MakeNoise<AtomDomain<T>, AbsoluteDistance<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Proves soundness of the implementation of `MakeNoise` over scalars for `IntExpFamily` in `mod.rs` at commit f5bb719 (outdated[1]).

The intuition of this implementation is that a vector-valued mechanism can be used to release a scalar-valued input, by transforming the input into a singleton vector, applying the vector mechanism, and then unpacking the resulting singleton vector.

This matches the code and proof for the float case, `MakeNoise<AtomDomain<T>, AbsoluteDistance<QI>, MO> for FloatExpFamily<P>`, except for elementary data 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..286ce37c rust/src/measurements/noise/nature/integer/mod.rs`

Pseudocode

```python
class IntExpFamily:
    def make_noise(
        self, input_space: tuple[AtomDomain[T], AbsoluteDistance[QI]]
    ) -> Measurement[AtomDomain[T], T, AbsoluteDistance[QI], MO]:
        t_vec = make_vec(input_space)  #
        m_noise = self.make_noise(t_vec.output_space())  #

        return t_vec >> m_noise >> then_index_or_default(0)  #
```

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 exception (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 elements x, x' in `input_domain`, `function`(x) returns an error if and only if `function`(x') returns an error.

2. (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. Neither constructor `make_vec` nor `MakeNoise.make_noise` have manual preconditions, and the postconditions guarantee a valid transformation and valid measurement, respectively. `then_index_or_default` 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. □