fn make_odomter_to_filter

Michael Shoemate

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

Proves soundness of `fn make_odomter_to_filter`.

1 Hoare Triple

Precondition

Compiler-verified

- Argument `odometer` of type `Odometer<DI, MI, MO, Q, A>`.
- Argument `d_in` of type `MI_Distance`, the associated distance type of `MI`.
- Argument `d_out` of type `MO_Distance`, the associated distance type of `MO`.
- Generic `DI` implements `Domain`.
- Generic `MI` implements `Metric`.
- Generic `MO` implements `Measure`.
- `MI_Distance` implements `ProductOrd`.
- `MO_Distance` implements `ProductOrd`.
- `(DI, MI)` implements `MetricSpace`.

User-verified

Pseudocode

```python
def make_odometer_to_filter(
    odometer: Odometer[DI, MI, MO, Q, A],
    d_in: MI_Distance,
    d_out: MO_Distance,
) -> Measurement[DI, OdometerQueryable[MI, MO, Q, A], MI, MO]:
    function = odometer.function

    def function(
        arg: DI_Carrier, query_wrapper: Wrapper | None  #
    ) -> OdometerQueryable[MI, MO, Q, A]:
        continuation_rule = new_continuation_rule(d_in, d_out, MI, MO) #

        wrapper = compose_wrappers(continuation_rule, query_wrapper)  #
        return function.eval_wrap(arg, wrapper)  #

    def privacy_map(d_in_p: MI_Distance) -> MO_Distance:
```

```
17         if d_in_p.total_gt(d_in):
18             raise "input distance must not be greater than d_in"
19 
20         return d_out
21 
22     return Measurement.new(
23         odometer.input_domain,
24         Function.new_interactive(function),
25         odometer.input_metric,
26         odometer.output_measure,
27         PrivacyMap.new_fallible(privacy_map),
28     )
```

Postcondition

For every setting of the input parameters `(odometer, d_in, d_out, DI, MI, MO, Q, A)` to `make_odometer_to_filter` such that the given preconditions hold, `make_odometer_to_filter` raises an exception (at compile time or run time) or returns a valid odometer. A valid odometer has the following properties:

1. (Data-independent exceptions). For every pair of elements x, x' in `input_domain`, `function`(x) and `function`(x') either both raise an exception, or neither raise an exception.
2. (Wrapping guarantee). Interactive measurement queryables spawned while evaluating external queries are wrapped by the wrapper function accompanying the external query.
3. (Valid odometer queryable). For every element x in `input_domain`, where `function`(x) does not raise an exception, `function`(x) returns a valid odometer queryable.

Proof. Data-independent exceptions `Function.eval_wrap` on line 14 has data-independent exceptions, because the function is from `odometer`, which is a valid odometer. Since this is the only location where an exception can be raised, the data-independent exceptions guarantee holds. □

Proof. Wrapping Guarantee The query wrapper is labeled `query_wrapper` on line 9. `wrapper` on line 13 includes `query_wrapper` by the postcondition on `compose_wrappers`.

`Function.eval_wrap` on line **??** guarantees to wrap all spawned IM queryables with the provided wrapper, satisfying the wrapping guarantee. Since this is the only location where interactive mechanism queryables can be spawned, the wrapping guarantee holds. □

Proof. (Privacy Guarantee). By the postcondition of **??**fm Under the assumption that the input data is a member of the input domain, the precondition of `make_odometer_to_filter` is met, so by its postcondition the return value is a valid odometer queryable. □