Privacy Proofs for OpenDP: Clamping

Sílvia Casacuberta

Summer 2021

Contents

1 Versions of definitions documents

When looking for definitions for terms that appear in this document, the following versions of the definitions documents should be used.

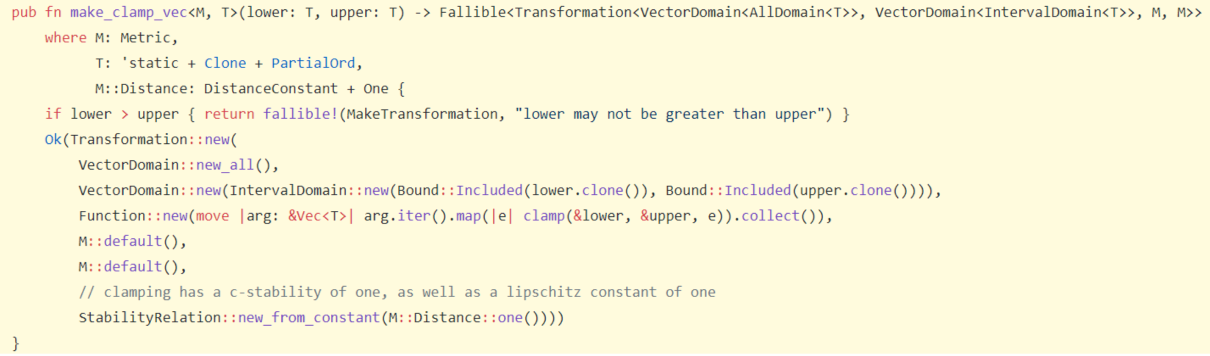
- **Pseudocode definitions document:** This proof file uses the version of the pseudocode definitions document available as of September 6, 2021, which can be found at this link (archived here).
- **Proof definitions document:** This file uses the version of the proof definitions document available as of September 6, 2021, which can be found at this link (archived here).

2 Algorithm implementation

2.1 Code in Rust

The current OpenDP library contains the `make_clamp_vec` function implementing the clamping function. This is defined in lines 25-38 of the file `manipulation.rs` in the Git repository[1] (`https://github.com/opendp/opendp/blob/58feb788ec78ce739caaf3cad8471c79fd5e7132/rust/opendp/src/trans/manipulation.rs#L25-L38`).

[1] As of June 16, 2021. Since then, the trait `PartialOrd` has been updated to `TotalOrd`.



```
pub fn make_clamp_vec<M, T>(lower: T, upper: T) -> Fallible<Transformation<VectorDomain<AllDomain<T>>, VectorDomain<IntervalDomain<T>>, M, M>>
    where M: Metric,
          T: 'static + Clone + PartialOrd,
          M::Distance: DistanceConstant + One {
    if lower > upper { return fallible!(MakeTransformation, "lower may not be greater than upper") }
    Ok(Transformation::new(
        VectorDomain::new_all(),
        VectorDomain::new(IntervalDomain::new(Bound::Included(lower.clone()), Bound::Included(upper.clone()))),
        Function::new(move |arg: &Vec<T>| arg.iter().map(|e| clamp(&lower, &upper, e)).collect()),
        M::default(),
        M::default(),
        // clamping has a c-stability of one, as well as a lipschitz constant of one
        StabilityRelation::new_from_constant(M::Distance::one())))
}
```

2.2 Pseudocode in Python

We present a simplified Python-like pseudocode of the Rust implementation below. The necessary definitions for the pseudocode can be found at "List of definitions used in the pseudocode".

The use of `code`*-style parameters in the preconditions section below (for example,* `input_domain`*) means that this information should be passed along to the* `Transformation` *constructor.*

Preconditions

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

- **User-specified types:**
    - Type `T` must have trait `TotalOrd`.

Postconditions

- Either a valid `Transformation` is returned or an error is returned.

```
def MakeClamp(L: T, U: T):
    input_domain = VectorDomain(AllDomain(T))
    output_domain = VectorDomain(IntervalDomain(L, U))
    input_metric = SymmetricDistance()
    output_metric = SymmetricDistance()

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

    def function(data: Vec[T]) -> Vec[T]:
        def clamp(x: T) -> T:
            return max(min(x, U), L)
        return list(map(clamp, data))

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

3 Proof

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

3.1 Symmetric distance

Theorem 1. *For every setting of the input parameters* `(L, U)` *to* `MakeClamp` *such that the given preconditions hold,* `MakeClamp` *raises an exception (at compile time or run time) or returns a valid transformation with 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` *has the associated type for* `input_metric` *and* `d_out` *has 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. **(Appropriate output domain).** In the case of `MakeClamp`, this corresponds to showing that for every vector v of elements of type `T`, `function`(v) is a vector of elements of type `T` which are contained in the interval `[L, U]`. For that, we need to show two things: first, that `function(v)` has type `Vec[T]`. Second, that they belong to the interval `[L, U]`.

Firstly, that `function`(v) has type `Vec[T]` follows from the assumption that element v is in `input_domain` and from the type signature of `function` in line 10 of the pseudocode (Section 2.2), which takes in an element of type `Vec[T]` and returns an element of type `Vec[T]`. 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.

Secondly, we need to show that the vector entries belong to the interval `[L, U]`. For that, it is foremost necessary that `L` $\leq$ `U`. This condition is already checked when declaring `output_domain = VectorDomain(IntervalDomain(L, U))` in line 3 of the pseudocode. This check already exists via the construction of `IntervalDomain`, which returns an error if `L` $>$ `U`. The rest follows from the definition of `clamp` in line 11. According to line 11 in the pseudocode, there are 3 possible cases to consider:

1. `x` $>$ `U`: then `clamp(x)` returns `U`.

2. `x` $\in$ `[L, U]`: then `clamp(x)` returns `x`.

3. `x` $<$ `L`: then `clamp(x)` returns `L`.

In all three cases, the returned value of type `T` is contained in the interval `[L, U]`. Hence, the vector `function`(v) returned in line 13 of the pseudocode is an element of `output_domain`.

Lastly, both `L` and `U` have type `T` by the type signature of `MakeClamp`. Both the definition of `IntervalDomain` and that of the `clamp` function (line 11 in the pseudocode, which uses the `min` and `max` functions) require that `T` implements `TotalOrd`, which holds by the preconditions.

(Domain-metric compatibility). For `MakeClamp`, both the input and output cases correspond to showing that `VectorDomain(T)` is compatible with the symmetric distance metric. This follows directly from the definition of symmetric distance, as stated in "List of definitions used in the pseudocode".

(Stability guarantee). Throughout the stability guarantee proof, we can assume that $\texttt{function}(v)$ and $\texttt{function}(w)$ are in the correct output domain, by the *appropriate output domain property* shown above.

Since by assumption $\texttt{relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, by the `MakeClamp` stability relation (as defined in line 7 in the pseudocode), we have that $\texttt{d\_in} \leq \texttt{d\_out}$. Moreover, v, w are assumed to be `d_in`-close. By the definition of the symmetric difference metric, this is equivalent to stating that $d_{Sym}(v, w) = |\text{MultiSet}(v) \Delta \text{MultiSet}(w)| \leq \texttt{d\_in}$.

Let $\mathcal{X}$ be the domain of all elements of type `T`. By applying the histogram notation,[2] it follows that

$$d_{Sym}(v, w) = \|h_v - h_w\|_1 = \sum_{z \in \mathcal{X}} |h_v(z) - h_w(z)| \leq \texttt{d\_in} \leq \texttt{d\_out}.$$

By Definition 3.10 in "List of definitions used in the proofs" and the definition of `clamp` in lines 11–13 in the pseudocode, it follows that the `function` defined in `MakeClamp`, which maps elements from `VectorDomain` to `VectorDomain`, is a row transform. Therefore, by Lemma 3.13 in "List of definitions used in the proofs", it follows that for every pair of elements v, w in `input_domain`,

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

Then, by the initial assumptions `relation(d_in, d_out) = True` and $\texttt{d\_in} \leq \texttt{d\_out}$, it follows that

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

Hence,

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

as we wanted to show. □

[2] Note that there is a bijection between multisets and histograms, which is why the proof can be carried out with either notion. For further details, please consult https://www.overleaf.com/project/60d214e390b337703d200982.

4 Old proof for the stability guarantee

Salil suggested introducing the definition of row transform and adding a general lemma for its stability guarantee, as shown in Section 3.3 in "List of definitions used in the proofs". However, we keep the longer old proof (case-by-case) for completeness, and in case any issues arise when revising the definition and theorems relating to row transforms

Proof. **(Stability guarantee).** Throughout the stability guarantee proof, we can assume that $\texttt{function}(v)$ and $\texttt{function}(w)$ are in the correct output domain, by the *appropriate output domain property* shown above.

Since by assumption $\texttt{relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, by the `MakeClamp` stability relation (as defined in line 7 in the pseudocode), we have that $\texttt{d\_in} \leq \texttt{d\_out}$. Moreover, v, w are assumed to be `d_in`-close. By the definition of the symmetric difference metric, this is equivalent to stating that $d_{Sym}(v, w) = |\text{MultiSet}(v)\Delta\text{MultiSet}(w)| \leq \texttt{d\_in}$.

Let $\mathcal{X}$ be the domain of all elements of type `T`. By applying the histogram notation,[3] it follows that

$$d_{Sym}(v, w) = \|h_v - h_w\|_1 = \sum_{z\in\mathcal{X}} |h_v(z) - h_w(z)| \leq \texttt{d\_in} \leq \texttt{d\_out}.$$

We now consider $\text{MultiSet}(\texttt{function}(v))$ and $\text{MultiSet}(\texttt{function}(w))$. For each element $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$, where z has type `T`, if $z \in \text{MultiSet}(v)\Delta\text{MultiSet}(w)$, we will assume wlog that $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$. We consider the following cases:

1. $z > \texttt{U}$ or $z < \texttt{L}$: then, in the former case, $\texttt{clamp}(z) = \texttt{U}$. First consider the case when $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets. Then, $|h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| = 0$ because we have both $h_{\texttt{function}(v)}(z) = 0$ and $h_{\texttt{function}(w)}(z) = 0$. Thus the sum

$$\sum_{z\in\mathcal{X}} |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$$

remains invariant, because the quantity $|h_v(z) - h_w(z)|$ is added to $|h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})|$, given that $\texttt{clamp}(z) = \texttt{U}$.

Suppose z has multiplicity $k_v \geq 0$ in $\text{MultiSet}(v)$ and multiplicity $k_w \geq 0$ in $\text{MultiSet}(w)$, where $k_v \neq k_w$. After considering z, the value $h_{\texttt{function}(v)}(\texttt{U})$ becomes $h_{\texttt{function}(v)}(\texttt{U}) + k_v$, and $h_{\texttt{function}(w)}(\texttt{U})$ becomes $h_{\texttt{function}(w)}(\texttt{U}) + k_w$. Hence the quantity $|h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})|$ increases by at most $|h_v(z) - h_w(z)|$, since, by the triangle inequality,

$$|(h_{\texttt{function}(v)}(\texttt{U}) + k_v) - (h_{\texttt{function}(w)}(\texttt{U}) + k_w)| \leq$$
$$\leq |h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})| + |k_v - k_w| =$$
$$= |h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})| + |h_v(z) - h_w(z)|.$$

The same argument applies whenever $z < \texttt{L}$.[4]

[3] Note that there is a bijection between multisets and histograms, which is why the proof can be carried out with either notion. For further details, please consult https://www.overleaf.com/project/60d214e390b337703d200982.

[4] The first subcase discussed here, i.e., when $k_v = k_w$, is also proven by the triangle inequality expression above, but it seemed clean to separate the case where the total sum remains invariant.

2. $z \in$ `(L, U)`: then, $\texttt{clamp}(z) = z$. Since $h_v(z) = h_{\texttt{function}(v)}(z)$ and $h_v(w) = h_{\texttt{function}(w)}(z)$, it follows that $|h_v(z) - h_w(z)| = |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$. Hence the histogram count, i.e., the quantity

   $$\sum_{z \in \mathcal{X}} |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|,$$

   remains invariant.

3. $z = \texttt{U}$ or $z = \texttt{L}$: in the former case, $\texttt{clamp}(z) = \texttt{U}$. If $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets, then the histogram count remains invariant under the addition of element z. Otherwise, if $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$, or if z is in their union but with different multiplicity, then element z can increase the quantity $|h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})|$ by at most $|h_v(z) - h_w(z)|$, following the same reasoning with the triangle inequality as in case 2.

   The same argument applies whenever $z = \texttt{L}$.

By aggregating the three cases above, we conclude that

$$\sum_{z \in \mathcal{X}} |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| \leq \sum_{z \in \mathcal{X}} |h_v(z) - h_w(z)|.$$

By the initial assumptions, we recall that $\texttt{d\_in} \leq \texttt{d\_out}$, and that v, w are $\texttt{d\_in}$-close. Then,

$$\sum_{z \in \mathcal{X}} |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| \leq \sum_{z \in \mathcal{X}} |h_v(z) - h_w(z)| \leq \texttt{d\_in} \leq \texttt{d\_out}.$$

Therefore,

$$|\text{MultiSet}(\texttt{function}(v)) \Delta \text{MultiSet}(\texttt{function}(w))| \leq \texttt{d\_out},$$

as we wanted to show. □