CS3110 Spring 2017 Lecture 12: DRAFT: Constructive Real Numbers continued

Robert Constable

Reminder: Prelim in class next Tuesday. It will not cover the real numbers beyond lecture 11 and comments on lecture 11 added here. Part of this lecture will be devoted to a review for the closed book in class prelim.

	Date for	Due Date
Prelim	Tue. March 14, in class	
PS3	Out on Thur. March 2	March 16
PS4	Out on March 16	March 30
PS5	Out on April 10	April 24
PS6	Out on April 24	May 8 (day of last lecture)

1 Properties of the constructive real numbers

- Bishop's preliminaries
- *operation* versus *function* and the importance of equality
- Bishop uses "set" where we say "type". He requires a notion of equality of elements for every set.
- One-one function is a correspondence.

– *Countably infinite* is 1-1 correspondence with $\mathbb{Z}^+$.

- $\mathbb{Q}$ is countably infinite because we can enumerate the rationals.

– *Finite sets* iff 1-1 correspondences with $\mathbb{Z}_n = \{0, 1, ..., n-1\}$ (could use $\{1, 2, ..., n\}$)

– To say that a set is *subfinite* is to say that it has *at most* n elements.

- Bishop uses *constructive logic*, which we related to the OCaml "logical types" and to the Coq logical operators.

$A \Rightarrow B$	iff	there is a computable function from evidence for A to evidence for B.
$A \& B$	iff	we know a pair consisting of evidence for A and B.
$A \vee B$	iff	there is evidence for A (tagged "left") or for B ("right").
$\sim A$	iff	$A \Rightarrow$ False False is the empty type.
$\forall x{:}T.A(x)$	iff	there is a computable function f taking any $t \in T$ to evidence for $A(t)$
$\exists x{:}T.A(x)$	iff	there is a pair $<t, a>$ where $t \in T$ and a is evidence for $A(t)$

Note: $\forall x$ and $\exists x$ are examples of *dependent types*. (See attached pages 10-11 from Bishop.)

1.1 From the rational numbers to the reals

The numbers studied in early Greek mathematics were the rational numbers. The idea of ratios was important in Greek philosophy and ontology. It was a shock when Pythagoras discovered irrational numbers, and it became a challenge of mathematics from that point onward to understand what we now know as the real numbers.

The key discovery of Pythagoras was that in a square with sides s and hypotenuse h, the square of the hypotenuse is equal to the sum of the squares of the sides, i.e. $h^2 = s^2 + s^2$. Already from this Pythagoras saw the existence of a number that is not rational. Let the sides of the square be of unit length, then we have $h^2 = 1 + 1$. But h cannot be a ratio a/b because then $(a/b)^2 = 2$, hence $a^2 = 2b^2$. Thus 2 divides a^2 an even number of times and 2 divides $2b^2$ an odd number of times. Hence there is no solution to $a^2 = 2b^2$ in the integers. We can also directly prove that the square root of 2, $\sqrt{2}$, is not a rational number. Once it was known that there is such an entity as an irrational number, many deep mathematical questions arose.

The study of this topic eventually led to the formulation of the mathematical notion of the *real numbers*; it took until the late 1800's for this concept to become clear and well understood. Here are some of the historical highlights.

By 1835 William Hamilton offered a precise definition of the irrational numbers as limits of a sequence of rational numbers. He published this in his book *Algebra as the Science of Pure Time*. Oddly enough, it was not until 1860 that the notion of a rational number as a pair of integers was proposed and explored. It was not until Peano in 1889 that a rigorous account of the natural numbers and integers was achieved. In 1874 Cantor wrote his first article on set theory, and that provided a framework for a precise definition of the natural numbers and the rational numbers. In 1899 Hilbert proposed a set of axioms for the real numbers, in a four page article. The modern set theoretic account of natural numbers, rational numbers, and real numbers was established by Cantor in his set theory, but he also discovered Cantor's paradox about sets. A great deal of effort was need to establish set theory on firm ground which was accomplished in the formalization of Zermelo Fraenkel set theory with the axiom of choice, ZFC. We do not have time to investigate set theory and this approach to the real numbers. Furthermore, that approach does not lead to a computationally meaningful notion of real numbers. That accomplishment is due to L.E.J. Brouwer whom we have mentioned before. This is a very rigorous notion that has been implemented in proof assistants as we noted above. It is formulated in type theory rather than set theory because set theory does not have a computational basis. This choice of type theory creates an opportunity for us. A very accessible historical account of the development of the notion of the real numbers is give in Morris Kline's book *Mathematical Thought from Ancient to Modern Times* [2]. Another good historical account that includes the role of Brouwer and other constructive mathematicians can be found in the book *Foundations of Set Theory* [1].

Many new computational questions arose such as how to perform the arithmetic operations on these new numbers. This led to the discovery that the algebra of the reals is also a field. There followed a great enrichment of algebra. Discovering the real numbers also led to the need to understand what it means to "compute with" them more generally, i.e. what is a function from reals to reals? We write the type of these functions as $\mathbb{R} \to \mathbb{R}$.

Once it was discovered that we could perform all of the arithmetic operations

of a field on these new "numbers," another topic in algebra arose and a whole new branch of mathematics was created called *real analysis* or just *analysis* for short. Many of the questions investigated were computational, and others were algebraic. The ties to geometry were established by Rene Descartes. In addition a whole category of investigation opened related to geometric ideas, it is called *topology*. One of its founders was L.E.J. Brouwer, and he worked to understand the fundamental character of the real numbers which he associated with the geometric idea of a line or more intuitively for most people, with the *continuum of time* which is essential to the human experience.

2 Further motivation to study constructive reals

One reason to study constructive reals is that the proof assistants are making them practical for results in *cyber-physical systems* (CPS). Another reason is that the computer science development and implementation are part of the intellectual history of constructive mathematics. Bishop's book is a landmark that computer science has "brought to life" and deepened.

The role of computer science in intellectual history is clear in the creation of the *theory of computing* (e.g. the notion of unsolvability, the creation of computational complexity measures and the famous P vs NP problem – two Cornell Turing Awards in this area). It is also clear in *artificial intelligence*, the understanding and extension of intelligence into science and mathematics. The creation of ever more expressive programming languages is another major contribution of computer science to intellectual history. These languages with their ever richer type systems are capable of directly capturing fundamental ideas from mathematics, logic, and science. We see this in our ability to support highly reliable programming of cyber-physical systems, computational geometry, homotopy type theory (HTT), and other important areas of mathematics.

3 Differences between constructive analysis over the reals, $\mathbb{R}$, and ordinary calculus over the "classical" reals

1. In calculus books, the operations on the reals are not defined by algorithms. Indeed we imagine that we can say things in calculus that don't make sense computationally. For example, in calculus we assume that given two reals, say r_1 and r_2 we know:

   - Is $r_1 = r_2$? – In fact we can't decide this in general.
   - Is $r_1 = 0$? – We can't decide this in general.
   - Is $r_1 < r_2$? – We can't decide this in general.

   Bishop needed results such as Proposition 2.16 and Corollary 2.17, page 26, to deal with these concepts constructively.

2. We will see that all functions on the reals are continuous. It is not possible to create "step functions."

Since we need to also cover a review for the prelim, in this lecture we will only look briefly at the definition of the basic arithmetic operations on the constructive reals, the definition of addition and multiplication and the concept of a positive real number. We follow the coverage of this from Lecture 11 that uses text from Bishop's book *Constructive Analysis*. We will cover these points.

- $(x + y) \ = \ fun\ n \rightarrow x(2n) + y(2n)$.
- the canonical bounds K_x, see page 19 of Bishop in Lecture 11.
- multiplication $(x \times y) \ = \ fun\, n \rightarrow x(2kn) \times y(2kn)$ where $k \ = \ max(K_x, K_y)$.
- injecting the rational numbers into the reals. Given a rational number a/b it is injected into the reals by the function $fun\ n \rightarrow a/b$.
- $(fun\ n \ \rightarrow \ x_n)$ is *positive* if and only if *for some* n, $x_n > 1/n$. See Bishop and Bridges Definition 2.7 page 21.

4 Prelim review

We won't require students to write a lot of OCaml code in the prelim. The exam will stress the concepts behind OCaml and some material covered in the problem sets. A nice question that I am giving away is this: *Why is OCaml so well suited to defining the rational numbers and the constructive reals*?

On the other hand, you should remember some of the code you wrote in PS1 and PS2, for example the is-prime function from PS1 and how to specify it from lecture Lectures 6 and 7, and the basic list processing functions from PS2. Know what the map-reduce paradigm is about in Lecture 7.

There are some concepts that are ideal for an exam such as currying or uncurrying functions, from Lectures 4 and 5. This requires knowing an important concept but not complex code. The same is true for knowing the basic functions on lists. We reviewed the actual code for many of these operations already in Lecture 5.

I am thinking of using the *array type* for a problem, so you should know what that is. Below is some code that manipulates OCaml arrays. I won't ask you to write code, but I might ask you to think about the conceptual value of a *functional data type like this* where we have direct access to each element.

There is some chance that I might ask a philosophical question like this. L.E.J. Brouwer espoused the view that "*all mathematical truths are experienced truths*". This idea seems natural in a programming course, especially one that treats basic mathematical ideas such as rational numbers, real numbers, lists, etc. We have also seen in detail how this idea applies to logic when we see the correspondence between types and propositions, especially in the case of polymorphic types. There will surely be a question about the value of polymorphic types, both its practical value in programming and its explanatory value in thinking about logical expressions with a computational meaning. There will probably be questions such as these: what is the logical meaning of the types $(\alpha \star \beta) \to \alpha$ and $\alpha \to \beta \to (\alpha \star \beta)$.

There will be a question about *dependent types* from Lecture 8 section 3.4. So it will be important to review what was discussed in Lecture 8 about the Coq treatment of logic using dependent types. I am fond of mentioning how nice it would be to have an OCaml type like this one $x : \alpha \star \beta(x)$ which consists of pairs (a, b) where a belongs to type α and b

belongs to the type $\beta(a)$. A simpler version of this idea would be a dependent type such as $x : (\alpha \star \beta)\ where\ b(x)$ for some Boolean condition $b(x)$ on the pair (a, b). For example, we could define the rational numbers as $x : (int \star int)\ where\ (snd(x) \neq 0)$.

There is a chance that I will ask something simple about showing that a type specification is not programmable as we saw with Pierce's law, but I would use something far simpler, more like $(\alpha \star \neg\alpha) \rightarrow void$. Recall that I gave you an OCaml definition of the void type, but you will only need to know that such a type exists in OCaml.

```
# [|1;2;3 |] ;;
- : int array = [|1; 2; 3|]
# let a = [|1;2;3;4|] ;;
val a : int array = [|1; 2; 3; 4|]

# a.(3) <- 5 ;;
- : unit = ()
# print_int a.(3) ;;
5- : unit = ()
# print_int a.(1) ;;
2- : unit = ()

# let value n = [|2;3;5;7;11|].(n) ;;
val value : int -> int = <fun>
# value 3 ;;
- : int = 7
# let update2 n m = [|2;3;5;7;11|].(n) <- m ;;
val update2 : int -> int -> unit = <fun>
#
# print_int [|2;3;5;7;11|].(2);;
5- : unit = ()
```

References

[1] A. A. Fraenkel, Y. Bar-Hillel, and A. Levy. *Foundations of Set Theory*, volume 67 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 2nd edition, 1984.

[2] Morris Kline. *Mathematical Thought from Ancient to Modern Times*. Oxford University Press, 1972.