`fn new_continuation_rule`

Michael Shoemate

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

Proves soundness of `fn new_continuation_rule`.

1 Hoare Triple

Precondition

Compiler-verified

Types consistent with pseudocode.

User-verified

None

Pseudocode

```
def new_continuation_rule(d_in: MI_Distance, d_out: MO_Distance) -> Wrapper:

    def wrapper(queryable: Queryable) -> Queryable:
        state = queryable
        del queryable  # (moved by rust ownership)

        def transition(query: Query[Any]) -> Answer[Any]:  #

            if state is None:
                raise "filter is exhausted"
            else:
                queryable = state

            answer = queryable.eval_query(query)

            match queryable.eval_poly(OdometerQuery.PrivacyLoss(d_in.clone())):  #
                case OdometerAnswer.PrivacyLoss(pending_d_out):
                    pass
                case _:
                    raise "expected privacy loss"

            if pending_d_out.total_gt(d_out):  #
                state.take()  #
                raise "filter is now exhausted"

            return answer   #

        return Queryable.new_raw(transition)

    return Wrapper.new(wrapper)  #
```

Postcondition

Theorem 1.1. Returns a function that wraps a queryable. The wrapped queryable refuses to release any query that would cause the privacy loss to exceed `d_out`.

Proof. Line 30 returns a function that wraps a queryable.

The new queryable, whose transition function is defined on line 7, runs the routine on line 16 for every query that changes the privacy loss. This routine queries the original queryable for what the new privacy loss is, after executing the query. If the new privacy loss exceeds `d_out` on line 22, the query is rejected, and the inner queryable is freed from memory on line 23.

Otherwise, the answer is returned on line 26. □