# fn make_stable_truncate

Michael Shoemate

February 6, 2026

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Proves soundness of `make_stable_truncate` in `mod.rs` at commit f5bb719 (outdated[1]).

# 1 Hoare Triple

## Precondition

**Caller Verified**

None

## Function

```python
def make_stable_truncate(
    input_domain: DslPlanDomain,
    input_metric: FrameDistance[SymmetricIdDistance],
    plan: DslPlan,
) -> Transformation[
    DslPlanDomain,
    DslPlanDomain,
    FrameDistance[SymmetricIdDistance],
    FrameDistance[SymmetricDistance],
]:
    # the identifier is protected from changes, so we can use the identifier from the input
    metric
    # instead of the identifier from the middle_metric to match truncations
    input, truncations, truncation_bounds = match_truncations(
        plan, input_metric[0].identifier
    )

    if truncations.is_empty():
        return ValueError("failed to match truncation")

    t_prior = input.make_stable(input_domain, input_metric)
    middle_domain, middle_metric = t_prior.output_space()

    for bound in truncation_bounds:
        for key in bound.by:
            # raises if the key is not infallible row-by-row
            make_stable_expr(  #
                WildExprDomain(
                    columns=middle_domain.series_domains,
                    context=Context.RowByRow,
```

---

[1]See new changes with `git diff f5bb719..07b9440 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 error (at compile time or run time) or returns a valid transformation. A valid transformation has the following properties:

1. (Data-independent runtime errors). For every pair of members $x$ and $x'$ in `input_domain`, $\text{invoke}(x)$

and $\texttt{invoke}(x')$ either both return the same error or neither return an error.

2. (Appropriate output domain). For every member $x$ in $\texttt{input\_domain}$, $\texttt{function}(x)$ is in $\texttt{output\_domain}$ or raises a data-independent runtime error.

3. (Stability guarantee). For every pair of members $x$ and $x'$ in $\texttt{input\_domain}$ and for every pair $(\texttt{d\_in}, \texttt{d\_out})$, where $\texttt{d\_in}$ has the associated type for $\texttt{input\_metric}$ and $\texttt{d\_out}$ has the associated type for $\texttt{output\_metric}$, if $x, x'$ are $\texttt{d\_in}$-close under $\texttt{input\_metric}$, $\texttt{stability\_map}(\texttt{d\_in})$ does not raise an error, and $\texttt{stability\_map}(\texttt{d\_in}) = \texttt{d\_out}$, then $\texttt{function}(x), \texttt{function}(x')$ are $\texttt{d\_out}$-close under $\texttt{output\_metric}$.

*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 $\texttt{input\_domain}$, $\texttt{function}(x)$ is in $\texttt{output\_domain}$ or raises a data-independent runtime exception. $\qquad\square$

*Stability guarantee.* By line 60, $\texttt{total\_num\_ids}$ is the total number of ids an individual may contribute to a dataset:

$$d_{\mathrm{SymId}}(\texttt{function}(x), \texttt{function}(x')) \leq \texttt{total\_num\_ids}. \tag{1}$$

By the postcondition of $\texttt{match\_truncations}$, for each $\texttt{truncation\_bound}$ on line 65,

$$\max_{id} ||d_{\mathrm{Sym}}(\texttt{function}(x)_{id,g}, \texttt{function}(x')_{id,g})||_\infty \leq \texttt{truncation.per\_group},$$
$$\max_{id} ||d_{\mathrm{Sym}}(\texttt{function}(x)_{id,g}, \texttt{function}(x')_{id,g})||_0 \leq \texttt{truncation.num\_groups},$$

where $g$ denotes the group when partitioned by $\texttt{truncation\_bound.by}$.

The preconditions of 68 are satisfied on line 69 ($\texttt{id\_bound.by}$ is equal to $\texttt{truncation.by}$), so by the postcondition of $\texttt{truncate\_id\_bound}$,

$$||d_{\mathrm{Sym}}(\texttt{function}(x)_g, \texttt{function}(x')_g)||_\infty \leq \texttt{row\_bound.per\_group},$$
$$||d_{\mathrm{Sym}}(\texttt{function}(x)_g, \texttt{function}(x')_g)||_0 \leq \texttt{row\_bound.num\_groups},$$

where $\texttt{row\_bound}$ denotes the return value.

For each truncation on line 65, $\texttt{truncate\_id\_bound}$ 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 $\texttt{match\_truncations}$, 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 $\texttt{input\_domain}$ and for every pair $(\texttt{d\_in}, \texttt{d\_out})$, where $\texttt{d\_in}$ has the associated type for $\texttt{input\_metric}$ and $\texttt{d\_out}$ has the associated type for $\texttt{output\_metric}$, if $x, x'$ are $\texttt{d\_in}$-close under $\texttt{input\_metric}$, $\texttt{stability\_map}(\texttt{d\_in})$ does not raise an exception, and $\texttt{stability\_map}(\texttt{d\_in}) \leq \texttt{d\_out}$, then $\texttt{function}(x), \texttt{function}(x')$ are $\texttt{d\_out}$-close under $\texttt{output\_metric}$. $\qquad\square$