Privacy Proofs for OpenDP: Bounded Sum with Known n

Sílvia Casacuberta, Grace Tian, Connor Wagaman

Version as of September 21, 2021 (UTC)

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 transformation `make_bounded_sum_n` implementing the bounded sum function with known n. This is defined in lines 29-49 of the file `mod.rs` in the Git repository[1] (`https://github.com/opendp/opendp/blob/8bbb0fab1da9b86c50235a36d7026189be43b1ab/rust/opendp/src/trans/sum/mod.rs#L29-L49`).

[1]Updated on September 20, 2021.

```rust
pub fn make_sized_bounded_sum<T>(
    size: usize, bounds: (T, T)
) -> Fallible<Transformation<SizedDomain<VectorDomain<BoundedDomain<T>>>, AllDomain<T>, SymmetricDistance, AbsoluteDistance<T>>>
    where T: DistanceConstant<IntDistance> + Sub<Output=T>, for <'a> T: Sum<&'a T> + ExactIntCast<usize> + CheckedMul + CheckNull,
          IntDistance: InfCast<T> {
    let size_ = T::exact_int_cast(size)?;
    let (lower, upper) = bounds.clone();
    if lower.checked_mul(&size_).is_none()
        || upper.checked_mul(&size_).is_none() {
        return fallible!(MakeTransformation, "Detected potential for overflow when computing function.")
    }
    Ok(Transformation::new(
        SizedDomain::new(VectorDomain::new(
            BoundedDomain::new_closed(bounds)?), size),
        AllDomain::new(),
        Function::new(|arg: &Vec<T>| arg.iter().sum()),
        SymmetricDistance::default(),
        AbsoluteDistance::default(),
        // d_out >= d_in * (M - m) / 2
        StabilityRelation::new_from_constant((upper - lower) / T::exact_int_cast(2)?)))
}
```

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 in Section 1.

Preconditions

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

- **User-specified types:**
  - Variable `n` must be of type `usize`.
  - Type `T` must have traits `DistanceConstant(IntDistance)`, `TotalOrd`, `CheckedMul`, `Sum(Output=T)`, `Sub(Output=T)`, and `ExactIntCast(usize)`.
  - `IntDistance` must have trait `InfCast(T)`. (Note that this bullet point is not needed in this proof, but it is needed in the code so a hint can be constructed; otherwise a binary search would be needed to construct the hint.)
  - Variables `U` and `L` must be of type `T`, and we must have $\texttt{L} \leq \texttt{U}$. **Question:** Is this inequality correct? Should it be, for example, $\leq$? Also, does bounded sum require all $v \in [L, U]$ or $v \in (L, U)$ or something else? (connor) Changed this from $\texttt{L} < \texttt{U}$ to $\texttt{L} \leq \texttt{U}$, based on the code at `https://github.com/opendp/opendp/blob/main/rust/opendp/src/dom.rs#L95-L96`

Postconditions

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

```python
def MakeBoundedSumN(L: T, U: T, n: usize):
    input_domain = SizedDomain(VectorDomain(IntervalDomain(L, U)), n)
    output_domain = AllDomain(T)
    input_metric = SymmetricDistance()
    output_metric = AbsoluteDistance(T)
```

```
6
7     n_ = exact_int_cast(n, T)
8     if checked_mul(L, n_).is_none or checked_mul(U, n_).is_none:
9         raise Exception('Potential overflow')
10
11    def relation(d_in: u32, d_out: T) -> bool:
12        return d_out >= inf_cast(d_in, T) * (U - L) / exact_int_cast(2, T)
13
14    def function(data: Vec[T]) -> T:
15        return sum(data)
16
17    return Transformation(input_domain, output_domain, function,
      input_metric, output_metric, stability_relation = relation)
```

(silvia) There is a naming difference between the code and the pseudocode due to a recent change in the Rust code: old n to new *size* (but I prefer using n in the proof – more concise).

3 Proof

3.1 Symmetric Distance

Theorem 1. *For every setting of the input parameters* `(L, U, n)` *to* `MakeBoundedSumN` *such that the given preconditions hold,* `MakeBoundedSumN` *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`(v) *are* `d_out`*-close under* `output_metric`*.*

Proof. **(Part 1 – appropriate output domain).** In the case of `MakeBoundedSumN`, this corresponds to showing that for every vector `v` in `SizedDomain(VectorDomain(IntervalDomain (L, U)), n)`, where `L` and `U` have type `T`, `function(v)` belongs to `AllDomain(T)`.

The output correctness follows from the type signature of `function` as defined in line 14 and from the overflow check done through the `checked_mul` function in line 9.

(connor) I recommend including a discussion here of the `exact_int_cast`. You should demonstrate that the checked mul thing does the check that we want it to. In doing that demonstration, it is important to show that the number of rows `n` is not underestimated. The exact_int_cast ensures that. `exact_int_cast` is needed to ensure that our checked mul checks the thing we want it to check

The latter ensures that `function(v)` is contained within the interval `[get_min_value(T), get_max_value(T)]`, and hence prevents any overflow from occurring in line 15. If overflow can occur, line 9 will raise an exception for potential overflow. The former automatically enforces that `function(v)` has return type `T`. Since the Rust code successfully compiles, by the type signature the appropriate output domain property must hold. Otherwise, the code will raise an exception for incorrect input type.

(Part 2 – domain-metric compatibility). For `MakeBoundedSumN`, proving this part corresponds to showing that `SizedDomain(VectorDomain(IntervalDomain (L, U)), n)` is compatible with `SymmetricDistance()` (see line 4), and that `AllDomain(T)` is compatible with `AbsoluteDistance(T)` (see line 5). The former follows directly from the list of compatible domains in the definition of symmetric distance, as described in the pseudocode definitions document in section 1. The latter follows directly from the list of compatible domains in the definition of absolute distance, as described in the pseudocode definitions document in section 1.

(connor) Problem with current stability guarantee. Ideas copied from Slack, so sorry about the formatting.

I think if we did ((upper - lower) + 1) / 2, then we'd be in good shape. This is because we want to make sure that our d_out is big enough (for privacy, it's worse to have d_out small). Because upper and lower are both ints, the difference will be an int.

Current relation: Assume (U-L) divisible by 2. Then we're good. Now assume (U-L) not divisible by 2. Then `(U-L)/2` ¡ (U-L)/2, where `(U-L)/2` is done on a computer. Uh-oh.

"+1" relation: Assume (U-L) divisible by 2. Then `((U-L)+1)/2` = (U-L)/2, so we're good. Now assume (U-L) not divisible by 2. Then ((U-L)+1) divisible by 2, so ((U-L)+1)/2 is greater than the true (U-L)/2, so we're good.

Another idea is to do something like "if upper-lower is odd, do the +1 strategy. If it's even, use the original relation". Not sure if this conditional structure has any implementation-based advantages or disadvantages, but it'll be a tight bound.

(connor) End of notes about the current stability guarantee.

(silvia) We ended up concluding that the best option is to move the constant to the other side – namely, $2 \cdot \texttt{d\_out} \geq \texttt{d\_in} \cdot (\texttt{U} - \texttt{L})$.

(Part 3 – stability guarantee). Throughout the stability guarantee proof, we can assume that `function(v)` and `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 `MakeBoundedSumN` stability relation (as defined in line 11 in the pseudocode), we have that $\texttt{d\_out} \geq \texttt{d\_in} \cdot (\texttt{U} - \texttt{L})/2$. Moreover, $\texttt{v}, \texttt{w}$ are assumed to be `d_in`-close. By the definition of the symmetric distance metric in the proof definitions document linked in section 1, this is equivalent to stating that $d_{Sym}(v, w) = |\text{MultiSet}(v)\Delta\text{MultiSet}(w)| \leq \texttt{d\_in}$.

Further, applying the histogram notation,[2] it follows that

$$d_{Sym}(v, w) = \|h_v - h_w\|_1 = \sum_z |h_v(z) - h_w(z)| \leq \texttt{d\_in}.$$

We want to show that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \frac{\texttt{U-L}}{2}.$$

This would imply that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \frac{\texttt{U-L}}{2} \leq \texttt{d\_in} \cdot \frac{\texttt{U-L}}{2}, \tag{1}$$

[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 the proof definitions document in section 1.

and by the stability relation this will imply that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \texttt{d\_out}, \tag{2}$$

as we want to see. $\square$

3.2 First proof: using the path property (adjacent pairs approach)

To show that $d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \frac{\texttt{U-L}}{2}$, we will use the three lemmas described in the section "The path property of symmetric distance on sized domains" from Section 5.2 in the document "List of definitions used in the proofs". With these three lemmas, which are applicable to `MakeBoundedSumN` because `input_domain` is a a sized domain and `input_metric` is symmetric distance, it suffices to show the following: For all vectors $x, y \in \texttt{input\_domain}$ such that $d_{Sym}(x, y) = 2$, it follows that

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

By Lemma 5.6 from "List of definitions used in the proofs", we know that vectors x, y only differ on one element, given that, by assumption, $d_{Sym}(x, y) = 2$. Wlog, let this different element be the k-th element of x and y, where $x_k = \alpha$, $y_k = \beta$ with $\alpha \neq \beta$.[3] Then,

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

$$= \left| \sum_{i=0}^{\texttt{n}-1} x_i - \sum_{i=0}^{\texttt{n}-1} y_i \right| = \left| \sum_{i=0}^{\texttt{n}-1} (x_i - y_i) \right| = |0 + (x_k - y_k)| = |\alpha - \beta| \leq |\texttt{U-L}| = \texttt{U-L},$$

since $\texttt{U} \geq \texttt{L}$. Therefore, applying Lemma 5.7 from "List of definitions used in the proofs", it follows that `function` is $(\texttt{U-L})/2$-stable. By definition, this implies that for any $v, w \in$ `input_domain`,

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot (\texttt{U-L})/2.$$

Lastly, by Equations 1 and 2 this implies that

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

as we want to prove.

[3]The first element of a vector is indexed by 0.

3.3 Second proof: direct method (all pairs approach)

3.3.1 General inequality

The general statement that we will need to prove is the following. For any elements $a_1, \ldots, a_n \in$ `[L, U]` and $b_1, \ldots, b_n \in \mathbb{Z}$ such that $\sum_i b_i = 0$,

$$\Big|\sum_i a_i b_i\Big| \leq \frac{a_{\max} - a_{\min}}{2} \cdot \Big(\sum_i |b_i|\Big).$$

Note that this corresponds to the tightest possible `[L, U]` interval.

Let u denote the vector formed by all the elements of v and w *without multiplicities* (i.e., u contains exactly once each of the elements in $\mathrm{MultiSet}(v) \cup \mathrm{MultiSet}(w)$, in any order). Let u_i denote the i-th element of u, and similarly for v and w, and let m denote $\texttt{len}(u)$. Then, by definition,

$$d_{Sym}(v, w) = \sum_z \Big|h_v(z) - h_w(z)\Big| = \sum_i \Big|h_v(u_i) - h_w(u_i)\Big|;$$

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) = \Big|\texttt{function}(v) - \texttt{function}(w)\Big| = \Big|\sum_i v_i - \sum_i w_i\Big| =$$

$$= \Big|\sum_i u_i \cdot h_v(u_i) - \sum_i u_i \cdot h_w(u_i)\Big| = \Big|\sum_i u_i \cdot (h_v(u_i) - h_w(u_i))\Big|.$$

Because by assumption $v, w \in$ `input_domain = SizedDomain(VectorDomain(IntervalDomain (L, U)), n)`, we know that $\texttt{len}(v) = \texttt{len}(w) = \texttt{n}$. Therefore,

$$\sum_i (h_v(u_i) - h_w(u_i)) = \texttt{n} - \texttt{n} = 0. \tag{3}$$

We now separate the positive values from the negative ones by defining vectors x, y, λ and μ as follows. Let

$$h_v(u_{k_1}) - h_w(u_{k_1}) \leq \ldots \leq 0 \leq h_v(u_{k_m}) - h_w(u_{k_m})$$

be the sequence of the $\{h_v(u_i) - h_w(u_i)\}$ in increasing order. Let s be the smallest value such that $h_v(u_{k_s}) - h_w(u_{k_s})$ is greater or equal to 0 (we set $t = m$ if all the values are negative). Then, we define the vector entries of x, y, λ, μ as

$$x_j = h_v(u_{k_j}) - h_w(u_{k_j}),$$

$$\lambda_j = u_j,$$

for $s \leq j \leq m$, and

$$y_j = h_v(u_{k_j}) - h_w(u_{k_j}),$$

$$\mu_j = u_j$$

for $0 \leq j < s$.[4] That is, x contains all of the positive values and y all of the negative ones.

[4] It is not necessary that the entries of x_j and y_j are ordered; only that they only contain positive and negative values, respectively, and that the λ and μ values match their corresponding indices.

Let r denote the length of vectors x and λ as constructed above, and by construction s denotes the length of vectors y and μ above (where $r+s=m$). Hence we obtain the values $x_1,\dots,x_r \geq 0$ and $y_1,\dots,y_s \leq$ for some $r,s \in \mathbb{Z}$, such that

$$\sum_i x_i + \sum_j y_j = 0 \quad \text{and so} \quad \sum_i x_i = \sum_j |y_j|,$$

by Equation 3. Then,

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) = \Big| \sum_i u_i \cdot (h_v(u_i) - h_w(u_i)) \Big| =$$

$$= |\lambda_1 x_1 + \cdots + \lambda_r x_r + \mu_1 y_1 + \cdots \mu_s y_s| = \Big| \overline{\lambda} \sum_i x_i + \overline{\mu} \sum_j y_j \Big| =$$

$$= \frac{|\overline{\lambda} - \overline{\mu}|}{2} \Big(\sum_i x_i + \sum_j |y_j| \Big) = |\overline{\lambda} - \overline{\mu}| \sum_i x_i,$$

where

$$\overline{\lambda} = \frac{\sum \lambda_i x_i}{\sum x_i}, \quad \overline{\mu} = \frac{\sum \mu_j y_j}{\sum y_j} = \frac{\sum \mu_j |y_j|}{\sum |y_j|};$$

i.e., they correspond to the weighted arithmetic mean.

By definition of the `input_domain`, the entries of v and w are contained within the interval `[L, U]`, and hence $\texttt{U} \geq \max\{\lambda_i, \mu_j\}$ and $\texttt{L} \leq \min\{\lambda_i, \mu_j\}$. Then,

$$\frac{\texttt{U-L}}{2} \Big(\sum_i x_i + \sum_j |y_j| \Big) = \frac{\texttt{U-L}}{2} \cdot 2 \sum_i x_i = (\texttt{U-L}) \sum_i x_i.$$

Since $|\overline{\lambda} - \overline{\mu}| \leq \texttt{U-L}$, it follows that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) = \frac{\overline{\lambda} - \overline{\mu}}{2} \Big(\sum_i x_i + \sum_j |y_j| \Big) = \frac{\overline{\lambda} - \overline{\mu}}{2} \Big(\sum_i x_i - \sum_j y_j \Big) =$$

$$\frac{\overline{\lambda} - \overline{\mu}}{2} \Big(\sum_i |h_v(u_i) - h_w(u_i)| \Big) = \frac{\overline{\lambda} - \overline{\mu}}{2} \cdot d_{Sym}(v, w) \leq \frac{\texttt{U-L}}{2} \cdot d_{Sym}(v, w).$$

Hence,

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \frac{\texttt{U-L}}{2} \cdot d_{Sym}(v, w),$$

as we wanted to show.