexts which introduce variables over proof objects, the variables being available for use within the body of the judgement $a : A$. Again, we will see examples of this below which should help clarify this rather general definition.

We describe the usual connectives via natural deduction rules for their introduction and elimination. These rules are exactly the ones we would expect for a classical logic except that the rules allowing proofs of $\neg\neg\psi \Rightarrow \psi$ or $\psi \vee \neg\psi$ are not included. Our rules also include mention of proof objects.

We need one non–logical rule:

$$\frac{A\ prop}{x : A[x : A]}\ assumption$$

This says that when A is a proposition, the hypothetical judgement $x : A[x : A]$ can be derived.

6.1 Equality Rules

At the level of judgements we have all the rules governing equality that one would expect. For example:

$$\frac{a : A}{a = a : A}\ refl \qquad \frac{a = b : A}{b = a : A}\ symm \qquad \frac{a : A \quad A = B}{a : B}\ prop - eq$$

$$\frac{C(x)\ prop\ [x : A] \quad a : A}{C(a)\ prop}\ subst - prop$$

$$\frac{c(x) : C(x)[x : A] \quad a : A}{c(a) : C(a)}\ subst - obj$$

6.2 Propositional rules

$$\frac{A\ prop \quad B\ prop}{A \Rightarrow B\ prop} \Rightarrow -form \qquad \frac{b(x) : B(x)[x : A]}{\lambda(b) : A \Rightarrow B} \Rightarrow -intro$$

The b in this rule is an abstraction of the form $(v)e$ where v is some variable which, if it appears free in the expression e, will be bound in $(v)e$. The usual term equality holds here:

$$(v)e(x) = e[x/v]$$

—that is, free occurrences of v in e which are free for x are replaced by x.

In intuitionistic (and so classical) logic we have the valid proposition $A \Rightarrow A$ for any proposition A. We should expect this to have a proof in the system we are describing, and so it does. First, consider using the $\Rightarrow$-*intro* rule without mentioning the proof objects (so that it looks like a conventional natural-deduction rule), and build a derivation which shows this sentence to be valid. We can build

$$\frac{\dfrac{A\ prop}{A[A]}\ assumption}{A \Rightarrow A} \Rightarrow -intro$$

Now we can consider the same derivation, this time with the proof objects added:

$$\frac{\dfrac{A\ prop}{x : A[x : A]}\ assumption}{\lambda((x)x) : A \Rightarrow A} \Rightarrow -intro$$

So something of the form λe is a proof object associated with an implication. This makes concrete the idea, originating with Heyting, that the proof of an implication is an algorithm which, given a proof of the antecedent of the implication, constructs a proof of the consequent. (Readers familiar with the lambda calculus [1] will appreciate why λ was chosen to denote such proof objects in this system.) Note that in this trivial case, given a proof of A, the proof of $A \Rightarrow A$, $\lambda((x)x)$, does indeed return a proof of A: from an algorithmic viewpoint it is just the identity function.

$$\frac{c : A \Rightarrow B \quad a : A}{apply(c, a) : B} \Rightarrow -elim \qquad \frac{a : A \quad b(x) : B[x : A]}{apply(\lambda(b), a) = b(a) : B} \Rightarrow -eq$$

The rule $\Rightarrow$-*elim* is the formal counterpart of *modus ponens*, while $\Rightarrow$-*eq* (as with all the -*eq* rules) tells us how certain expressions simplify (reading the equality left-to-right), and so can be thought of as a computation rule when λ and *apply* are given their obvious algorithmic meanings.

If we now reconsider the rules above, replacing $\Rightarrow$ by $\rightarrow$ and 'prop' by 'type', then we catch a first glimpse of the **propositions–as–types** principle which has been so influential. In particular, if we allow our view to switch between propositions and types, we see that implication (a logical notion) has identical properties to the function–space type–former (a computational notion). This identity extends to all the other standard logical connectives.

$$\frac{A\ prop \quad B\ prop}{A \wedge B\ prop} \wedge - form \qquad \frac{a : A \quad b : B}{(a, b) : A \wedge B} \wedge - intro$$

So, given a proof of a conjunction, we can construct further proofs referring to its two component proofs.

$$\frac{x : A \wedge B \quad d(y, z) : C((y, z))[y : A, z : B]}{split(x, d) : C(x)} \wedge - elim$$

$$\frac{a : A \quad b : B \quad d(x, y) : C((x, y))[x : A, y : B]}{split((a, b), d) = d(a, b) : C((a, b))} \wedge - eq$$

This shows that given a pair of proofs we can project out the components. Thus we see that the logical notion of conjunction is associated with the computational notion of forming and manipulating a Cartesian product. Once again, the point about propositions and types being two views of the same idea comes through.

To illustrate this, consider the valid proposition $(A \wedge B) \Rightarrow A$. We can build a proof object for this as in the following derivation:

$$\dfrac{\dfrac{\dfrac{A \wedge B \;\; prop}{x : A \wedge B[x : A \wedge B]}\; assumption \quad \dfrac{A \; prop}{y : A[y : A]}\; assumption}{split(x,(y,z)y) : A \; [x : A \wedge B]}\; \wedge - elim}{\lambda((x)split(x,(y,z)y)) : (A \wedge B) \Rightarrow A}\; \Rightarrow - intro$$

We can see how this object is used computationally by applying it to a proof of $A \wedge B$, which will have the form (a, b) where a is a proof of A and b is a proof of B. Instead of giving the fully formal derivation, we paraphrase it by the following sequence:

$$apply(\lambda((x)split(x,(y,z)y)),(a,b)) = split((a,b),(y,z)y) = a$$

So the proof object that witnesses $(A \wedge B) \Rightarrow A$ again has a computational interpretation: given a proof of $A \wedge B$ it returns a proof of A.

$$\frac{A \; prop \quad B \; prop}{A \vee B \; prop}\; \vee - form$$

$$\frac{a : A}{i(a) : A \vee B}\; \vee - intro \qquad\qquad \frac{b : B}{j(b) : A \vee B}\; \vee - intro$$

The interpretation of $\vee$ is where the distinction between our logic and a classical one becomes clear: in order to prove a proposition of the form $A \vee B$, we have to provide either a proof of A or a proof of B, and record, for later use, which of these we have provided. This means that the proposition $A \vee \neg A$ is not true—that is, not provable—since we cannot, for arbitrary A, exhibit either a proof of A or one of $\neg A$.

This point is important since, as we shall see, from a propositional point of view, $\vee$ represents a disjoint union $+$, and $\Rightarrow$ represents $\rightarrow$, the function–space constructor. If we consider the definition, perhaps in some notional programming language,

$$Number =_{df} Float + Int$$

and the existence of a function

$$add : Number \rightarrow Number$$

we can see that in computing addition, *add* needs to be able to tell, for some argument $n : Number$, from which summand n originally came, since the operation of addition which *add* has to carry out depends on this information.

The remaining rules for disjunction are

$$\frac{c : A \vee B \quad d(x) : C(i(x))[x : A] \quad e(y) : C(j(y))[y : B]}{when(c,d,e) : C(c)}\; \vee - elim$$

$$\frac{a : A \quad d(x) : C(i(x))[x : A] \quad e(y) : C(j(y))[y : B]}{when(i(a), d, e) = d(a) : C(i(a))} \vee - eq$$

$$\frac{b : B \quad d(x) : C(i(x))[x : A] \quad e(y) : C(j(y))[y : B]}{when(j(b), d, e) = e(b) : C(j(b))} \vee - eq$$

To give some idea of how these work (since they are somewhat notationally dense) consider the following simple example. We would hope that, given a proof of $(A \vee B) \Rightarrow C$ and a proof of A, we would be able to prove that C holds. Assuming that we have a proof of C, we can derive the judgement

$$\lambda((x)when(x, (y)c, (z)c)) : C \ [c : C]$$

and assuming that A holds—that is, that $a : A$—we have the derivation

$$\frac{a \ : \ A}{i(a) \ : \ A \vee B} \vee - intro$$

Given all this, we can prove C with the following derivation:

$$\frac{\dfrac{a : A}{i(a) : A \vee B} \vee - intro \qquad \lambda((x)when(x, (y)c, (z)c)) : (A \vee B) \Rightarrow C \ [c : C]}{apply(\lambda((x)when(x, (y)c, (z)c)), i(a)) : C \ [c : C]}$$

Then the various equality rules allow us to show that

$$\begin{array}{rcl} apply(\lambda((x)when(x, (y)c, (z)c)), i(a)) & = & when(i(a), (y)c, (z)c) \\ & = & (y)c(a) \\ & = & c \end{array}$$

as required.

6.3 Rules for quantifiers

The rules for the universal quantifier are completely standard:

$$\frac{A \ prop \quad B(x) \ prop}{\forall(A, B) \ prop} \forall - form \qquad \frac{b(x) : B(x)[x : A]}{\lambda(b) : \forall(A, B)} \forall - intro$$

$$\frac{a : A \quad c : \forall(A, B)}{apply(c, a) : B(a)} \forall - elim \qquad \frac{a : A \quad b(x) : B(x)[x : A]}{apply(\lambda(b), a) = b(a) : B(a)} \forall - eq$$

Note that, as for implication, a proof of a universal proposition is viewed as a function: one that, given a proof that some object is in the domain, returns a proof that the object has the property which is stated as being universal. Also note that these rules are closely related to the rules for $\Rightarrow$; indeed the latter

rules can be derived from the former just by observing that the proposition B does not vary in the case of implication.

The rules for the existential quantifier require that, in order to justify a claim that we have proved an existential proposition, we exhibit an object in the required domain and a proof that it has the properties claimed. Hence the natural way of representing the proof object for an existential proposition is as a pair consisting of the object whose existence is claimed and a proof that it has the claimed property.

$$\frac{A\ prop \quad B(x)\ prop}{\exists(A,B)\ prop}\ \exists - form \qquad \frac{a : A \quad b(a) : B(a)}{(a,b) : \exists(A,B)}\ \exists - intro$$

$$\frac{c : \exists(A,B) \quad d(x,y) : C((x,y))\ [x : A, y : B(x)]}{split(c,d) : C(c)}\ \exists - elim$$

$$\frac{a : A \quad b(a) : B(a) \quad d(x,y) : C((x,y))\ [x : A, y : B(x)]}{split((a,b),d) = d(a,b) : C((a,b))}\ \exists - eq$$

6.4 Rules for natural numbers

The rules for the natural numbers follow the pattern for all the other rules we have seen. Note that the judgement $n : \mathbf{N}$ is clearly most naturally interpreted as "n is a natural number", and $\mathbf{N}$ does not have a clear interpretation as a proposition, though it does as a set or a type. Perhaps "n is a witness to the proposition that there are natural numbers" might be one way of reading the judgement, in which, as a proposition, $\mathbf{N}$ is "there are natural numbers".

Rather than worrying too much about how we might informally interpret $\mathbf{N}$, we just rely on the following rules to give it meaning:

$$\frac{}{\mathbf{N}\ prop}\ \mathbf{N} - form \qquad \frac{}{0 : \mathbf{N}}\ \mathbf{N} - intro \qquad \frac{x : \mathbf{N}}{succ(x) : \mathbf{N}}\ \mathbf{N} - intro$$

$$\frac{n : \mathbf{N} \quad d : C(0) \quad e(x,y) : C(succ(x))\ [x : \mathbf{N}, y : C(x)]}{rec(n,d,e) : C(n)}\ \mathbf{N} - elim$$

$$\frac{d : C(0) \quad e(x,y) : C(succ(x))\ [x : \mathbf{N}, y : C(x)]}{rec(0,d,e) = d : C(0)}\ \mathbf{N} - eq$$

$$\frac{n : \mathbf{N} \quad d : C(0) \quad e(x,y) : C(succ(x))\ [x : \mathbf{N}, y : C(x)]}{rec(succ(n),d,e) = e(n, rec(n,d,e)) : C(succ(n))}\ \mathbf{N} - eq$$

These rules give us the usual interpretation of $\mathbf{N}$ as the set of natural numbers. However, we often want to talk about finite sets with a known number

of elements; as we will see, the sets with zero elements, one element and two elements turn out to be particularly important. For this reason we also have sets with k members ($k > 0$):

$$\frac{}{\mathbf{N}_k \ prop}\ \mathbf{N}_k - form \qquad \frac{}{m_k : \mathbf{N}_k}\ \mathbf{N}_k - intro,\ 0 \leq m < k$$

$$\frac{n : \mathbf{N}_k \quad a_0 : C(0_k) \ldots a_{k-1} : C((k-1)_k)}{R_k(n, a_0, \ldots, a_{k-1}) : C(n)}\ \mathbf{N}_k - elim$$

$$\frac{a_0 : C(0_k) \ldots a_{k-1} : C((k-1)_k)}{R_k(i_k, a_0, \ldots, a_{k-1}) = a_i : C(i_k)}\ \mathbf{N}_k - eq$$

$\mathbf{N}_0$ is the set containing no members; as a proposition it has no proofs, which means that we can interpret $\mathbf{N}_0$ as absurdity. Therefore $\mathbf{N}_0$-*elim*:

$$\frac{n : \mathbf{N}_0}{R_0(n) : C(n)}\ \mathbf{N}_0 - elim$$

says that if we have a proof of absurdity then *any* proposition C follows, which is exactly the rule *ex falso quodlibet*.

As usual, we can use $\mathbf{N}_0$ to define negation:

$$\neg P =_{df} P \Rightarrow \mathbf{N}_0$$

Computationally this says that a proof of $\neg P$ is a function that, given a proof of P, will construct for us a proof of $\mathbf{N}_0$, which is evidently not possible since no such proof exists.

Similarly, we can interpret $\mathbf{N}_1$ as the proposition that is true everywhere (though, of course, any nonempty type could be chosen for this role), and $\mathbf{N}_2$ can stand for the type which in programming languages is normally known as something like "Boolean"—the type containing exactly two distinct elements.

6.5 Rules for Equality

The final set of rules that we examine deal with the notion of equality. We already have equality at the judgement level, as shown by the *eq* rules in the previous sections. These rules allow us to reason about how we can compute with objects and how they transform into other objects via computation. However, it is clear from the structure of the rules that equality at the judgement level cannot be embedded within other judgements, since objects and types cannot include the equality. For example, if we want to say something simple like "a, b, and c are all equal", we cannot write

$$a = b : \mathbf{N} \wedge b = c : \mathbf{N}$$

since judgemental equality is the only equality we have so far and there is no notion of conjunction for judgements.

In order for the system to reach its full power, we want to have equality as a type; this will enable us to combine equalities together to form more complicated expressions. We need to be able to form **dependent types**—types that are parametrised by objects. To do this we need to move from an equality which appears explicitly in a judgement to its expression as a type. That means that the equality can then appear in further types (and objects in higher universes).

The rules for moving from judgements to types are straightforward:

$$\frac{a : A \quad b : A}{I(A, a, b)\ prop}\ I - form \qquad \frac{a = b : A}{e : I(A, a, b)}\ I - intro$$

$$\frac{r : I(A, a, b)}{a = b : A}\ I - elim \qquad \frac{r = e : I(A, a, b)}{r : I(A, a, b)}\ I - eq$$

Note that these rules introduce two new constants: I for forming types, and e which witnesses that two objects are the same.

We can now show, for example, that equality is symmetric. If A is a type and $a : A$ and $b : A$, and if we assume that $c : I(A, a, b)$, then we have to show that there is a witness for $I(A, b, a)$; but this follows trivially by the rules above and the equality rules from section 6.1. We can similarly show that all the other standard properties of equality hold at this type level just as they do at the judgemental level.

7 Propositions as Types

It turns out that the rules given above still make sense in general if we replace uses of "proposition" with uses of "set" or "type" and the connectives and quantifiers are replaced by various operations from set theory, as in the following table.

Propositions	**Sets**
$\vee$, disjunction	$+$, disjoint union
$\wedge$, conjunction	$\times$, Cartesian product
$\Rightarrow$, implication	$\rightarrow$, function–space constructor
$\exists$, existential	$\sum$, disjoint union over a family
$\forall$, universal	$\prod$, product over a family

Indeed, Martin–Löf's original theory was intended as a constructive set theory; the logical interpretation is recovered if we consider a proposition to be represented by the set of all its proofs.

This idea was written-up by Howard [26]. It came from the suggestive similarity between the formal descriptions of, as one case, function application and

implication elimination and, as another case, abstraction within the λ-calculus and implication introduction.

Rather than going into more details here, we direct the reader to [26], [37] and [43].

8 Mathematical considerations

The axiom of choice (in an informal form of our syntax):

$$(\forall x : A)(\exists y : B(x))C(x, y) \Rightarrow (\exists f : (\forall x : A)B(x))(\forall x : A)C(x, apply(f, x))$$

is derivable in this system. The proof, following the one in [31], goes informally as follows. Assume

$$z : (\forall x : A)(\exists y : B(x))C(x, y) \tag{2}$$

If

$$x : A \tag{3}$$

then we have

$$apply(z, x) : (\exists y : B(x))C(x, y)$$

So

$$fst(apply(z, x)) : B(x)$$

and

$$snd(apply(z, x)) : C(x, fst(apply(z, x)))$$

Now we abstract on x—that is, discharge assumption (3)—to get

$$\lambda((x)snd(apply(z, x))) : (\forall x : A)C(x, fst(apply(z, x)))$$

We also have

$$\lambda((x)fst(apply(z, x))) : (\forall x : A)B(x)$$

so

$$apply(\lambda((x)fst(apply(z, x))), x) = fst(apply(z, x)) : B(x)$$

Hence, by substitution,

$$C(x, apply(\lambda((x)fst(z, x)), x)) = C(x, fst(apply(z, x)))$$

and therefore

$$\lambda((x)snd(apply(z, x))) : (\forall x : A)C(x, apply(\lambda((x)fst(z, x)), x)$$

Existential introduction now yields

$$(\lambda((x)fst(apply(z, x))), \lambda((x)snd(apply(z, x)))) :$$

$$(\exists f : (\forall x : A)B(x))(\forall x : A)C(x, apply(f, x))$$

and so, by abstraction on z—that is, by discharging our first assumption 2—we get

$$\lambda((z)(\lambda((x)fst(apply(z, x))), \lambda((x)snd(apply(z, x))))) :$$
$$(\exists f : (\forall x : A)B(x))(\forall x : A)C(x, apply(f, x))$$

This completes the proof of the axiom of choice.

We also want to be sure that we can do arithmetic in our theory. This we can show by considering Peano's axioms. Only one of the five axioms—the fourth one, which says that 0 is not the successor of any natural number—is not already available by simple constructions using the rules we have introduced above. In order to prove this axiom, we have to introduce **universes.**

These can be regarded as an extension to the system that allows the idea "every object has a type" to appear uniformly (or, equivalently, that allows every object to be a member of some set). In particular, the propositions or types or sets are objects in the theory that do not themselves have sets in which to reside.

A more general problem is that the theory as it stands allows to construct only finitely many new sets—for example, we cannot construct a function which, given some natural number n, returns the n–fold product of $\mathbf{N}$ with itself: such a function has no type within the system.

For similar reasons we cannot hope to model the important and powerful idea of abstract data types. Such types would typically be defined by stating the existence of a type with various desired properties. So we would expect such an object to reside in a type of the form $\exists(A, B)$. But we do not currently have a type that B could be; in other words, we do not have a type that could contain the abstract type. We shall say more on this, with examples, towards the end of this paper.

Finally, we might want, for programming purposes, to be able to write functions which take types as arguments, thereby allowing us to model ideas like parametric polymorphism. Again, we currently have no way of writing down the type of such a function, so it certainly cannot be constructible in the current system.

For all of these reasons we need to extend the language to include a type that contains all our current types, so that our current types are themselves objects in this new type. The type that contains all the types we have seen so far is denoted by U_0, and we have new rules such as

$$\frac{}{U_0 \ type}\ U-form1 \qquad \frac{A \ type}{A \ : \ U_0}\ U-form2 \qquad \frac{}{\mathbf{N} \ : \ U_0}\ \mathbf{N}-form$$

$$\frac{A \ : \ U_0 \quad B(x) \ : \ U_0}{\forall(A, B) \ : \ U_0}\ U_0-intro$$

In the rules we had previously, all occurrences of $A \ prop$ are replaced by $A \ : \ U_0$.

We can now construct type–valued functions like

$$\lambda((x)rec(x, \mathbf{N}, (x, y)(\mathbf{N} \times y))) \ : \ \mathbf{N} \to U_0$$

In particular, we can show that the fourth Peano axiom, which we express as

$$I(\mathbf{N}, 0, succ(n)) \to \mathbf{N}_0 \ [n \ : \ \mathbf{N}]$$

is derivable in the theory, as follows.

First assume that

$$x : I(\mathbf{N}, 0, succ(n)) \ [n \ : \ \mathbf{N}] \tag{4}$$

We can show, using the U_0-*intro* rules, that

$$rec(m, \mathbf{N}_1, (y, z)\mathbf{N}_0) \ : \ U_0 \ [m \ : \ \mathbf{N}]$$

The $\mathbf{N}$-*eq* rules give us

$$rec(0, \mathbf{N}_1, (y, z)\mathbf{N}_0) = \mathbf{N}_1 \ : \ \mathbf{U}_0 \tag{5}$$

and

$$rec(succ(n), \mathbf{N}_1, (y, z)\mathbf{N}_0) = \mathbf{N}_0 \ : \ U_0 \ [n \ : \ \mathbf{N}] \tag{6}$$

By I-*elim* on 4, we have

$$0 = succ(n) : \mathbf{N}[n : \mathbf{N}]$$

and from this it follows that

$$rec(0, \mathbf{N}_1, (y, z)\mathbf{N}_0) \ = \ rec(succ(n), \mathbf{N}_1, (y, z)\mathbf{N}_0) : \ U_0 \ [n : \mathbf{N}]$$

Further, from 5 and 6 we obtain

$$\mathbf{N}_1 = \mathbf{N}_0 \tag{7}$$

Since $\mathbf{N}_1$-*intro* yields

$$0_1 : \mathbf{N}_1$$

we also have

$$0_1 : \mathbf{N}_0$$

by 7; so, by discharging the assumption 4, we finally have

$$\lambda((x)0_1) : I(\mathbf{N}, 0, succ(n)) \to \mathbf{N}_0 \ [n : \mathbf{N}]$$

Now that we have a type U_0 that contains all our old types, there remains the question of what type U_0 itself appears in and whether we can extend the theory so that objects like $\forall(A, U_0)$ can also be admitted as elements of some type. The answer is that we add another type U_1 which contains U_0 and all the elements built from it using the usual type constructors. In fact this sequence of types can be extended so that we get U_n for any natural number n. The one thing that we cannot have is a type that contains all types including itself; that would make the system inconsistent.

9 Program specification and Derivation

Having reviewed much of the formal machinery, we are, at last, in a position to say something about how it is put to use in programming.

One of the central problems in computer science is to develop a program p that meets—that is, correctly implements—a given specification S. There are of course other problems linked to this one:

- How do you develop the specification itself?
- How do you know that the specification correctly expresses what the customer wants?
- How do you manage change in specifications (perhaps as required by changing customer or technological requirements) as time passes, and how do you reflect these faithfully in the program?

All these problems are very real and important, and are the object of much research, but we will ignore them in what follows.

The problem on which we concentrate can be expressed within the system presented above as

> *given a type S, which should be viewed as a specification, derive a program p such that $p : S$*

So we view specifications as either types or propositions. Viewing them as types, we wish to construct an element of the type. Viewing them as propositions, we wish to show that the specification is provable (in other words, that it does not express an impossible state of affairs); moreover, since we are working in a constructive logic, we will then use the witness as a program which meets the specification.

This approach has several advantages, amongst which are

- that the specification and program development process (building a derivation of p) all go on in one system, and
- that a program is at once a computational object (so it can carry out the task set by the specification) and a proof that the specification has been met.

9.1 A Simple Example

An example of a specification is one for a natural number division algorithm:

$$\forall(\mathbf{N},(n)\forall(\mathbf{N},(m)\exists(\mathbf{N},(k)\exists(\mathbf{N},(r)I(\mathbf{N},n,plus(prod(m,k),r)))))) \qquad (8)$$

where we already have terms

$$plus =_{df} (x, y)rec(x, y, (a, b)succ(b))$$

for addition and

$$prod =_{df} (x, y)rec(x, 0, (a, b)plus(y, b))$$

for multiplication.

Note that (8) states that for any two natural numbers their quotient and remainder exist, which is what we expect if we are defining division. But note that because this is a constructive logic, the proof not only shows us that this is the case but also explicitly computes the quotient and remainder. Indeed, the proof object that we would construct for (8) would be of the form

$$\lambda((n)\lambda((m)(k, (r, p))))$$

Applying this object to natural numbers a and b would return a structure containing k, r and p where k is the quotient, r the remainder, and p a proof that $a = (b \times k) + r$.

9.2 Abstract Data Types

One of the most important ideas to emerge from studies of good programming practice is that of **separation of concerns**. This refers to the fact that in building large pieces of software, we have to solve highly complex problems which usually require several people working concurrently (for reasons of economy or efficiency, for example). This means that the division of labour amongst the programmers has to be carefully considered so that inconsistencies in assumptions about properties of the system being built do not cause the system to fail when all the separately built parts are brought together. One way of dealing with this is to identify structures which can be logically separated out from the rest of the problem and allow two views of them—the view of the person implementing them and the view of the person using them.

These views share part of the structure, a part known as the **interface**. This names the operations provided by the data type and gives their types, so that the user knows what the type makes available. It also tells the person implementing the program what operations and types have to be implemented; the interface can be viewed as a contract between the two sides. Then the user knows about the structure only as far as the interface describes it. Since this means that, for the user, the way that the structure is implemented is hidden and inaccessible, such a structure is known as an **abstract data type** (ADT). This separation of implementation and usage for an ADT means that if, for some later reason, perhaps a change of hardware or an improved algorithm for some aspect of the ADT, the implementer wants to change a part, then because

the user of the ADT has used only the operations provided by the interface and has had no access to the implementation, any software the user has written does not have to change. It also means that the user and implementer can work concurrently on the implementation and use of the ADT, since they each only have to respect the interface and their concerns have been separated.

Having described the importance of the ADT idea, we now have to describe how ADTs can be modelled within the system we have been presenting.

One ADT commonly used as a building–block for many other structures is the list. Informally, a **list** is a sequence of elements from some type where order is significant and repeated occurrences of elements are allowed. There is a distinguished element, the **empty list**, and a binary operation, usually called **cons**, which adds an element to the start, or **head**, of a list.

In specifying the list ADT, we have to state that such a type exists and that each of the operations that allow us to compute with lists exists also; so it is not surprising that the type that models the ADT has the outermost form of an existential proposition, or what has become widely known in computer science as an **existential type**.

We will first consider a list of natural numbers. We can write it as

$$\exists(U_0,(L)\exists(L,(e)\exists(L\Rightarrow\mathbf{N},(h)\exists(L\times\mathbf{N}\Rightarrow L,(c)\forall(\mathbf{N},(n)\forall(L,$$

$$(l)(I(\mathbf{N},apply(h,apply(c,(n,l))),n)\wedge I(L,apply(h,e),e))))))))$$

An object in this type has the form

$$(list,(empty,(head,(cons,\lambda((n)\lambda((l)p))))))) \tag{9}$$

where *list* is the type whose existence is claimed by the type (read as a proposition), *empty*, *head,* and *cons* are the various operations which form part of the ADT, and the last component is a proof that, for any natural number and any list, the operations satisfy the equalities that define them.

We can generalise the ADT for lists of natural numbers to allow it to be parametrised by the underlying type. This gives us a single ADT which can be specialised to any underlying type—including, for example, the ADT for lists itself. The generalisation is very easy: we simply add another level of quantification, as follows.

$$\forall(U_0,(T)\exists(U_0,(L)\exists(L,(e)\exists(L\Rightarrow T,(h)\exists(L\times T\Rightarrow L,(c)\forall(T,(n)\forall(L,$$

$$(l)(I(T,apply(h,apply(c,(n,l))),n)\wedge I(L,apply(h,e),e)))))))$$

An object of this type has the form

$$\lambda((t)(list,(empty,(head,(cons,\lambda((n)\lambda((l)p))))))))$$

which, when applied to some type T (which is bound to t), has as value an object like that in (9) but with the underlying type T instead of the fixed type $\mathbf{N}$ we had before.

9.3 Further work

An example of ongoing work in this area is that of providing simpler and more elegant semantics for the specification language Z than currently exists (see [23]). For reasons closely linked with the work of Martin-Löf presented above, this is being done by examining formal systems for intuitionistic logic. The point is that an intuitionistic basis for Z will yield not only a logic for Z as a specification language but also a logic for program derivation, in the sense that we will be able to derive programs that meet given Z specifications in much the same way as, above, we have been able to derive programs from the types, propositions, or sets treated there.

Although it turns out that giving such a logic for Z is fairly straightforward, we are still left with the problem of making the process of derivation meaningful to a programmer rather than a person working in intuitionistic logic. The rules that give the program derivation steps are very primitive, and it is usually the case that many of these primitive rules are required to make a derived rule which encapsulates one step at the level at which a programmer would normally work. So the larger challenge is to develop, from the primitive rules provided by the underlying logical system, derived rules that match a programmer's view of program derivation from Z specifications.

This is a clear illustration of the difference between work in formal logic (which has the distinctive characteristic that no one ever really wants to do a proof *within* the formalism, only *about* the formalism) and computer science (where we do want to develop formal systems which are usable). While the formal systems are invaluable as vehicles for expressing the semantics and logic of our programming endeavours, they have nothing to offer in the way of methods for actually making derivations within them.

Making such formal systems practicable has also given rise to a huge volume of work on the development of software supporting uses of formal systems, in the sense of syntax checkers, type checkers, proof checkers and proof assistants and theory managers (systems which store, index, allow retrieval of, and ensure the consistency of the huge formal theories that programming logics depend on). A simple example of such a system is described in [38]. It should be noted that work in this area of proof assistants is still at an early stage and there are many unsolved problems, not the least of which is to develop good interfaces to such systems. Too often the system is developed and used by a small team of people who get to know it so well that they lose sight of the fact that new users would find it very hard to use because little attention has been paid to the modes of interaction with the system and, in particular, to making those modes clear and understandable for a new user.

Computer science can be seen a discipline which has both revived the need for formal systems and seen them put to practical use. In this respect, constructive mathematics and its underlying formal systems have proved, and are likely to continue to be, of paramount importance.

References

[1] H. P. Barendregt, *The Lambda Calculus,* North–Holland, Amsterdam, 1984.

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

[3] Errett Bishop, *Foundations of Constructive Analysis,* McGraw-Hill, New York, 1967.

[4] Errett Bishop, "Mathematics as a numerical language", in *Intuitionism and Proof Theory* (A. Kino, J. Myhill, and R.E. Vesley, eds), 53-71, North–Holland, Amsterdam, 1970.

[5] Errett Bishop, "Schizophrenia in contemporary mathematics", in *Errett Bishop: Reflections on Him and His Research* (Murray Rosenblatt, ed.), Contemporary Mathematics **39**, 1-32, American Math. Soc., Providence RI, 1984.

[6] E.A. Bishop and D.S. Bridges, *Constructive Mathematics*, Grundlehren der math. Wissenschaften **279**, Springer–Verlag, Heidelberg, 1985.

[7] Nicolas Bourbaki, *Elements of the History of Mathematics* (translated from the French by John Meldrum), Springer-Verlag, Heidelberg, 1991.

[8] Douglas Bridges, "A constructive development of Chebyshev approximation theory", J. Approx. Th. **30**(2), 99-120, 1980.

[9] Douglas Bridges, "A constructive proximinality property of finite-dimensional linear spaces", Rocky Mountain J. Math. **11**(4), 491-497, 1981.

[10] Douglas Bridges, "A constructive analysis of the Remes algorithm", J. Approx. Theory **32**(4), 257-270, 1981.

[11] Douglas Bridges, "Recent progress in constructive approximation theory", in *The L.E.J. Brouwer Centenary Symposium* (A.S. Troelstra and D. van Dalen, eds), 41-50, North-Holland, Amsterdam, 1982.

[12] Douglas Bridges, "Constructive Truth in Practice", to appear in *Truth in Mathematics* (Proceedings of the conference held at Mussomeli, Sicily, 13-21 September 1995, H.G. Dales and G. Oliveri, eds), Oxford University Press, Oxford, 1997.

[13] Douglas Bridges, *Constructive Mathematics—Its Set Theory and Practice*, D.Phil. thesis, Oxford University, 1975.

[14] Douglas Bridges, "A constructive Morse theory of sets", in *Mathematical Logic and Its Applications* (D.G. Skordev, ed.), Plenum Press, New York, 1987.

[15] Douglas Bridges,"Constructive Mathematics: A Foundation for Computable Analysis", to appear in Proc. Dagstuhl Workshop on *Computability and Constructivity in Analysis* Dagstuhl, Germany, April 21-25, 1997).

[16] Douglas Bridges and Osvald Demuth, "On the Lebesgue measurability of continuous functions in constructive analysis", Bull. Amer. Math. Soc. 24(2), 259-276, 1991.

[17] Douglas Bridges and Fred Richman, *Varieties of Constructive Mathematics,* London Math. Soc. Lecture Notes 97, Cambridge University Press, 1987.

[18] L.E.J. Brouwer, *Over de Grondslagen der Wiskunde,* Doctoral Thesis, University of Amsterdam, 1907. Reprinted with additional material (D. van Dalen, ed.) by Matematisch Centrum, Amsterdam, 1981.

[19] M.A.E. Dummett, *Elements of Intuitionism.* Oxford University Press, Oxford, 1977.

[20] S. Feferman, "Constructive theories of functions and classes", in: *Logic Colloquium '78* (M. Boffa, D. van Dalen, K. McAloon, eds), North–Holland, Amsterdam, 1979.

[21] H. Friedman, "Set theoretic foundations for constructive analysis", Ann. of Math. **105**, 1-28, 1977.

[22] N.D. Goodman and J. Myhill, "Choice implies excluded middle", Zeit. Logik und Grundlagen der Math. **24**, 461, 1978.

[23] M.C. Henson and S. Reeves, "Intensional Z" (extended abstract), in *FMP '97: Proceedings of Formal Methods Pacific '97* (L. Groves and S. Reeves, eds), 305–306, Springer–Verlag, Singapore, 1997.

[24] A. Heyting, *Intuitionism—An Introduction (*Third Edition). North-Holland, Amsterdam, 1971.

[25] David Hilbert, "Die Grundlagen der Mathematik", Hamburger Mathematische Einzelschriften 5, Teubner, Leipzig, 1928. Reprinted in English translation in [44], in which the exact quotation appears on page 476.

[26] W.A. Howard, "The formula–as–types notion of construction", in *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (J.P. Seldin and J.R. Hindley, eds), Academic Press, 1980.

[27] S. Karlin and W.J. Studden, *Tchebycheff Systems: With Applications in Analysis and Statistics*, Interscience, New York, 1966.

[28] B.A. Kushner, *Lectures on Constructive Mathematical Analysis*, Amer. Math. Soc., Providence RI, 1985.

[29] P. Martin-Löf, "An intuitionistic theory of types: predicative part", in *Logic Colloquium 1973* (H.E. Rose and J.C. Shepherdson, eds), 73-118, North–Holland, Amsterdam, 1975.

[30] P. Martin-Löf, "Constructive mathematics and computer programming", in *Proceedings of 6th International Congress for Logic, Methodology and Philosophy of Science* (L. Jonathan Cohen ed), North–Holland, Amsterdam, 1980.

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

[32] P. Martin-Löf, "Constructive mathematics and computer programming", in *Mathematical Logic and Programming Languages* (C.A.R. Hoare and J.C. Shepherdson, eds), Prentice–Hall International, Englewood Cliffs, N.J.,1985.

[33] Ray Mines, Fred Richman, Wim Ruitenburg, *A Course in Constructive Algebra,* Universitext, Springer-Verlag, Heidelberg, 1988.

[34] A.P. Morse, *A Theory of Sets*, Academic Press, New York, 1965.

[35] John Myhill, "Some properties of intuitionistic Zermelo–Fraenkel set theory", in *Cambridge Summer School in Mathematical Logic* (A. Mathias and H. Rogers, eds.), 206–231, Lecture Notes in Mathematics **337**, Springer–Verlag, Berlin, 1973.

[36] John Myhill, "Constructive Set Theory", J. Symbolic Logic **40**(3), 347-382, 1975.

[37] S. Reeves, "Constructive Mathematics and programming", in *Mathematical Structures for Software Engineering* (B. de Neumann, D. Simpson, and G. Slater, eds), 219–246, Oxford University Press, 1991.

[38] S. Reeves, "Computer support for students' work in a formal system: Macpict", Int. J. Math. Education in Science and Technology **26**(2), 159–175, 1995.

[39] Fred Richman, "The fundamental theorem of algebra: a constructive development without choice", at `html://www.math.fau.edu/Richman/html/docs.htm`

[40] Fred Richman, "Intuitionism as generalization" Philosophia Math. **5**, 124-128, 1990 (MR #91g:03014).

[41] Fred Richman, "Interview with a constructive mathematician", Modern Logic **6**, 247–271, 1996.

[42] A.S. Troelstra and D. van Dalen, *Constructivity in Mathematics: An Introduction* (two volumes). North Holland, Amsterdam, 1988.

[43] S. Thompson, *Type Theory and Formal Programming,* Addison–Wesley, Wokingham, England, 1991.

[44] Jean van Heijenoort, *From Frege to Gödel, A Source Book in Mathematical Logic 1879-1931,* Harvard University Press, Cambridge, Mass., 1967.

[45] W.P. van Stigt, *Brouwer's Intuitionism,* North-Holland, Amsterdam, 1990.