1 Introduction

These notes summarize ideas discussed up to and including Lecture 11, and they add material related to the assigned reading on the simple theory \( Q \) and on primitive recursive functions. The course textbook *Type Theory and Functional Programming* by Simon Thompson on the course web page covers the natural numbers in several sections, e.g. 2.9, 4.8, and 6.1. In due course, we will connect to the textbook approach. First we compare and contrast the standard ways of presenting numbers.

1.1 Overview

These notes and the lectures on them compare the standard ways of explaining the natural numbers and how we reason about them. As we mentioned in lecture, the history of the foundations of the natural numbers is fascinating. The story began with an attempt to understand the real numbers. Cauchy explained them in terms of sequences of rational numbers. That left the question of understanding sequences and rational numbers. Sequences were reduced to functions on natural numbers, and rational numbers were reduced to pairs of integers. Integers could be reduced to signed natural numbers.

These ideas about numbers are fundamental to understanding how computer scientists, mathematicians, and programmers reason about them and create numerical algorithms that are known to be correct by the very method of constructing them.

To know for certain that algorithms are correct, we must know precisely what we require them to accomplish. That information is given by mathematical specifications. Those specifications are grounded in the ways we
understand numbers. There are at least three distinct ways that specifications are given precisely.

The first approach is *axiomatic* and abstract. We state logical properties of the numbers using first-order logic. It might be the case that these first-order properties describe numbers so well that they capture our intuition completely. The classical first-order theory of numbers is called *Peano Arithmetic* (PA). The constructive theory is called *Heyting Arithmetic*, and differs in just one axiom from PA, e.g. HA does not use the law of excluded middle (or the equivalent double negation elimination).

These established first-order theories, while simple and elegant, are not the route by which any of us learned to compute with numbers and grasp them intuitively. Moreover, we will see more deeply that this is not an entirely adequate way to understand numbers, even with the constructive semantics of HA.

We already know that the very simple theory Q is too weak. This is quite easy to see. We noted in Lecture 11 that we need some form of induction that is not present in Q. We continue this line of investigation here.

Another approach is more computational than constructive first-order logic; we define methods of computing recursive functions over the natural numbers. This corresponds more closely to how we are taught about numbers as children, e.g. how to add and multiply and to how we are taught about functions in programming languages. This approach became the basis for the very definition of numbers in constructive type theory. We examine this method based on early draft notes from Per Martin-Löf from the late 1960’s.

The Martin-Löf approach will lead us to constructive type theory in a very natural way. In that setting, we will see the richest computational account of numbers available. It’s connection to programming languages will be clearly manifest as we go.

Type theory leads us to powerful new computational ideas and to a unification of insights discovered by Church and Curry when they were examining an approach to mathematics based on computable functions and types. This approach is closely related to the insights of Brouwer. That connection was a late discovery and a discovery that has been enhanced and clarified in the type theory of Nuprl [2] which manages to unify all three main approaches to the natural numbers discussed in the literature.

### 1.2 Axiomatic approach to numbers

We have already informally identified different approaches to understanding the natural numbers, 0,1,2,.... The first is based on a very simple theory of
numbers axiomatized in first-order logic. The theory Q defined by Tarski, Mostowski, and Robinson circa 1953, see [1]. We have axiomatized this in intuitionistic first-order logic.

Our first axiomatization could be called pure logic. We used only predicates and relations to define the theory. There are no constants and no functions involved.

The typical use of first-order logic to define mathematical theories involves an equality relation since equality is a basic relation in mathematics. In a sense, mathematical objects come with a precise notion of equality. The equality relation is the basis of the abstract character of mathematics. Here are the standard axioms for equality. Equality is a binary relation. It has the type \( \text{Eq} : (D \times D) \to \text{Prop} \). Thus for two elements of \( D \), we can form the proposition \( \text{Eq}(x, y) \). Intuitively we understand what equality means on numbers, but we will see that this relation is subtle, and the axiomatic approach is abstract, so we don’t have a strong concrete idea of equality when we approach it from the point of view of axiomatic first-order logic, the standard approach in many mathematical textbooks.

**Axiom Equality1**: Equality is reflexive, \( \forall x : D.\text{Eq}(x, x) \).

**Axiom Equality2**: Equality is symmetric, \( \forall x, y : D.\text{Eq}(x, y) \Rightarrow \text{Eq}(y, x) \).

**Axiom Equality3**: Equality is transitive, \( \forall x, y, z : D.\text{Eq}(x, y) \Rightarrow \text{Eq}(y, z) \Rightarrow \text{Eq}(x, z) \).

Any theory attempting to characterize the natural numbers must recognize the special role of zero. To characterize zero in logic, we use a predicate Zero\( (x) \) asserting that an object is of the domain of discourse. We assert that zero exists by postulating the “zero property, Zero\( (x) \).”

**Axiom Zero**: The domain of discourse \( D \) includes an element, \( \exists x : D.\text{Zero}(x) \).

The realizer for this axiom is an ordered pair, \( < d, \text{iszero} > \). Given the realizer, we can define its first component, \( \text{spread}(< d, \text{iszero} >; z, iz.z) \). It would be convenient to display this element of \( D \) as 0. We know Zero\( (0) \), and the evidence is \( \text{spread}(< d, \text{iszero} >; z, iz.iz) \).

All predicates and relations over \( D \) must respect the equality relation, so we require this axiom as well.

**Axiom Zero Unique** Zero is unique, \( \forall x, y : D.\text{Zero}(x) \Rightarrow \text{Zero}(y) \Rightarrow \)
We intend that there is a unique zero element among the numbers, so we include the axiom of uniqueness.

**Axiom Zero Unique**: The zero element of $D$ is unique, $\forall x, y : D.\text{Zero}(x) \& \text{Zero}(y) \Rightarrow Eq(x, y)$.

In constructive first-order logic we can also state another key property, that we can recognize the unique zero object.

**Axiom Zero Decidable** zero is recognizable, i.e. “zeroness” is decidable, $\forall x : D.\text{Zero}(x) \lor \neg \text{Zero}(x)$.

Zero is the starting point for generating the natural numbers. The other numbers are created as successors of zero. One way to describe them logically is to define the successor relation. So we introduce the binary relation, $\text{Suc}(x, y)$ to say that the element $y$ of $D$ is the successor of the element $x$.

**Axiom Successor Relation**, successor is a single valued binary relation on numbers, $\forall x, y, z : D.\text{Suc}(x, y) \Rightarrow \text{Suc}(x, z) \Rightarrow Eq(y, z)$.

The key axiom about successor is that given any number, we can find its unique successor.

**Axiom Successor Function**, we can find the successor of any number, $\forall x : D. \exists y : D.\text{Suc}(x, y)$.

We know from the constructive meaning of the logical operators that this axiom gives us a realizer for creating the successor of any number. Let us name the realizer as $\lambda(x. \text{spair}(x))$. From this we can create a function that finds the successor element, namely $\lambda(x. \text{spread}(\text{spair}(x); s, se.s))$. The $s$ is an element of $D$ and the object $se$ is a witness to $\text{Suc}(x, s)$. Let $S$ be this function, so given any element $n$ of $D$, $S(n)$ is its unique successor. We know that $S$ behaves like a function in the sense that if $Eq(n, m)$, then $S(n) = S(m)$.

We also know $\text{Suc}(0, S(0))$, and by repeating the successor operation, we know $\text{Suc}(S(0), S(S(0)))$.

These simple axioms give us a basis for defining the theory $Q$ by adding the following seven additional axioms.
**Axiom Q1**, successor is injective, \( \forall x, y : D.Eq(S(x), S(y)) \Rightarrow Eq(x, y) \).

If we write \( Eq(s, t) \) for \( s = t \) where \( s \) and \( t \) are terms defined from the realizers, then this axiom has the same form as in Boolos & Jeffrey, namely \( \forall x, y : D.S(x) = S(y) \Rightarrow x = y \).

**Axiom Zero not a Successor Q2**, \( \forall x. \sim Succ(x, 0) \).

**Axiom Predecessor Q3**, nonzero elements have a predecessor, \( \forall x : D.\sim Zero(x) \Rightarrow \exists y : D.S(y) = x \).

If we use the set type notation to create the type \( \{ x : D \mid \sim Zero(x) \} \), we can then create a function, \( pred \) with the type \( \{ x : D \mid \sim Zero(x) \} \rightarrow D \). The axiom then have the simple form

**Axiom Q3’**, \( \forall x : \{ z : D \mid \sim Zero(z) \}.S(pred(x)) = x \).

The next two axioms of Q define the addition and multiplication functions. We need to do this using relations, \( Add(x, y, z) \) and \( Mult(x, y, z) \), We first need the axioms that these relations respect equality. We write this for addition and leave the axiom for multiplication as an exercise.

**Axiom Relation**, the addition relation respects equality,

\[ \forall x, x', y, y', z, z' : D.Mult(x, y, z) \Rightarrow Eq(x, x') \Rightarrow Eq(y, y') \Rightarrow Mult(x', y', z') \Rightarrow Eq(z, z') \.\]

**Axiom Addition of Zero Q4**, addition of zero, \( \forall x, y : D.Add(x, 0, x) \).

**Axiom Addition of Successor Q5**, addition of \( x \) and a successor, \( \forall x, y, z : D.Add(x, y, z) \Rightarrow Add(x, S(y), S(z)) \).

**Axiom Multiplication by Zero Q6**, multiplication of \( x \) and zero, \( \forall x : D.Mult(x, 0, 0) \).

**Axiom Multiplication by Successor Q7**, multiplication of \( x \) and a successor, \( \forall x, y, z, w : D.Mult(x, s(y), z) \Rightarrow Add(z, x, w) \Rightarrow Mult(x, S(y), w) \).

This simply says that \( x \times S(y) = (x \times y) + x \).

These are axioms that allow us to calculate in an awkward way with numbers in a unary notation. However, we cannot prove simple theorems such as these.
We can construct an addition function, $\forall x, y : D. \exists z : D. \text{Add}(x, y, z)$.

We can construct a multiplication function, $\forall x, y : D. \exists z : D. \text{Mult}(x, y, z)$.

We will see what must be done to allow these obviously necessary constructions.

If we try to construct these functions by induction, we will see that we do not have axioms that justify this method. The effort to define induction will lead us to an alternative approach to number theory, one that starts with the mechanism of primitive recursion to define an entire class of functions. This approach treats total function definition as the main method of understanding numbers.

2 Primitive Recursive Functions

We look at an untitled early draft notes by Per Martin-Löf mentioned earlier. We will also look at his influential writings in the early 1980’s [3, 4, 5]. This early article, perhaps 1969 begins as follows:

“About the simplest mathematical language for which a non-trivial theory of meaning can be formulated, is the language in which the only objects and functions dealt with are the natural numbers and the primitive recursive functions. Trivial as this language may seem, it is nevertheless rich enough to allow us to frame answers to such questions as What is a natural number? What is zero? What is the successor of a natural number? Moreover, the answers that we shall give to these questions will serve as paradigms for our answers to the more general questions What is a type? What is a mathematical object of a given type? which we will pose later for the theory of types.

2.1 Primitive recursive arithmetic

The Swedish mathematician and philosopher Martin-Löf is building on a tradition of study that traces its origins to writings of the Swedish logician Thoralf Skolem who developed a theory of recursive arithmetic in which he gave up the notion of existence expressed as a quantifier, $\exists y : D. P(x, y)$ in favor of defining recursive functions that compute the object claimed to exist. He called his theory recursive arithmetic and developed it starting in 1939. The idea was to express the logical claim $\forall x : D. \exists y : D. P(x, y)$ as $\forall x : D. P(x, f(x))$ for $f$ a primitive recursive function.

These recursive functions were given using a simple syntax, which in the case of two arguments, binary functions, is given as follows for addition and multiplication. Let $s(x)$ denote the ambiguous value of the successor
function. Skolem did not use Church’s lambda notation as we do. We would write this as \(\lambda(x.s(x))\). On the other hand, Skolem’s notation is very clear and intuitive, and it established a tradition adopted in logic and used to this day. We list three functions defined in this manner, all familiar recursive functions used in modern programming languages. Then we add a fourth less common function, hyper-exponentiation and describe a pattern that leads to the famous Ackermann function, a function known to be non-primitive recursive while all of the \(a_i(x, y)\) are primitive recursive.

1. \(\text{add}(x, 0) = x\)
2. \(\text{add}(x, s(y)) = s(\text{add}(x, y))\)
1. \(\text{mult}(x, 0) = 0\)
2. \(\text{mult}(x, s(y)) = \text{add}(x, \text{mult}(x, y))\)
1. \(\text{exp}(x, 0) = s(0)\)
2. \(\text{exp}(x, s(y)) = \text{mult}(y, \text{exp}(x, y))\)
1. \(\text{hypeexp}(x, 0) = s(x)\)
2. \(\text{exp}(x, s(y)) = \text{exp}(y, \text{hypeexp}(x, y))\)

The general pattern here is that we are defining a sequence of functions, say \(a_i(x, y)\) starting with \(a_0(y) = s(y)\). Each definition includes a recursive call to the function being defined. This call is always on a value less than the principal input argument. In these examples that argument is the second input.

1. \(a_{i+1}(x, 0) = y\)
2. \(a_{i+1}(x, s(y)) = a_n(x, a_{i+1}(x, y))\)

The Ackermann function is essentially \(\lambda(x.a_x(x, x))\). It grows faster than all primitive recursive functions, which is not hard to prove, and hence cannot be equal to any of them. Nevertheless, it appears to be a perfectly sensible computable function. There are mathematicians, such as Edward Nelson at Princeton, who do not believe that the definition of the exponential function is justified. He claims in his book Predicative Arithmetic, Mathematical Notes 32, Princeton University Press, 1986, that this function
is infeasible to compute and does not make sense. Can you see how the exponential function, \( a_3 \) differs in an essential way from the previous functions in the sequence?

We will see a strong connection between induction and primitive recursion. One connection is that if we express properties of these functions using the existential quantifier, then induction on the principal argument is a natural way to state that the recursive definition defines a total computable function, i.e. one that terminates on all inputs. For example, consider this theorem.

**Addition is a total function:**
\[
\forall x, y : \mathbb{N}. \exists z : \mathbb{N}. (\text{add}(x, y) = z).
\]

The proof is by induction on \( y \). In the base case, \( y = 0 \) we see that \( z = x \). If we assume \( \exists z : \mathbb{N}. (\text{add}(x, y) = z) \), then as we saw in lecture it is possible to prove by induction \( \exists z : \mathbb{N}. (\text{add}(x, S(y)) = z) \).

In a way this defeats the reason Skolem used primitive recursive functions, on the other hand, it shows that induction provides an explanation for termination of the \text{add} function.

Exercises. In Lecture 12 we discussed some of the questions that will appear on Problem Set 4. One of them is to finish this proof and discuss its contribution to believing that the addition function is total. Note that the argument applies equally well to all of the primitive recursive functions discussed above.

Another topic we mentioned in lecture is that the definition of the exponential function is somewhat more complex than for addition and multiplication. Can you see this extra complexity. The Princeton mathematician Edward Nelson believes that the exponential function should not be considered computable. Can you think of a reason for his opinion?

**References**


