`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

- Argument `d_out` of type `U`.

User-verified

None

Pseudocode

```
def new_continuation_rule(d_out: U) -> Wrapper:

    def wrapper(queryable: Queryable) -> Queryable:

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

            if isinstance(query, Query.External): #
                pending_loss: PendingLoss[U] = queryable.eval_internal(query)
                if isinstance(pending_loss, PendingLoss.New):
                    pending_d_out = pending_loss[0]
                    if pending_d_out.total_gt(d_out): #
                        raise f"insufficient privacy budget: {pending_d_out} > {d_out}"

            return queryable.eval_query(query) #

        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 18 returns a function that wraps a queryable.

The new queryable, whose transition function is defined on line 5, runs the routine on line 7 for every query that could change the privacy loss. This routine queries the original queryable for what the pending privacy loss would be, after executing the query. If the new privacy loss exceeds `d_out` on line 11, the query is rejected.

Otherwise, the query is accepted, and the query is passed through to the original queryable on line 14. □