MakeNoise<VectorDomain<AtomDomain<T>>, LpDistance<P, QI>, MO> for IntExpFamily<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Proves soundness of the implementation of `MakeNoise` over vectors for `IntExpFamily` in `mod.rs` at commit f5bb719 (outdated[1]).

This mechanism samples from the `IntExpFamily` distribution, where the tails are clipped to the smallest and largest representable integers in the native integer type. This is done by first sampling from the `ZExpFamily`, the equivalent distribution supported on all integers, and then clipping the tails by clamping. The clamping is done by saturating the cast of the sampled value to the native integer type.

1 Hoare Triple

Precondition

Compiler-Verified

- Generic `T` implements trait `Integer` and `SaturatingCast<IBig>` The saturating cast is for infallible postprocessing of big ints back to type `T`.
- Const-generic `P` is of type `usize`
- Generic `QI` implements trait `Integer`
- Generic `MO` implements trait `Measure`
- Type `IBig` implements trait `From<T>`. This infallible exact cast is for converting integers to big ints in the preprocessing transformation.
- Type `RBig` implements trait `TryFrom<QI>`. This is for fallible casting from input sensitivity of type `QI` to a rational in the privacy map.
- Type `ZExpFamily<P>` implements trait `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..0b2097a6 rust/src/measurements/noise/nature/integer/mod.rs`

Pseudocode

```
class IntExpFamily:
    def make_noise(
        self, input_space: tuple[VectorDomain[AtomDomain[T]], LpDistance[P, QI]]
    ) -> Measurement[VectorDomain[AtomDomain[T]], T, LpDistance[P, QI], MO]:
        distribution = ZExpFamily(
            scale=integerize_scale(self.scale, 0)
        )  #

        t_int = make_int_to_bigint(input_space)
        m_noise = distribution.make_noise(t_int.output_space())
        return t_int >> m_noise >> then_saturating_cast()
```

Postcondition

Theorem 1.1.

Theorem 1.2. 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 exception (at compile time or run time) or returns a valid measurement. A valid measurement has the following property:

1. (Privacy guarantee). For every pair of elements x, 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 exception, and `privacy_map`(`d_in`) $\leq$ `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_measure`.

Proof. Line 7 constructs a new random variable following a distribution equivalent to `IntExpFamily`, but without clipped tails.

Neither constructor `make_int_to_bigint` nor `MakeNoise.make_noise` have manual preconditions, and the postconditions guarantee a valid transformation and valid measurement, respectively. `then_saturating_cast` 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. □