

# fn make\_stable\_truncate

Michael Shoemate

July 3, 2025

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

Proves soundness of `make_stable_truncate` in `mod.rs` at commit `f5bb719` (outdated<sup>1</sup>).

## 1 Hoare Triple

Precondition

Caller Verified

None

Function

```
1 def make_stable_truncate(  
2     input_domain: DslPlanDomain,  
3     input_metric: FrameDistance[SymmetricIdDistance],  
4     plan: DslPlan,  
5 ) -> Transformation[  
6     DslPlanDomain,  
7     DslPlanDomain,  
8     FrameDistance[SymmetricIdDistance],  
9     FrameDistance[SymmetricDistance],  
10 ]:  
11     # the identifier is protected from changes, so we can use the identifier from the input  
12     # metric  
13     # instead of the identifier from the middle_metric to match truncations  
14     input, truncations, truncation_bounds = match_truncations(  
15         plan, input_metric[0].identifier  
16     )  
17     if truncations.is_empty():  
18         return ValueError("failed to match truncation")  
19  
20     t_prior = input.make_stable(input_domain, input_metric)  
21     middle_domain, middle_metric = t_prior.output_space()  
22  
23     for bound in truncation_bounds:  
24         for key in bound.by:  
25             # raises if the key is not infallible row-by-row  
26             make_stable_expr( #  
27                 WildExprDomain(  
28                     columns=middle_domain.series_domains,  
29                     context=Context.RowByRow,
```

<sup>1</sup>See new changes with `git diff f5bb719..4332753 rust/src/transformations/make_stable_lazyframe/truncate/mod.rs`

```

30         ),
31         PartitionDistance(middle_metric[0]),
32         key,
33     )
34
35     output_domain = middle_domain.clone()
36     for truncation in truncations: #
37         output_domain = truncate_domain(output_domain, truncation)
38
39     def function(plan: DslPlan) -> DslPlan:
40         for truncation in truncations:
41             match truncation:
42                 case Truncation.Filter(predicate):
43                     plan = DslPlan.Filter(
44                         input=plan,
45                         predicate=predicate,
46                     )
47
48                 case Truncation.GroupBy(keys, aggs):
49                     plan = DslPlan.GroupBy(
50                         input=plan,
51                         keys=keys,
52                         aggs=aggs,
53                         apply=None,
54                         maintain_order=False,
55                         options=GroupbyOptions.default(),
56                     )
57         return plan
58
59     def stability_map(id_bounds: Bounds) -> Bounds:
60         #
61         total_num_ids = id_bounds.get_bound({}).per_group
62
63         # each truncation is used to derive row bounds
64         new_bounds = []
65         for truncation_bound in truncation_bounds: #
66             # each truncation is used to derive row bounds
67             new_bounds.append(
68                 truncate_id_bound( #
69                     id_bounds.get_bound(truncation_bound.by), #
70                     truncation_bound,
71                     total_num_ids,
72                 )
73             )
74         return Bounds(new_bounds)
75
76     t_truncate = Transformation.new(
77         middle_domain,
78         output_domain,
79         Function.new(function),
80         middle_metric,
81         FrameDistance(SymmetricDistance),
82         StabilityMap.new_fallible(stability_map),
83     )
84     return t_prior >> t_truncate

```

## Postcondition

**Theorem 1.1.** For every setting of the input parameters (`input_domain`, `input_metric`, `plan`) to `make_stable_truncate` such that the given preconditions hold, `make_stable_truncate` raises an exception (at compile time or run time) or returns a valid transformation. A valid transformation has the following properties:

1. (Appropriate output domain). For every element  $x$  in `input_domain`, `function(x)` is in `output_domain`

or raises a data-independent runtime exception.

2. (Stability guarantee). For every pair of elements  $x, x'$  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  $x, x'$  are  $d\_in$ -close under `input_metric`, `stability_map(d_in)` does not raise an exception, and `stability_map(d_in) ≤ d_out`, then `function(x), function(x')` are  $d\_out$ -close under `output_metric`.

*Appropriate Output Domain.* By line 26, the grouping keys are stable row-by-row transformations of the data, therefore the preconditions of 36 are satisfied. By the postcondition of 36, for every element  $x$  in `input_domain`, `function(x)` is in `output_domain` or raises a data-independent runtime exception.  $\square$

*Stability guarantee.* By line 60, `total_num_ids` is the total number of ids an individual may contribute to a dataset:

$$d_{\text{SymId}}(\text{function}(x), \text{function}(x')) \leq \text{total\_num\_ids}. \quad (1)$$

By the postcondition of `match_truncations`, for each `truncation_bound` on line 65,

$$\begin{aligned} \max_{id} ||d_{\text{Sym}}(\text{function}(x)_{id,g}, \text{function}(x')_{id,g})||_{\infty} &\leq \text{truncation.per\_group}, \\ \max_{id} ||d_{\text{Sym}}(\text{function}(x)_{id,g}, \text{function}(x')_{id,g})||_0 &\leq \text{truncation.num\_groups}, \end{aligned}$$

where  $g$  denotes the group when partitioned by `truncation_bound.by`.

The preconditions of 68 are satisfied on line 69 (`id_bound.by` is equal to `truncation.by`), so by the postcondition of `truncate_id_bound`,

$$\begin{aligned} ||d_{\text{Sym}}(\text{function}(x)_g, \text{function}(x')_g)||_{\infty} &\leq \text{row\_bound.per\_group}, \\ ||d_{\text{Sym}}(\text{function}(x)_g, \text{function}(x')_g)||_0 &\leq \text{row\_bound.num\_groups}, \end{aligned}$$

where `row_bound` denotes the return value.

For each truncation on line 65, `truncate_id_bound` on line 68 computes upper bounds on the resulting distance between adjacent datasets. All acquired bounds are valid upper bounds on the distance between the two datasets, by the postcondition of `match_truncations`, that each truncation does not invalidate the truncation bounds of the previous truncations.

It is shown that for every pair of elements  $x, x'$  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  $x, x'$  are  $d\_in$ -close under `input_metric`, `stability_map(d_in)` does not raise an exception, and `stability_map(d_in) ≤ d_out`, then `function(x), function(x')` are  $d\_out$ -close under `output_metric`.  $\square$