fn make_fully_adaptive_composition_queryable

Michael Shoemate

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

Proves soundness of `fn make_fully_adaptive_composition_queryable`.

1 Hoare Triple

Precondition

Compiler-verified

Types matching those in the pseudocode.

- Generic `DI` implements `Domain`.
- Generic `MI` implements `Metric`.
- Generic `MO` implements `CompositionMeasure`.
- `(DI, MI)` implements `MetricSpace`.

User-verified

`data` is a member of `input_domain`.

Pseudocode

```
def new_fully_adaptive_composition_queryable(
    input_domain: DI,
    input_metric: MI,
    output_measure: MO,
    data: DI_Carrier,
) -> OdometerQueryable[Measurement[DI, TO, MI, MO], TO, MO_Distance]:

    require_sequentiality = matches(
        output_measure.composability(Adaptivity.FullyAdaptive)?,
        Composability.Sequential
    )

    privacy_maps = []  # Vec<MO_Distance>

    def transition(  #
        self_: OdometerQueryable[Measurement[DI, TO, MI, MO], TO, MO_Distance],
        query: Query[OdometerQuery[Measurement[DI, TO, MI, MO]]],
    ):
        # this queryable and wrapped children communicate via an AskPermission query
        # defined here, where no-one else can access the type
        @dataclass
```

```
        class AskPermission:
            id: usize

        match query:
            # evaluate external invoke query
            case Query.External(
                OdometerQuery.Invoke(meas)
            ):  #
                assert_components_match(  #
                    DomainMismatch, input_domain, meas.input_domain
                )

                assert_components_match(  #
                    MetricMismatch, input_metric, meas.input_metric
                )

                assert_components_match(  #
                    MeasureMismatch, output_measure, meas.output_measure
                )

                let enforce_sequentiality = Rc::new(RefCell::new(false));

                if require_sequentiality:
                    # when the output measure doesn't allow concurrent composition,
                    # wrap any interactive queryables spawned.
                    # This way, when the child gets a query it sends an AskPermission query
                    # to this parent queryable, giving this sequential odometer queryable
                    # a chance to deny the child permission to execute
                    child_id = privacy_maps.len()

                    def callback():
                        if enforce_sequentiality.borrow():
                            self_.eval_internal(AskPermission(child_id))
                        else:
                            return ()

                    seq_wrapper = Wrapper.new_recursive_pre_hook(  #
                        callback
                    )
                else:
                    seq_wrapper = None

                answer = meas.invoke_wrap(data, seq_wrapper)  #

                enforce_sequentiality.borrow_mut() = True

                # We've now increased our privacy spend.
                # This is our only state modification
                privacy_maps.push(meas.privacy_map)  #

                return Answer.External(OdometerAnswer.Invoke(answer))

            # evaluate external privacy loss query
            case Query.External(OdometerQuery.PrivacyLoss()):
                d_mids = [m.eval(d_in) for m in privacy_maps]
                d_out = output_measure.compose(d_mids)
                return Answer.External(OdometerAnswer.Map(d_out))

            case Query.Internal(query):
                # Check if the query is from a child queryable
                #     who is asking for permission to execute
                if isinstance(query, AskPermission):  #
                    # deny permission if the sequential odometer has moved on
                    if query.id + 1 != privacy_maps.len():
```

```
86                          raise ValueError("sequential odometer has received a new query")
87
88                      # otherwise, return Ok to approve the change
89                      return Answer.internal(())
90
91                  raise ValueError("query not recognized")
92
93      return Queryable.new(transition)  #
```

Postconditions

Theorem 1.1. For every setting of the input parameters `(input_domain, input_metric, output_measure, data, DI, TO, MI, MO)` to `new_fully_adaptive_composition_queryable` such that the given preconditions hold, `new_fully_adaptive_composition_queryable` raises a data-independent error or returns a valid odometer queryable (`queryable`). A valid odometer queryable is an `OdometerQueryable` with the following properties:

1. (Data-independent errors). For every pair of members x and x' in `input_domain`, and every query q, $\texttt{queryable.invoke}_x(q)$ and $\texttt{queryable.invoke}_{x'}(q)$ either both return the same error or neither return an error.

2. (Privacy Guarantee). For every pair of members x and x' in `input_domain`, every adversary $\mathcal{A}$, and for every pair $(\texttt{d\_in}, \texttt{d\_out})$, where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if x and x' are `d_in`-close under `input_metric`, `queryable.eval(privacy_loss)` does not return an error, and $\texttt{queryable.eval(privacy\_loss)} = \texttt{d\_out}$, then $\texttt{View}(\mathcal{A} \leftrightarrow \texttt{queryable}_x), \texttt{View}(\mathcal{A} \leftrightarrow \texttt{queryable}_{x'})$ are `d_out`-close under `output_measure`, where $\texttt{queryable}_x$ denotes all messages received by $\mathcal{A}$ under x.

3. (Pending Privacy Guarantee). For every pair of members x and x' in `input_domain`, every adversary $\mathcal{A}$, every `OdometerQuery` q, and for every pair $(\texttt{d\_in}, \texttt{d\_out})$, where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if x and x' are `d_in`-close under `input_metric`, $\texttt{queryable.eval\_internal}(q)$ does not return an error, and $\texttt{queryable.eval\_internal}(q) = \texttt{d\_out}$, then the privacy loss remains the same if $\texttt{d\_out} = \texttt{PendingLoss.Same}$, otherwise $\texttt{View}(\mathcal{A} \leftrightarrow \texttt{queryable}'_x), \texttt{View}(\mathcal{A} \leftrightarrow \texttt{queryable}'_{x'})$ are `d_out`-close under `output_measure`, where $\texttt{queryable}'_x$ denotes all messages received by $\mathcal{A}$ under x, as well as $\texttt{queryable.eval}(q)$.

Proof of data-independent errors. The only interaction with the data is on line 64. By the postcondition of a valid measurement, errors are data-independent. □

Proof of privacy guarantee. The transition function follows the $\mathcal{G}$-OdomCon(IM) procedure described in Algorithm 6 of [HST+23], with some adjustments that do not materially affect the privacy analysis.

- When an odometer invokes $\mathcal{M}_{k+1}$, the OpenDP Library implementation eagerly accounts for all future privacy loss of the child mechanism immediately (on line 70), even if the analyst never interacts with the spawned interactive mechanism.

- The OpenDP Library represents interactive mechanisms via queryables, where the state s is hidden from the adversary. When invoking $\mathcal{M}_k$, no initial state λ is passed, and the response is a queryable with hidden state. By the definition of a valid interactive measurement, the view an adversary gains from this queryable has privacy loss d_k for some choice of `d_in`.

- The queryable for child k can be viewed as $\mathcal{M}_k$ together with s_k. Since the queryable itself satisfies d_k-DP, the queryable can be returned to the user to provide query access, child queryables ($\mathcal{M}_k$ and s_k) can be removed from the odometer state, and the handling of child update queries $m = (j, q)$ can be removed while preserving equivalency with Algorithm 6. This equivalently reduces the odometer state to $(s, [d_1, ...d_k])$.

- In addition to storing the dataset x, the state also contains the input domain, input metric and privacy measure. Each incoming mechanism must share these same supporting elements. This is an implicit requirement of the paper made explicit in the implementation.

We now discuss how the pseudocode implements this equivalent algorithm. The input domain, input metric, privacy measure and data are moved into the transition function, as well as a mutable list of privacy losses from line 13, to form the state.

First consider the case where the privacy measure satisfies concurrent composition. Lines 34 and 38 are necessary to guarantee that the constructed odometer queryable gives valid privacy loss guarantees with respect to `input_metric` and `output_measure`. On line 64, the user-verified preconditions for invoking the measurement are met, by the precondition that the `data` is a member of `input_domain` and the check that the measurement's input domain matches `input_domain` on line 30. Therefore by the definition of a valid measurement, `answer` satisfies `measurement.privacy_map(d_in)`-DP, with respect to `output_measure`, for some choice of `d_in`.

Now consider the case where the privacy measure does not satisfy concurrent composition. Then `seq_wrapper` on line 58 is a wrapper that will cause wrapped queryables to send an `AskPermission` query with their self-reported id to this queryable before answering any query. By line 83, permission is only granted if the most recently spawned child is asking for permission to execute a query. This ensures that an adversary may only interact with the most recently spawned child mechanism.

Now that we've shown a correspondence between the pseudocode and Algorithm 6 of [HST+23], the proof from Appendix B.1 shows that the pseudocode implements a valid $\mathcal{D}$ DP privacy loss accumulator for interactive mechanisms.

The transition function is then used to construct a queryable on line 93. □

Proof of pending privacy guarantee. The relevant handler is on line **??**. The proof follows similarly to the proof of the privacy guarantee, but with the addition of the privacy loss of the pending query on line **??**.

In the case of a pending privacy loss query, since the privacy loss query does not change the privacy loss of the odometer, `PendingLoss.Same` is returned on line **??**. □

References

[HST+23] Samuel Haney, Michael Shoemate, Grace Tian, Salil Vadhan, Andrew Vyrros, Vicki Xu, and Wanrong Zhang. Concurrent composition for interactive differential privacy with adaptive privacy-loss parameters, 2023.