CS3110 Spring 2017 Lecture 13: Comments on Prelim, More Constructive Reals

Robert Constable

	Date for	Due Date
PS4	Out on March 20	March 30
PS5	Out on April 10	April 24
PS6	Out on April 24	May 8 (day of last lecture)

1 Discussion of Prelim

The prelim exams are graded and are ready for pick up. This section of the lecture summarizes comments from the grading session and discusses points brought to our attention by grading. Recall that one of the key goals of this course is to teach important enduring ideas in computer science that are well expressed in OCaml and other functional languages, current and future.

Most students found Question 1 easy. Below is code for question 1 that shows correct answers and additional related code that might be of interest.

In Question 2, it was best to give a polymorphic type to the expression $fun\ x\ \ \rightarrow fst\ x$. That type is $(\alpha \star \beta)\ \ \rightarrow\ \ \alpha$. Question 3 was easy and most people solved it well. In Question 4 on Currying, it is possible to give a polymorphic type $(\alpha\ list \star\ int)\ \ \rightarrow\ \ \alpha$. When an expression has a polymorphic type it is always best to give that. You should know why.

For Question 5 part (c), the run time of a naive algorithm can be expressed in terms of $\sqrt{p}$ to a power. For part (d) we reference comments in the lecture and lecture notes that there is a polynomial time algorithm in the number of digits of the input.

The point about Question 6 is that access to any element of the array is constant time, whereas finding the ith element of a list depends on the position of the element in the list. The operations on arrays are repeated below in this section.

Question 7 is a topic that was covered at least three times in lecture, using examples such as $(L\alpha \mid R\beta)$. It represents a *logical disjunction*, also called the logical or as in A v B.

For Question 8 the key is to produce a real number whose nth approximation is 1/n unless after m steps of evaluation some program halts, then we produce a fixed 1/m thereafter, creating a positive real. If the program does not halt, the resulting real number is zero.

```
# (fun x -> x + x) (2*3) + 1 ;;
- : int = 13

# (fun x -> x + x) ((2*3) + 1) ;;
- : int = 14

# [|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 = ()
```

```
# type evenodd = Even of int | Odd of int ;;
type evenodd = Even of int | Odd of int
# (fun x -> match x with Even x -> x/2 | Odd x -> (x+1)/2 ) ;;
- : evenodd -> int = <fun>
# (fun x -> match x with Even x -> x/2 | Odd x -> (x+1)/2 ) (Odd 15) ;;
- : int = 8
```

2 Decimal Real Numbers

Our reference resource for studying the real numbers is Chapter 2 of the book by Bishop and Bridges, *Constructive Analysis* [1]. Chapter 2 is available on the PRL research group's web page, www.nuprl.org, and it is listed as an explicit resource on the CS3110 web page in Lecture 11, and Lecture 12 includes relevant pages from this book.

We can follow a thread in Chapter 2 that tells us how to implement and use the constructive reals in OCaml. A constructive real number is an OCaml function from integers to rationals that satisfies certain properties. The properties tell us what happens when we compute with these numbers. Even though the constructive reals satisfy many of the algebraic properties of the rational numbers, they open up a whole new branch of mathematics to a computational treatment that is simultaneously very useful and practical and yet completely precise mathematically. It is this dual character that we are exploring in this course. Last year we used these reals to do some computational geometry. They are becoming increasingly important in the study of Cyber-Physical Systems (CPS) where correctness can be a matter of life and death. As robots enter the mainstream of ordinary modern life, understanding how to build safe CPS systems will become increasingly important to everyone. The idea of using constructive reals in mathematics goes back to L.E.J. Brouwer, one of the great mathematicians of the last century [5, 4, 2] whom I have mentioned before and will discuss again in due course.

`http://nuprl.org/MathLibrary/ConstructiveAnalysis/Constructive_Analysis_Ch2.html`

2.1 Decimal numbers as constructive reals

We can use decimal real numbers such as π, $\sqrt{2}$, and Euler's number e to illustrate Bishop's definition of the real numbers. His account is not oriented toward the *decimal numbers*, but it is easy to use it to define them. Let's look at π as an example. Here are its first 20 digits.

$\pi = 3.14159265358979323 84.$

We can generate an unbounded number of digits using the Nuprl implementation of the Bishop and Bridges definition. The origin of this definition can be traced back to Brouwer and his student Heyting [3]. We can also look up a million digits on the web and get access to any of the first 200 million digits. For $\sqrt{2}$, we can examine a million digits on the web, and using Nuprl we can calculate any number of digits. These real numbers, π, $\sqrt{2}$, e are precisely defined and computable. Indeed any computable real number can be written as an unending decimal, e.g. we can keep asking for more digits. However, we need to use the underlying mathematical definition to make sense of this decimal expansion. It is a mistake to try to take the decimal representation as the primitive definition, a mistake that is subtle, even Turing made it.

In a Bishop style presentation of *pi* we would give an algorithm to generate the successive approximations, e.g. 3, 3.1, 3.14, 3.141, 3.1415, 3.14159, 3.141592, 3.1415926, ... It is evocative to think of the real number as the point on continuum defined by this unending sequence of nested intervals ((((...)))). When we use decimal numbers, we are approximating not by 1/n but by $1/10^n$. The first set of matching parentheses represent the interval 3( )4. Then inside this interval we define a smaller one, 3(3.1(...)3.2)4.

Next we have
3(3.1(3.14(...)3.15)3.2)4, refining the inner intervals into smaller ones,
3(3.1(3.14(3.141(3.1415((...))3.1416)3.142)3.15)3.2)4. Of course to make this process work, we need a precise mathematical definition of *pi*. We do not go into that topic here. There is sufficient information in calculus textbooks and on the web.

These comments on the decimals are informal intuitive ideas that Bishop and Bridges make precise, influenced by Brouwer and his student Heyting who wrote a book on this topic [3].

3 Results from Bishop and Bridges Chapter 2

Bishop and Bridges use a more general definition of the reals than what is required to define their decimal representation. The decimal representation is an instance of the general definition. On the other hand, it is not quite as easy to connect this fully general account to our experience with the decimal reals. The advantage of the Bishop and Bridges account is that we see the core ideas in a simple and general form.

They define a real number as a *regular sequence* of rational numbers $\{x_n\}$. The term "regular" fixes a rate at which the rational approximations get closer to each other. Their definition is that a sequence of rational numbers is *regular* if and only if $|x_m - x_n| \leq (1/m + 1/n)$. We can see this for *pi* which can have a sequence like that given above 3, 3.1, 3.141, 3.1415, ... We can take $x_1 = 3$, $x_2 = 3.1$, $x_3 = 3.14$, $x_4 = 3.141$, etc.

Two real numbers x_n and y_n are *equal* if and only if *for all* natural numbers n, $|x_n - y_n| \leq 2/n$.

In this lecture we will highlight some of the most useful and insightful definitions and results from Chapter 2. The most key ideas are these. The definition of *equality* (2.2). The definition of the arithmetic operations (2.4). This requires the unfamiliar notion of a *canonical bound*, K_x. This bound is used to give a definition of multiplication that produces a regular sequence. It is a technical matter that arises from the desire for a very simple definition of a real number as a regular sequence. To accommodate multiplication, Bishop and Bridges use this simple idea of a canonical bound. Otherwise the definition of the arithmetic operators is straightforward, and it is easy to prove their **Proposition** 2.3 showing that the arithmetic operations produce regular sequences. In **Proposition** 2.6 they show that the algebraic operations form a field. This is an easy result.

3.1 Key theorems about the constructive reals

We have already stressed the idea of a *positive real number* which they give in **Definition 2.7**. A real number is *positive* if and only if $x_n > 1/n$ *for some* natural number n. Also recall the above definition of when two real numbers are equal. It is a feature of constructive mathematics that when we define a type of mathematical objects we also say when two of them are equal. **The equality definitions take us from coding or programming into the realm of mathematics.**

Here are the theorems of Chapter Two of interest for this course.

Lemma 2.14 page 25: For every real number x, $|x - x_n| \leq 1/n$.

Lemma 2.15 page 25: For all real numbers x,y, if $x < y$ then we can find a rational number α strictly between them, $x < \alpha < y$.

Lemma 2.18 For all real numbers x,y, if $\neg(x > y)$, then $x \leq y$. [1].

One of the interesting theorems in Chapter 2 is the claim that the constructive real numbers are *uncountable* in the sense that there is no computable enumeration of them. This is *Theorem 2.19.* Here we simplify it by giving only the result similar to Cantor's famous theorem that the reals are uncountable. At some level this seems impossible because we can enumerate all possible OCaml programs, and list them. So the constructive reals seem to be obviously countable. What is going on? This is very much like the result from computing theory that there is no recursive enumeration of all programs that halt on all of their inputs. By a simple "diagonal argument" we know there can not be such an enumeration. Suppose the recursive enumeration of all halting programs of one input is $p_1, p_2, ..., p_n, ...$, given by the function $enum(i, n)$. Then we could define the diagonalization of this list, $enum(n, n) + 1$. This would be a computable function somewhere on the list, say $enum(u, n) = enum(n, n) + 1$. But then we get the contradiction $enum(u, u) = enum(u, u) + 1$ which is impossible.

Cantor's Theorem 2.19 (page 27): Let a_n be a computable sequence of real numbers, then we can construct a real number x that is not on the list, i.e. for all n, $x \neq a_n$.

The full version of Theorem 2.19 in the book is essentially this. The wording is changed a bit to stress that we can construct the real number x. The proof itself gives that construction exactly. This is the character of constructive mathematics, the proofs often provide implicit algorithms. In many cases we can directly "see the algorithm" from the proof. Even when we can't see it clearly, there are systematic methods of extracting the object claimed to exist from the proof itself. This is the basis of *program extraction from constructive proofs* that Cornell pioneered.

Full Theorem 2.19 Let (a_n) be a sequence of real numbers, and let x_0 and y_0 be real numbers with $x_0 < y_0$. Then we can construct a real number x such that $x_0 \leq x \leq y_0$ and $x \neq a_n$ for all positive integers n.

Some of these results will be on the final exam, so learn them now and

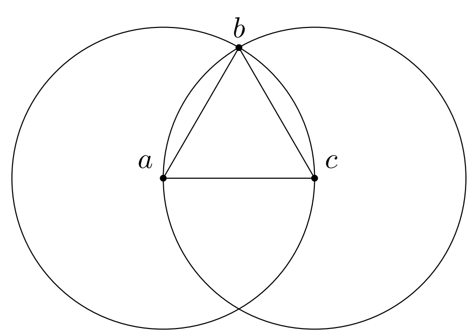
[1] $\neg(x > y)$ is the logical expression $(x > y) \rightarrow False$

review them before the exam.

4 Synthetic Geometry

The next problem set will require learning and implementing some results in *computational geometry*. It is interesting that the problems can be explained without using *real* numbers or floating point numbers, but those numbers are a way that we know how to compute answers to this kind of problem. We will also see that it is possible to implement geometric constructions directly without using the real numbers. When we think of geometry abstractly in terms of points and lines, we call the work *synthetic geometry* as opposed to analytic geometry or "real geometry" in the sense of using the real numbers. Euclidean geometry is synthetic, and it was designed around the concept of *ruler and compass constructions*.

In the meanwhile our notion of constructive computational mathematics has been enriched, and it is interesting to see how much of Euclid is constructive in the modern sense and what *constructive synthetic geometry* means now that mathematics is so much deeper than it was in 350 BCE.



References

[1] E. Bishop and D. Bridges. *Constructive Analysis*. Springer, New York, 1985.

[2] L.E.J. Brouwer. Intuitionism and formalism. *Bull Amer. Math. Soc.*, 20(2):81–96, 1913.

[3] A. Heyting. *Intuitionism, An Introduction.* North-Holland, Amsterdam, 1966.

[4] A. Heyting, editor. *L. E. J. Brouwer. Collected Works*, volume 1. North-Holland, Amsterdam, 1975. (see On the foundations of mathematics 11-98.).

[5] Mark van Atten. *On Brouwer*. Wadsworth Philosophers Series. Thompson/Wadsworth, Toronto, Canada, 2004.