MakeNoise<VectorDomain<AtomDomain<T>>, LpDistance<P, QI>, MO> for FloatExpFamily<P>

Michael Shoemate

This proof resides in **"contrib"** because it has not completed the vetting process.

Proves soundness of the implementation of `MakeNoise` over vectors for `FloatExpFamily` in `mod.rs` at commit f5bb719 (outdated[1]).

This mechanism samples from the `FloatExpFamily` distribution, where the tails saturate to the positive and negative infinity floats in the native float type. This is done by rounding floats to a fractional grid, adding noise from from the `ZExpFamily` distribution (supported on all integers) to the numerator of the fraction (with noise scale multiplied by the denominator), and then converting back to the nearest float, or saturating to positive or negative infinity.

1 Hoare Triple

Precondition

Compiler-Verified

- Generic `T` implements trait `Float`
- Const-generic `P` is of type `usize`
- Generic `QI` implements trait `Number`
- Generic `MO` implements trait `Measure`
- Type `i32` implements trait `ExactIntCast«T as FloatBits>::Bits>`, This requirement means that the raw bits of `T` can be exactly cast to an `i32`.
- Type `RBig` implements traits `TryFrom<T>` and `TryFrom<QI>`. This is for fallible exact casting from input sensitivity to a rational in the privacy map.
- Type `ZExpFamily<P>` implements traits `NoisePrivacyMap<LpDistance<P, RBig>, MO>` This bound requires that it must be possible to construct a privacy map for this combination of noise distribution, distance type and privacy measure.

User-Verified

None

[1] See new changes with `git diff f5bb719..9dfaa1b rust/src/measurements/noise/nature/float/mod.rs`

Pseudocode

```
class FloatExpFamily:
    def make_noise(self, input_space) -> Measurement[AtomDomain[T], T, AbsoluteDistance[QI],
     MO]:
        scale, k = self.scale, self.k
        distribution = ZExpFamily(scale=integerize_scale(scale, k)) #

        t_int = make_float_to_bigint(input_space)
        m_noise = distribution.make_noise(t_int.output_space())
        return t_int >> m_noise >> then_deintegerize_vec(k)
```

Postcondition

Theorem 1.1. For every setting of the input parameters (`self, input_space, MO, T, P, QI`) to `make_noise` such that the given preconditions hold, `make_noise` raises an error (at compile time or run time) or returns a valid measurement. A valid measurement has the following properties:

1. (Data-independent runtime errors). For every pair of members x and x' in `input_domain`, `invoke`(x) and `invoke`(x') either both return the same error or neither return an error.

2. (Privacy guarantee). For every pair of members x and x' in `input_domain` and for every pair (`d_in`, `d_out`), where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for
`output_measure`, if x, x' are `d_in`-close under `input_metric`, `privacy_map`(`d_in`) does not raise an error, and `privacy_map`(`d_in`) $=$ `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_measure`.

Proof. Line 4 constructs a new random variable following a distribution equivalent to `FloatExpFamily`, but without tails that saturate to infinity, and with gaps between adjacent point on the grid adjusted to one.

Neither constructor `make_float_to_bigint` nor `MakeNoise.make_noise` have manual preconditions, and the postconditions guarantee a valid transformation and valid measurement, respectively. `then_deintegerize_vec` also does not have preconditions, and its postcondition guarantees that it returns a valid postprocessor.

The chain of a valid transformation, valid measurement and valid postprocessor is a valid measurement. □