CS 3110 Spring 2016
Problem Set 3
Version 3 (last modified March 17, 2016)

Overview

In the previous problem set, you implemented rational numbers and their arithmetic operations. For this assignment, you will use this implementation to build the Real numbers. We will be using the definitions from *Foundations of Constructive Analysis* by Errett Bishop namely chapter 2. While we aim to keep this writeup self-contained, it is also a good idea to use this text to supplement your understanding of real numbers. This assignment must be done individually.

Objectives

- Gain mathematical maturity and an intuitive understanding of the number system
- Write Ocaml code that makes extensive use of Modules, functors, and signatures.
- Implement the constructive Reals and use them to solve interesting geometrical problems

Recommended reading

The following supplementary materials may be helpful in completing this assignment:

- Lectures 6 7
- The CS 3110 style guide
- Real World OCaml, Chapter 4,9,10
- Constructive Analysis, Chapter 2

What to turn in

Exercises marked [code] should be placed in `ps3.ml` and `rationals.ml`, and will be graded automatically. Exercises marked [written] should be placed in `written.txt` or `written.pdf` and will be graded by hand.

Compiling Instructions

We will compile your code using the following command,

```
cs3110 compile -l nums ps3.ml
```

the -l flag tells the compiler to include the nums library which contains the Big_int module we need to use.

Warm-up

Exercise 1.

[code] Let's first extend the rational module to include some useful functions. Note that we are now using Big Ints as opposed to regular ints. The reason for this is that a 32-bit integer will surely overflow and cause our approximations to be incorrect. A list of all its functions can be found here: http://caml.inria.fr/pub/docs/manual-ocaml/libref/Big_int.html. Look over the Big_int module and get familiar with it, we will be using this module extensively for this problem set.

> **Style tip:** There is a compact syntax for a local open: M.(e) evaluates e in an environment with all of the definitions of M available, for example, we can write Big_int.(add_big_int zero_big_int zero_big_int instead of Big_int.add_big_int Big_int.zero_big_int Big_int_int.zero_big_int. Take advantage of this!

TODO: Implement the `max : t-> t-> t` and `ceil: t-> t` functions in `rationals.ml` file

The Real Number System

The first proof of the existence of irrational numbers is often attributed to Pythagoras. It is believed that he discovered them by studying pentagrams. However, It wasn't until the work of Georg Cantor in 1871 where a rigorous definition of the real numbers was first given. Calculus itself develop without a rigorous definition of real numbers available. Since then, multiple definitions of real numbers have been proposed. In this problem set, we will work with the definitions provided by Errett Bishop.
Before discussing the construction of the real numbers it is important to understand the task at hand and what the set of real numbers should contain. First, the real numbers should include all the rational numbers and have the $+$ and $*$ operators. Namely, the real number system that we construct should form a field. We say that a set along with two operations $(+, *)$ is a **field** if it satisfies the following properties,

1. Closure under addition and multiplication. Consider $a, b \in F$ where F is a field, then we know that $a + b \in F$ and $a * b \in F$. Intuitively speaking, if we are operating within a field, we should never get an element that in not included within that field
2. Associativity under addition and multiplication. Let $a, b, c \in F$ then $a + (b + c) = (a + b) + c$ and $a * (b * c) = (a * b) * c$.
3. Commutativity of addition and multiplication. Let $a, b \in F$ then $a + b = b + a$ and $a * b = b * a$
4. Existence of the additive identity. There exists the element 0 such that $\forall a, a + 0 = 0 + a = a$
5. Existence of the multiplicative identity. There exists an element 1 such that $\forall a, a * 1 = 1 * a = a$
6. Existence of additive inverse. For every element a there exists an element $-a$ such that $a + (-a) = 0$
7. Existence of the multiplicative inverse. For every element $a \neq 0$ there exists an element a^{-1} such that $a * a^{-1} = 1$
8. Distributivity, For every element a, b, c, $a * (b + c) = a * b + a * c$.

A correct construction of the Real number system must satisfy all of these axioms. In this problem set *you are not expected to prove anything about the Reals* but we expect that you have a basic understanding of their definition, and the operations that we define.

Exercise 2: Definition of Real Numbers.

[code]As you learned in class, a real number can be defined as an infinite sequence of rational numbers getting closer and closer together. The formal given to us by Bishop of a constructive

real numbers is as follows.
Definition: Let, $\{x_i\}$ denote an infinite sequence of rational numbers (eg, $\{q_1, q_2, q_3, ...\}$). We say that $\{x_i\}$ is *regular* if for all $m, n \in \mathbb{N}$,

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

We say that a real number $r \in \mathbb{R}$ is regular sequence of rational numbers $\{x_i\}$, i.e $r \equiv \{x_i\}$

When defining addition and multiplication, it is important that this definition holds. This definition of real numbers motivates our type definition for our Real module. Namely, a real number is nothing more than a function from the natural numbers to the rational numbers, i.e $r : \mathbb{N} \to \mathbb{Q}$.
TODO: [Code](**Update March 17:** We now give you this type in the realease code so you need not change it. This is because we need your . file to match our .mli enjoy your freebie!)Given this definition of real, give a definition for the `type t` in the `MakeReal` functor in `ps3.ml`.

Exercise 3: Rationals as Reals, Multiplicative and Additive Identities.

[code]Surely, the field of Reals needs to contain rational numbers as well. This is done in the simplest way; we take a constant sequence. In other words, if $\frac{p}{q} \in \mathbb{R}$, then $\frac{p}{q} \equiv \{\frac{p}{q}\}$. The proof that this sequence satisfies the definition of rationals is trivial.
TODO: Implement the `make_real_from_rational: Rat.t -> t`, `one: t`, and `zero: t` functions in the `MakeReal` functor in `ps3.ml`.

Exercise 4: Definition of Addition, and Negation.

[code]Addition and Negation are done in the simplest way.
Definition: Let $x = \{x_n\}$ and $y = \{y_n\}$ be two real numbers. Then $x + y = \{x_{2n} + y_{2n}\}$. We claim that by adding real numbers this way will make the the sequence a valid sequence. Indeed, let $z \equiv x + y$, then

$$\begin{aligned} |z_n - z_m| &= |x_{2n} + y_{2n} - x_{2m} - y_{2m}| \\ &\leq |x_{2n} - x_{2m}| + |y_{2n} - y_{2m}| \\ &\leq \frac{1}{2n} + \frac{1}{2m} + \frac{1}{2n} + \frac{1}{2m} = \frac{1}{n} + \frac{1}{m} \end{aligned}$$

Definition: Let $x \equiv \{x_n\}$ be a real number, its additive inverse is then given by $-x \equiv \{-x_n\}$. The proof of this is trivial. And we see that $x + (-x) = \{x_{2n} - x_{2n}\} = \{0\}$.
TODO: Given the above definitions, implement the `add: t-> t -> t` and `negate: t-> t` function in the `MakeReal` functor.

Exercise 5: Defining Multiplication.

[code]Defining multiplication is a little trickier. The reason is that defining multiplication the naive way, i.e $x * y = \{x_n y_n\}$ will break the regularity requirement. To define multiplication, we need an upper bound. That is for every $x \in \mathbb{R}$ we need some number K_x such that $\forall n \in \mathbb{N}$

$$|x_n| < K_x$$

Luckily this number can be found by finding the least integer which is greater than $|x_1| + 2$. We call K_x the *canonical bound* for x. Using this value, we can now define multiplication
Definition: Let $x \equiv \{x_n\}$ and $y \equiv \{y_n\}$ be real numbers with canonical bounds K_x, K_y respectively. Let $k = \max(K_x, K_y)$. Then,

$$x * y = \{x_{2kn} y_{2kn}\}$$

We claim that this definition of multiplication will satisfy the regularity of the sequence. Indeed, let x, y be real numbers and let $z = x * y$, then

$$|z_n - z_m| = |x_{2kn} y_{2kn} - x_{2km} y_{2km}| = |x_{2kn} y_{2kn} - x_{2kn} y_{2km} + x_{2kn} y_{2km} - x_{2km} y_{2km}|$$

$$\leq |x_{2kn}||y_{2kn} - y_{2km}| + |y_{2km}||x_{2kn} - x_{2km}|$$

$$\leq k(\frac{1}{2kn} + \frac{1}{2km}) + k(\frac{1}{2km} + \frac{1}{2kn}) = \frac{1}{m} + \frac{1}{n}$$

This shows that the regularity requirement is satisfied by multiplication.
TODO: Given the above definition of multiplication, implement the `multiply: t -> t -> t` function in the `MakeReal` functor

Exercise 6: Defining Inverses.

[code]Defining inverses is perhaps the trickiest operation. The reason for this is that the naive implementation $x^{-1} \equiv \{x^{-1}\}$ does not account for the fact that an element in the sequence might be zero. Luckily it a known fact that if $x \neq 0$ then it will have at most a finite number of zeroes in its sequence. In other words, there exists a point in the sequence n such that $\forall m > n$.

$$|x_m| \geq N^{-1}$$

After some point every element in the sequence is bounded from below. Our challenge is to find that N. First, consider the following definition of a positive number.
Definition: A real number $x \equiv \{x_n\}$ is said to be positive if

$$x_n > n^{-1}$$

For some n. Now we can prove that positive real numbers are bounded from below.
Therefore, for every positive real number x we can find some n such that the definition holds. We claim that if we let $N \in \mathbb{Z}$ satisfy,

$$\frac{2}{N} \leq x_n - n^{-1}$$

then $\forall m \geq N$,

$$x_m \geq N^{-1}$$

After finding N we are now able to define inverses as follows
Definition: Let x be a non-zero real number. There exists a positive integer N with $|x_m| \geq N^{-1}$ for $m \geq N$. Then we define the inverser $y = x^{-1}$,

$$y_n = (x_{N^3})^{-1} \ (n < N)$$

$$y_n = (x_{nN^2})^{-1} \ (n \geq N)$$

We assert that this construction will provide a regular sequence and thus is a well-defined real number. We recommend reading chapter 2 of Erett Bishops book for a full proof.
TODO: Given the above definition of inverse, implement the `inverse: t -> t` function the `MakeReal` functor. (Hint: start by finding n and N).

Implementing the Square root, cosine, and exponential functions

Congratulations! Up to this point, you have built from scratch a powerful real number system supporting arbitrary $\epsilon = 1/n$ precision. To fully appreciate the awesomeness of this module, in this part you will implement some useful functions on Reals, which will give you arbitrarily precise representation of the familiar $\sqrt{2}$, π and e.

Exercise 7: Square Root of Reals.

[code]Let's start with the easy one. Consider a nonnegative real $x \equiv \{x_n\}$, i.e. $\forall n > 0, x_n \geq -n^{-1}$. This constraint is just to ensure the square root of x is indeed real.
Now the intuition of taking square root of a sequence of rationals is naturally to take square root of each element individually, but we don't know how to efficiently do that for an arbitrary rational. Fortunately we do know how to find the square root of a non-negative **integer**. In fact as you might have seen in class, finding the *int-sqrt* of an integer n can be done in $O(logn)$ time. Put more formally, given $n \in \mathbb{N}$, $m = isqrt(n)$ if $m \in \mathbb{N}, m^2 \leq n$ and $(m+1)^2 > n$.
TODO: Implement the helper function `isqrt` in `sqrt`.
To use *isqrt*, we need to approximate x_n by integers. This can be done by rewriting

$$x_n = p_n/q_n \approx a_n/2n \text{ for some } a_n \in \mathbb{Z}$$

a_n can be solved by the following formula. Note that here the division is integer division.

$$a_n = (p_n * 2n)/q_n$$

Now let's define $y \equiv y_n$ by

$$y_n = isqrt(a_n * 2n)/2n$$

It's easy to verify that y_n^2 approximates x_n by only the errors incurred by integer square root. Note that here there is indeed some caveats: we have actually introduced two types of errors. The error in approximating x_n with a_n and the error in approximating the integer square root of a_n. Fortunately both these errors can be simultaneously bound and y_n here is still regular, and in fact y is the square root of x. For those who are interested in the proof, please consult Mark Bickford's "Constructive Analysis and Experimental Mathematics using the NuPrl Proof Assistant".
TODO: Implement `sqrt: Reals.t -> Reals.t`
Now you can plug in some numbers built from rationals into your sqrt functions and approximate $\sqrt{2}$ to any precision you like!

Exercise 8: Cosine Function.

[code]From our previous success with root function, an initial attempt for finding the cosine value of any given real x might be to approximate the integer cosine value for each x_n. Unfortunately unlike roots, cosine cannot be easily approximated for arbitrary integers. So our approach here is a more general and powerful one, Taylor expansion.
Recall that the taylor expansion for cosine is the following:

$$cos(x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^{2n-2}}{(2n-2)!}$$

Note here the difference from conventional expansion due to the shift of index. The biggest difficulty here is that this is an infinite series, and for computation we can only approximate this series with its partial sum, which introduces yet another error that needs to be bound. Since the mathematics here is only a means and not an end, we will just introduce a technique commonly used in real analysis for construction of sequence without proving its correctness. Assume $y = cos(x)$, we shall define $y_n \in \mathbb{Q}$ through diagonalization. Namely to approximate y_n for some n, we take the partial sum of $cos()$ up to the n^{th} term and plug in the approximation x_n of x for computation.

$$y_n = \sum_{k=1}^{n} (-1)^{k-1} \frac{x_n^{2k-2}}{(2k-2)!}$$

We claim without proof here that thus constructed sequence is in reals.
TODO: Implement `cos:Reals.t -> Reals.t`
Note: Through a similar procedure we can define the function *arctan()*, which will give us π with infinite precision!

Exercise 9: Exponential Function.

[code]The procedure for approximating the exponential function is exactly the same as for cosine, and you already have the factorial function from implementing cosine. The Taylor

expansion is given as follows:

$$e^x = \sum_{n=1}^{\infty} \frac{x^{n-1}}{(n-1)!}$$

TODO: Implement `exp:Reals.t -> Reals.t`

Logic

Exercise 10.

[written]Expand the following abbreviated logical formulas into their primitive definitions, e.g. $\sim \alpha$ to $\alpha \to Void$. Then give a program in the corresponding type and comment on how it provides evidence for the "computational truth" of the formula.

1. $\sim(\alpha \vee \beta) \Rightarrow \sim\alpha \,\&\, \sim\beta$
2. $(\alpha \Rightarrow \gamma) \,\&\, (\beta \Rightarrow \gamma) \Rightarrow (\alpha \vee \beta) \Rightarrow \gamma$
3. $(\alpha \Rightarrow \beta) \Rightarrow (\sim\beta \Rightarrow \sim\alpha)$
4. The questions below are optional, they are a bit tricky but fun!

* $\forall x : int.\alpha(x) \Rightarrow \alpha(17)$
* $\sim\alpha(17) \Rightarrow \sim\forall x : int.\alpha(x)$
* $\sim\sim(\alpha \vee \sim\alpha)$

Comments

[written] At the end of the file, please include any comments you have about the problem set, or about your implementation. This would be a good place to document any extra Karma problems that you did (see below), to list any problems with your submission that you weren't able to fix, or to give us general feedback about the problem set.

Release files

The accompanying release file `ps3.zip` contains the following files:

- `writeup.pdf` is this file.
- `release/ps3.ml`, `written.txt` and `rationals.ml` are templates for you to fill in and submit.
- `ps3.mli` contains the interface and documentation for the functions that you will implement in `ps3.ml`