fn make_laplace

Michael Shoemate

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

Proves soundness of the implementation of `make_laplace` in `mod.rs` at commit f5bb719 (outdated[1]).
Perturbative noise mechanisms may be parameterized along many different axes:

- domain: scalar or vector
- domain dtype: i8, i16, i32, i64, u8, u16, u32, u64, f32, f64, UBig, IBig, RBig
- metric: absolute distance, l1 distance, modular distance
- metric dtype: i8, i16, i32, i64, u8, u16, u32, u64, f32, f64, UBig, IBig, RBig
- measure: max divergence, zero concentrated divergence
- distribution: laplace, gaussian

All parameterizations reduce to a single core mechanism that perturbs a vector of signed big integers with noise sampled from the appropriate discrete distribution.

The implementation of this function constructs a random variable denoting the noise distribution to add, and then dispatches to the `MakeNoise<DI, MI, MO>` trait which constructs the core mechanism and wraps it in pre-processing transformations and post-processors to match the desired parameterization.

1 Hoare Triple

Precondition

Compiler-Verified

- generic `DI` implements trait `Domain`
- generic `MI` implements trait `Metric`
- generic `MO` implements trait `Measure`
- type `DiscreteLaplace` implements trait `MakeNoise<DI, MI, MO>` This trait bound constrains the choice of input domain, input metric and output measure to those that can form valid measurements.
- type `(DI, MI)` implements trait `MetricSpace`

User-Verified

None

[1] See new changes with `git diff f5bb719..a80fce7 rust/src/measurements/noise/distribution/laplace/mod.rs`

Pseudocode

```python
def make_laplace(
    input_domain: DI,
    input_metric: MI,
    scale: f64,
    k: Option[i32],
) -> Measurement[DI, DI_Carrier, MI, MO]:
    return DiscreteLaplace(scale, k).make_noise((input_domain, input_metric))
```

Postcondition

Theorem 1.1. For every setting of the input parameters (`input_domain, input_metric, scale, k, DI, MI, MO`) to `make_laplace` such that the given preconditions hold, `make_laplace` 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. We first construct a random variable `DiscreteLaplace` representing the desired noise distribution. Since `MakeNoise.make_noise` has no preconditions, the postcondition follows, which matches the postcondition for this function. □