Privacy Proofs for OpenDP: Clamping II

Sílvia Casacuberta

Summer 2021

Contents

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_clamp` function implementing the clamping function for both the symmetric distance (with vectors) and the absolute distance (with elements) cases. In this proof, we deal with the clamping with absolute distance constructor. This is defined in lines 25-38 of the file `clamp.rs` in the Git repository[1] (`https://github.com/opendp/opendp/blob/main/rust/opendp/src/trans/clamp.rs#L46-L72`).

1.2 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".

Preconditions

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

- **User-specified types:**
  - Type `T` must have traits `TotalOrd`,[2] `DistanceCast`, and `Sub(Output=T)`.

[1] We refrain from using the clampable domain that is used in the pseudocode until `TotalOrd` has been fully implemented.

[2] For now, the OpenDP library only implements `PartialOrd`, but `TotalOrd` will soon be implemented.

```rust
impl<T, Q> ClampableDomain<AbsoluteDistance<Q>> for AllDomain<T>
    where Q: DistanceConstant + One,
          T: 'static + Clone + PartialOrd + DistanceCast + Sub<Output=T> {
    type Atom = T;
    type OutputDomain = IntervalDomain<T>;

    fn new_input_domain() -> Self { AllDomain::new() }
    fn new_output_domain(lower: Self::Atom, upper: Self::Atom) -> Fallible<Self::OutputDomain> {
        IntervalDomain::new(Bound::Included(lower), Bound::Included(upper))
    }
    fn clamp_function(lower: Self::Atom, upper: Self::Atom) -> Function<Self, Self::OutputDomain> {
        Function::new(move |arg: &T| clamp(&lower, &upper, arg).clone())
    }
    fn stability_relation(lower: Self::Atom, upper: Self::Atom) -> StabilityRelation<AbsoluteDistance<Q>, AbsoluteDistance<Q>> {
        // the sensitivity is at most upper - lower
        StabilityRelation::new_all(
            // relation
            enclose!((lower, upper), move |d_in: &Q, d_out: &Q|
                Ok(d_out.clone() >= min(d_in.clone(), Q::distance_cast(upper.clone() - lower.clone())?))),
            // forward map
            Some(move |d_in: &Q|
                Ok(Box::new(min(d_in.clone(), Q::distance_cast(upper.clone() - lower.clone())?)))),
            // backward map
            None::<fn(&_)->_>
        )
    }
}
```

Postconditions

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

```python
def MakeClampAbs(L: T, U: T):
    input_domain = AllDomain(T)
    output_domain = IntervalDomain(L, U)
    input_metric = AbsoluteDistance(Q)
    output_metric = AbsoluteDistance(Q)

    def Relation(d_in: Q, d_out: Q) -> bool:
        return d_out >= min(d_in, U-L)

    def function(x: T) -> T:
        return max(min(x, U), L)

    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".

2.1 Symmetric Distance

Theorem 1. *For every setting of the input parameters* `L, U` *to* `MakeClampAbs` *such that the given preconditions hold, the transformation returned by* `MakeClampAbs` *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. **(Appropriate output domain).** In the case of `MakeClampAbs`, this corresponds to showing that for every element x of type `T`, `function`(x) is an element of type `T` and contained in the interval `[L, U]`. For that, we need to show two things: first, that `function(x)` has type `T`. Second, that it belongs to the interval `[L, U]`.

Firstly, that `function`(x) has type `T` follows from the assumption that element x is in `input_domain` and from the type signature of `function` in line 10 of the pseudocode (Section 1.2), which takes in an element of type `T` and returns an element of type `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]`. This follows from the definition of `function` in line 10. According to line 10 in the pseudocode, there are 3 possible cases to consider:

1. `x` $>$ `U`: then `function(x)` returns `U`.
2. `x` $\in$ `[L, U]`: then `function(x)` returns `x`.
3. `x` $<$ `L`: then `function(x)` returns `L`.

In all three cases, the returned value of type `T` is contained in the interval `[L, U]`. Hence, the element `function`(x) returned in line 11 of the pseudocode is an element of `output_domain`.

Lastly, the necessary condition that `L` $\leq$ `U` is checked when declaring `output_domain = 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`. Both `L` and `U` have type `T` by their precondition requirement. Both the definition of `IntervalDomain` and that of `function` (line 10 in the pseudocode, which uses the `min` and `max` functions) require that `T` implements `TotalOrd`, which holds by the preconditions.

(Domain-metric compatibility). For `MakeClampAbs`, this corresponds to showing that both `AllDomain(T)` and `IntervalDomain(L, U)` are compatible with absolute distance. The first one follows directly from the definition of absolute distance, as stated in "List of definitions used in the proofs". The second one follows from the compatibility of absolute distance and `AllDomain(T)` along with the fact that `IntervalDomain(L, U)` is a subdomain of `AllDomain(T)`. By Theorem 2.1 in "List of definitions used in the pseudocode", this implies that `IntervalDomain(L, U)` is compatible with absolute distance as well.

(silvia) Flag: this is an example of the subdomain issues that we have been discussing during the week of July 19. Hence this paragraph might need some phrasing updates when the compatibility pairing constructor and the subdomain trait are implemented.

(Stability guarantee). Throughout the stability guarantee proof, we can assume that $\texttt{function}(x)$ and $\texttt{function}(y)$ 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 `MakeClampAbs` 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 absolute distance metric, this is equivalent to stating that $d_{Abs}(x, y) = |x - y| \leq \texttt{d\_in}$.

We now consider the values $\texttt{function}(x)$ and $\texttt{function}(y)$. There are three possible cases to consider:

1. Both $x \in \texttt{[L, U]}$ and $y \in \texttt{[L, U]}$: in this case, $x = \texttt{function}(x)$ and $y = \texttt{function}(y)$. Hence, $d_{Abs}(x, y) = |x - y| = |\texttt{function}(x) - \texttt{function}(y)|$.

2. Wlog, $x \in \texttt{[L, U]}$ and $y \notin \texttt{[L, U]}$: if $y < \texttt{L}$ then $y < \texttt{function}(y) = \texttt{L}$, and if $y > \texttt{U}$, then $y > \texttt{function}(y) = \texttt{U}$. In both cases, it follows that $|\texttt{function}(y) - x| < |y - x| = d_{Abs}(x, y)$, since $\texttt{function}(x) = x$.

3. Both $x, y \notin \texttt{[L, U]}$: in this case, if both $x, y < \texttt{L}$ or both $x, y > \texttt{U}$, then $|\texttt{function}(x) - \texttt{function}(y)| = 0$. Because the absolute value metric is always non-negative, it follows that $|\texttt{function}(x) - \texttt{function}(y)| \leq |x - y|$. On the other hand, if $x < \texttt{L}$ and $y > \texttt{U}$ or viceversa, then $|\texttt{U} - \texttt{L}| < |x - y|$. Since by the appropriate domain property we know that $\texttt{function}(x), \texttt{function}(y) = \texttt{[L, U]}$, it follows that $|\texttt{function}(x) - \texttt{function}(y)| \leq |\texttt{U} - \texttt{L}| < |x - y|$.

In all cases described above, we conclude that

$$d_{Abs}(\texttt{function}(x), \texttt{function}(y)) = |\texttt{function}(x) - \texttt{function}(y)| \leq |x - y| = d_{Abs}(x, y).$$

By the initial assumptions, we recall that $\texttt{d\_in} \leq \texttt{d\_out}$, and that x, y are `d_in`-close. Then,

$$d_{Abs}(\texttt{function}(x), \texttt{function}(y)) \leq |x - y| \leq \texttt{d\_in} \leq \texttt{d\_out}.$$

Therefore,

$$d_{Abs}(\texttt{function}(x), \texttt{function}(y)) \leq \texttt{d\_out},$$

as we wanted to show. □

(silvia) Flag: what would be the best way to present this now that we added the row-by-row transform lemma and in conjunction with the vector clamping transformation (with symmetric distance)? Or expand the row-by-row transform with absolute distance and vectors of length 1 (i.e., just `AllDomain`)?