Privacy Proofs for OpenDP: Row Fallible Transform

Grace Tian

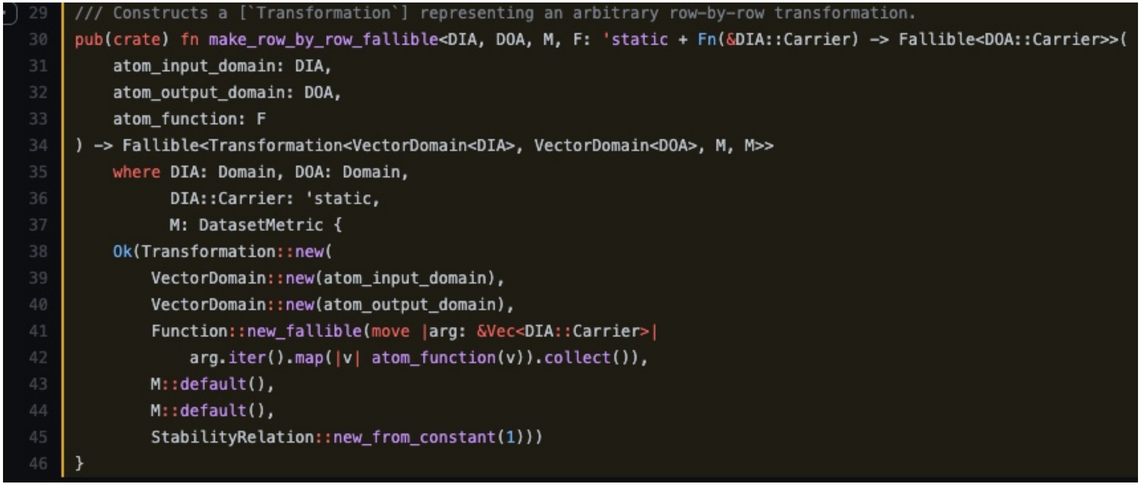
Summer 2021

Contents

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_row_by_row` function implementing the row transform function. This is defined in lines 29-46 of the file `manipulation.rs` in the Git repository (https://github.com/opendp/opendp/blob/main/rust/opendp/src/trans/manipulation.rs#L29-L46).



```
29  /// Constructs a [`Transformation`] representing an arbitrary row-by-row transformation.
30  pub(crate) fn make_row_by_row_fallible<DIA, DOA, M, F: 'static + Fn(&DIA::Carrier) -> Fallible<DOA::Carrier>>(
31      atom_input_domain: DIA,
32      atom_output_domain: DOA,
33      atom_function: F
34  ) -> Fallible<Transformation<VectorDomain<DIA>, VectorDomain<DOA>, M, M>>
35      where DIA: Domain, DOA: Domain,
36            DIA::Carrier: 'static,
37            M: DatasetMetric {
38      Ok(Transformation::new(
39          VectorDomain::new(atom_input_domain),
40          VectorDomain::new(atom_output_domain),
41          Function::new_fallible(move |arg: &Vec<DIA::Carrier>|
42              arg.iter().map(|v| atom_function(v)).collect()),
43          M::default(),
44          M::default(),
45          StabilityRelation::new_from_constant(1)))
46  }
```

1.2 Pseudo Code in Python

Preconditions

To ensure the correctness of the output, we require the following preconditions:

- **User-specified types:**

  - Variable atom_input_domain has type DIA
  - Variable atom_output_domain has type DOA
  - Variable atom_function has type F
  - Types DIA and DOA have trait Domain
  - Type F has trait Fn(&DIA::Carrier) -> Fallible(DOA::Carrier) (grace) Ask Mike about this.

Postconditions

- A Transformation is returned (i.e., if a Transformation cannot be returned successfully, then an error should be returned).

```python
def make_row_by_row(atom_input_domain : DIA, atom_output_domain : DOA, atom_function : F):
    input_domain = VectorDomain(DIA);
    output_domain = VectorDomain(DOA)
    input_metric = SymmetricDistance()
    output_metric = SymmetricDistance()

    def Relation(d_in : u32, d_out : u32) -> bool:
        return d_out <= d_in*1

    # how do I incorporate "fallible" for function call?
    def function(data : Vec(DIA)) -> Vec(DOA):
        return list(map(atom_function, data))

    return Transformation(input_domain, output_domain, function, input_metric, output_metric, stability_relation=Relation)
```

2 Proof

The necessary definitions for the proof can be found at "List of definitions used in the proofs".

Theorem 2.1. *For every setting of the input parameters (atom_input_domain, atom_output_domain, atom_function) to make_row_by_row such that the given preconditions hold, the transformation returned by make_row_by_row has the following properties:*

1. (Appropriate output domain). *For every element v in input_domain, function(v) is in output_domain.*

2. (Domain-metric compatibility). *The domain input_domain matches one of the possible domains listed in the definition of input_metric, and likewise output_domain matches one of the possible domains listed in the definition of output_metric.*

3. (Stability guarantee). *For every pair of elements v, w in input_domain and for every pair (d_in, d_out), where d_in is of the associated type for input_metric and d_out is the associated type for output_metric, if v, w are d_in-close under input_metric and Relation(d_in, d_out) = True, then function(v), function(w) are d_out-close under output_metric.*

Proof. 1. **(Appropriate output domain).** In the case of `make_row_by_row`, this corresponds to showing that for every vector v of elements of type `DIA`, `function`(v) is a vector of elements of type `DOA`.

The `function`(v) has type `Vec(DOA)` follows from the assumption that element v is in `input_domain` and from the type signature of `function` in line 11 of the pseudocode (Section 1.2), which takes in an element of type `Vec(DIA)` and returns an element of type `Vec(DOA)`. If the Rust code compiles correctly, then the type correctness follows from the definition of the type signature enforced by Rust. Otherwise, the code raises an exception for incorrect input type.

(grace) I think checking type signature is sufficient for this pf. How should this change if it is fallible?

2. **(Domain-metric compatibility).** The Symmetric distance is both the `input_metric` and `output_metric`. Symmetric distance is compatible with `VectorDomain(T)` for any generic type `T`, as stated in "List of definitions used in the pseudocode". The theorem holds because for `make_row_by_row_fallible`, the input domain is `VectorDomain(DIA)` and the output domain is `VectorDomain(DOA)`.

3. **(Stability guarantee).** From Lemma 3.1 in "List of definitions used in the proofs" on the symmetric distance of row transform, we know that

$$d_{Sym}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w).$$

(grace) Not sure how to change this for fallible.

Because `Relation`(`d_in`, `d_out`) $=$ `True`, it follows that `d_in` $\leq$ `d_out` by the `is_equal` stability relation defined in the pseudocode. Since vector inputs v, w are `d_in`-close, then the symmetric distance is bounded by `d_in` by definition the symmetric distance is bounded by d_{in}: $d_{Sym}(v, w) \leq$ `d_in`. It finally follows that the transformations are `d_out`-close: $d_{Sym}(\texttt{function}(v), \texttt{function}(w)) \leq$ `d_out`.

□