Flean Redesign

Joseph McKinsey

A review of the original Flean design

In the previous version of Flean, I made some significant changes from the original design of Flocq. In particular, I do not focus on the subset of the reals, instead I use two functions from the "float" to the rationals.

I considered an interpration function $i : \text{Float} \to \mathbb{Q}$ which interprets a floating point number into the rationals. We also have a rounding function $r : \mathbb{Q} \to \text{Float}$ which produces the nearest rounded function, where 0 rounds to `-0`. Then the "behavior" of floating point numbers is formulated from rounding functions: $(x : \text{Float}) + (y : \text{Float}) = r(i(x) + i(y))$.

One floating point type can have different rounding modes, such as round up, round down, round to nearest (with zero). The actual "data" is determined by the exponent range and precision in a `FloatCfg` object which parameterizes the "valid" elements on the mantissa and exponent. The actual interpetations were all formulated as $\left(1 + \frac{m}{p}\right)2^e(-1 \text{ if } s \text{ else } 1)$. The rounding functions were implemented using the floor-log, `Int.log`.

The main theorems connected r and i. First, $r(i(x)) = x$ except for `-0`. Second, $|i(r(x)) - x| \leq \varepsilon x$ for some ε for "normal numbers", and $|i(r(x)) - x| \leq \varepsilon$ for some ε for "subnormal numbers". These target goals were produced from the partial order properties instead. We define that for $x, y : \text{Float}$, $x \leq y$ iff $i(x) \leq i(y)$, so in particular, Float's are a preorder not a partial order.

Then for rounding, we have that $x \leq y$ implies $r(x) \leq r(y)$. If we consider the positive and negative parts separately, say "FloatP" for positive, then r is a total order homomorphism with a nice left inverse. For each $x \in \mathbb{R}$, we can find an $i(f_d) \leq x \leq i(f_u)$, and $r(i(f_d)) \leq r(x) \leq r(i(f_u))$ or $f_d \leq r(x) \leq f_u$. Then we can bound the difference between $r(x)$ and these lower and upper bounds, and in fact we can choose the closest if we need to (which can help defines our original).

Many of the annoying details occurred from the specific floor lemmas, the algebra around the log-rules (which has no decision procedures), and then how to combine the different rounding parts: FloatReps, Subnorms, etc. The "sign" was attached to every part, which caused difficulties since rounding depends on the sign.

In order to combine the different FloatReps and Subnorms, we combined them in a sum type. Then we define r by checking on the boundary 2^{-127}. When we rounded up, we would turn from a subnormal to a normal number, but this would not change the interpretation. This was unusually difficult. Oddly enough, this didn't depend on the specific rounding functions of the individual parts. At 0, we have a similar discontinuity, but we keep the distinction from `-0` to `+0`.

An additional problem was that there is no way to extend r to $\mathbb{R}$. Although $\mathbb{R}$ is not computational, there are good reasons to be able to compare floats and reals, which are a bit difficult to do. In order to create the extension at the end, we need that for all $x \in \mathbb{R} - \mathbb{Q}$, there exists q, s, such that $r(q) = r(s)$. That seems somewhat hard to establish after the fact. It seems easier to make $r : \mathbb{Q} \to \text{Float}$ as a particular computable subset of $r : \mathbb{R} \to \text{Float}$.

New Definitions

I've some cursory research which gives better names and more references to this, primarily "Ordered sets: Retracts and connections" by Henry Crapo, which helped me give better names to these things.

A total order F is a **retract** of a total order X when there is

- a monotone inclusion $i : F \to X$
- a monotone rounding function $r : X \to F$
- $r(i(x)) = x$ for all $x \in F$

There are two "main" rounding functions, $r_{\text{up}}(x)$ where $r_{\text{up}}(x) = \min(\{y \mid x \leq i(y)\})$, and $r_{\text{down}}(x) = \max(\{y \mid i(y) \leq x\})$. So when F is a lattice, we can always define a retraction from an ambient space. Furthermore, this is a Galois insertion and all Galois insertions are order retractions.

We also have that $r_{\text{down}}(x) \leq r(x) \leq r_{\text{up}}(x)$ for any x and y, so rounding functions are actually quite limited.

This definition does not apply to IEEE 756 because of `+0` and `-0`. For this, we have $i(r(i(x)) = i(x)$, so they have the same interpretation. We can't make our interpretation worse by rounding. When we relax this requirement along with loosening F to a preorder, we say that F **weakly-approximates** X. This isn't quite a "connection" since $r(i(r(x))) \neq r(x)$ since double rounding can mess with 0. This weak approximation does not compose as well, so we want to delay it as long as possible.

Composition

If F' approximates F and F approximates X, then we can compose it so that $r = r_1 \circ r_2$ and $i = i_2 \circ i_1$. Then $r \circ i = \text{id}$, and for weak approximations, we can't quite reduce $i \circ r \circ i = i$.

Restriction

If F approximates A, then we can pick any subset F' of F along with a rounding function $r : F \to F'$, then we can compose and get that F' approximates X.

Gluing

The primary construction of IEEE 754 is through the composition of positive and negative floats. Positive floats are composed of subnormal and normal numbers. Normal numbers are composed of its own logarithmically scaled copies of evenly spaced numbers. So there are three ways we glue them together:

1. Given F_n which are 2^p elements spaced 2^{n-p} apart, construct a F from the union of each F_n.
2. Given some F_1 and F_2, construct the union.
3. Given some F_1 and F_2, construct the disjoint union.

In the IEEE 756 and in Flean, the FloatRep and Subnorm + FloatRep construction use the fact that the rounding functions are compatible and they "agree" on the boundary. In the union of positive and negative floats, we need them to the agree on "both" sides of the boundary too, so there is a bit of work.

Gluing Details

Let F have a cover $\{F_n\}_{n \in I}$ such that $F_i \leq F_j$ when $i < j$. We'll have retractions $r_n : X \to F_n$, and a monotone selection function $s : X \to I$. Assuming the inclusions are compatible and lift to $i : F \to X$ and $f \in F_{s(i(f))}$ for all $f \in F$, then $r(x) \coloneqq r_{s(x)}(x)$ is a retraction.

For Lean, we will be dealing with F_n which have monotone inclusions into F instead, which slightly complicates the proof. Luckily $(f : F) \to F_{s(i(f))} \xrightarrow{g_n} F$ for provides an explicit surjectivity requirement, cutting down on the assumptions still.

For gluing together F_1 and F_2. We need
- a monotone function $s : X \to \{1, 2\}$,
- retractions $X \to F_1$ and $X \to F_2$

  - $F_1 \leq F_2$
- $(f : F) \xrightarrow{\text{lift}} F_{s(i(f))} \xrightarrow{g_n} F = \text{id}$
- i_{F_1} and i_{F_2} are compatible with a i_F

Then

$$r(x) := \begin{cases} r_1(x) \text{ for } s(x) = 1 \\ r_2(x) \text{ for } s(x) = 2\text{'} \end{cases}$$

which can usually massaged to when $x \leq$ boundary and $x >$ boundary instead.

Slight modification for duplicates

In some instances, F is not a total order because of duplicates: we have both `-0` and `+0`. More troublesome is that $f \notin F_{(s(i(f))}$ since s sends `-0` to F_+ instead of F_-. Instead, we have that for all $f \in F$, there exists an f' such that $i(f) = i(f')$ and $f' \in F_{s(i(f'))}$. Then we get that $r(i(f)) = r(i(f')) = f'$ and $i(r(i(f)) = i(f)$.

How to handle infinity

Adding $\pm\infty$ can be done to any order, but it would also break our retraction property. It's unclear where to add it. The most obvious place to me is to make ∞ the maximum element when $2^{p+1+\text{ emax}}$, effectively relabeling it.

How to handle NaN

I suggest that NaN is the equivalent of isNone, so we add NaN at the very end, specificially to indicate error conditions in operations and nothing else. We could interpret as 0 if we need to.

How to prove error rates

For many of our applications, we need that $|i(r(x)) - x| \leq \varepsilon(x)$. In Flean, I did this in two ways. In the first way, I bound $r_{\text{down}} \leq r \leq r_{\text{up}}$, and then I could use the fact that $r_{\text{up}} - r_{\text{down}}$ is small. I also had some convenient lemmas like $r(x) = r_{\text{up}(x)}$ or $r(x) = r_{\text{down}(x)}$.

Given a bound on rounding up and rounding down, say on $A_1, A_2 \subseteq X$, then I'd expect our gluing to satisfy the same bounds in A_i and when $s(x) = i$.

Similarly, I'd expect the same thing for $|i(r(x)) - x| \leq \varepsilon(x)$.

Other Operations on Total Order Retractions

We've covered composition, restriction, and gluing. The other common operation is scaling. In our case of linearly ordered fields, we can scale down all our rounding operations.

What about 2D?

As far as I know, people rarely care about "correct" 2D rounding.

Particular Design Choices

For Flean 1, I wanted to be able to parameterize the rounding modes, but I often ran into a problem: should they be instances of a typeclass. Similarly, I would have rounding functions be an instance of "ValidRounder" for the IntRounding. This worked out pretty well, so I'm wondering if having a more generic ValidRounder is the appropriate way to do this.

Similarly, I did not custom coercion and instead used an explicit `coe_q`. In retrospect, I feel this was a mistake. I will likely try to use coercion now as much as possible.

- Should global rounding mode be a typeclass?

Probably at the highest level, yes.

- How do I parameterize different rounding operations through the code? Do you parameterize based on a function or a setting?

  It should behave like a setting, which becomes a pair of functions for the left and right side. For most implementations, it should be passed explicity.
- Should I use explicit inclusion functions to the linear order, or should it be a coercion? If I make it a coercion, do I run into type class diamonds?

  If I don't use an explicit coercion, it would be nice to put the retraction in a typeclass somehow, maybe?
- Should it work for more target spaces than $\mathbb{Q}$? Probably. What functions and properties do I need?
  - $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$, $\lfloor \log_2(\cdot) \rfloor$ (monotone), $\cdot 2^p$ for any p. Order embedding of the integers.

  Obviously, this includes the dyadic rationals, rationals, and the reals. There will likely need to be a lot of properties involved here. I'm tempted to try theorems with the reals, see what lemmas are needed, and then add those to the interface.

Rounding options

We put the rounders in a separate typeclass.

```
class ValidRounder {X : Type*} {F : Type*} [LinearOrder X] [LinearOrder F]
    (r : X -> F) (i : F -> X) : Prop where
  r_monotone : Monotone r
  coe_monotone : StrictlyMono i
  left_inverse : Function.LeftInverse r i
```

Attempt 4 (or 5) for Gluing

We define retractions between X and F *on* sets $X' \subseteq X$ and $F' \subseteq F$ as functions $r : X \to F$ and $i : F \to X$ where

- r is monotone on X'
- i is monotone on F'
- i maps F' to X'
- r maps X' to F'
- $r(i(f)) = f$ for $f \in F'$.

Now if we have retractions $r_i : X_i \to F_i$ where $\cup_{i \in I} X_i = X$ and $\cup_{i \in I} F_i = F$, then we can reframe our previous gluing lemma.

- $s : X \to I$ is a monotone selection function on $\cup_{i \in I} X_i$
- $i \leq j$ implies $F_i \leq F_j$
- i_j are compatible with each other (say are all equal to i)
- $f \in F_{s(i(f))}$ for $f \in \cup_{i \in I} F_i$
- $x \in X_{s(x)}$ for $x \in \cup_{i \in I} X_i$.

When we get this, we can see that $r_{s(x)}(x)$ is a retraction on $s^{-1}(\{i\})$ and F_i for each i. This lets us split up the "construction" from "retraction on" and the full "retraction".

Restricted API

Now that we have retractions "on" a set, we have to create an API around it. For our purposes, we can implement "restriction" in the same way as the `PartialEquiv` class, except unbundled. This mirrors `ContinuousOn.comp` instead in the types.

- `ValidRounder.toPartialRounder`: associating a partial rounding to a valid rounder on `univ`
- `PartialRounder.restrict`: the restriction on the source

- `ValidRounder.trans`: the composition of two partial equivalences
- `PartialEquiv.ofSet`: the identity on a set s
- `EqOnSource`: equivalence relation describing the "right" notion of equality

Gluing Attempt ???

We will break up our gluing into pieces. First, we construct a "glued" copy $r : X \to F$ from pieces X_i to specific types F_i. We prove that they are valid roundings for each piece. Then we prove that $r \equiv r_i$ on each X_i, so that r is a valid rounding on each X_i. Finally, we combine the valid roundings to $\bigcup_i X_i$.

We say that r is a **valid rounding on** $X' \subset X$ when r is monotone on X', i is monotone on $r(X')$, $r(i(f)) = f$ for $x \in r(X')$, and $i(r(X')) \in X'$. This is equivalent to requiring that $r(X') \subset F'$ and $i(F') \subset X'$ as the left inverse property implies $F' \subset r(X')$.

If $r : X \to F$ is a valid rounding on each X_i, then there are some conditions required for it to a rounding on $\bigcup_i X_i$.

1. **Montonicity**: For $i < j$, $X_i \leq X_j$ in the sense that $\forall x \in X_i, y \in X_j, x \leq y$.

   This gaurantees that monotonicity on each X_i for r is compatible. Similarly, we can conclude $r(X_i) \leq r\big(X_j\big)$, which gives monotonicity for i. This theorem only applies when X and F are partial orders for r and i respectively.

2. **Closure**: For $x \in \bigcup_i X_i$, $x \in X_i$, so $i(r(x)) \in X_i \subset \bigcup_i X_i$, thus the closure property is satisfied.

3. **Left Inverse**: For $f \in \bigcup_i r(X_i)$, $f \in r(X_i)$ and since we have a valid rounding on X_i, this implies $r(i(f)) = f$.

However, getting to the point that we have a valid rounding on each of those subsets can be more challenging. In the floating point normal numbers, we have that $r : X \to \text{NormalNum} = F$ can be broken down into slices $\left[2^{e+p}, 2^{e+p+1}\right)$ where $r(x) = \text{norm}(r_e(x), e)$ where norm does any carry operations needed.

Since we we want to have something more like an equality of sorts (and its easy to prove injectivity about i), then we want to use $i(r(x)) = i(\text{norm}(r_e(x), e)) = i_e(r_e(x))$. Now, proving monotonicity on i was already easy, but proving r by using the injectivity and monotonicity of i is much easier. Closure on X_i requires proving that $i(r(X_i) \subset X_i$, which is again easy enough.

Finally, we prove $r(i(f)) = f$ for $f \in r(X_i)$ requires first identifying a specific $f' \in r_i(X_i)$ such that $i(f) = i_i(f')$, then using injectivity of i, $r(i(f)) = f$ iff $i(r(i(f))) = i(f)$, and $i(r(i(f)) = i(r(i_i(f'))) = i_i(r_i(i_i(f')) = i_i(f') = i(f)$.

Note a few things: we first get a weaker rounding $i(r(i(f)) = i(f)$ and then get $r(i(f))$, but we also require strict monotonicity of i to get monotonicity of r on each X_i, which implies $r(i(f)) = f$ anyways. So there is no particular value with first constructing the weaker rounding.

Finally, strict monotonicty of i is separate from proving i is injective (unless F is a partial order), so we still need to separately prove i is injective and mono. F will be a order where $i(f_1) \leq i(f_2) \leftrightarrow f_1 \leq f_2$ (strict monoticity), and then injectivity of i proves that this is a proper partial order.

For floats specifically

Separation: We need to prove that $\left[2^e, 2^{e+1}\right] \leq \left[2^f, 2^{f+1}\right]$ when $e < f$. note $e + 1 \leq f$. If $x \in \left[2^e, 2^{e+1}\right]$ and $y \in \left[2^f, 2^{f+1}\right]$, then $x \leq 2^{e+1} \leq 2^f \leq y$.

$\boldsymbol{i(r(x)) = i_e(r_e(x))}$: Should be fairly trivial rewriting while noting $\lfloor \log_2(x) \rfloor = e$ iff $x \in \left[2^e, 2^{e+1}\right)$ and then handling the $x = 2^e$ case separately to get the endpoints.

Picking f: We need that for each (m, e), we can pick an m with the same interpretation under exponent e. This is exactly m.

Injectivity: In the case of our floating point numbers $m2^e$ where $2^p \leq m < 2^{p+1}$, we have that $x_1 = m_1 2^{e_1} = m_2 2^{e_2} = x_2$ implies $2^{e_i+p} \leq x_i < 2^{e_i+p+1}$. Since $x \in [2^{e_i+p}, 2^{e_i+p+1})$ iff $\lfloor \log_2(x) \rfloor = e_i + p$ for each i, $e_1 + p = e_2 + p$, then $e_1 = e_2$ and $m_1 = m_2$.

Remember that all of this is so that we can establish that F is a partial order, and to prove that $r(i(f)) = f$ on each X_i. Only then can we use our gluing.

Is it worth creating the machinery?

I'm not sure that it's worth trying to build the general machinery given how much of the effort is specific to normal numbers. We can handle the sign case via a transformation of the given r and i. In my previous attempts, I tried to find specific information about the selection function s, but managing all the different F_i makes that very frustrating.

We will be doing something similar for the binary case, which will already be pretty different in the actual proof mechanism, but the general steps should be the same.