fn make_stable_truncate

Michael Shoemate

April 25, 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Proves soundness of `make_stable_truncate` in `mod.rs` at commit f5bb719 (outdated[1]).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,
```

[1] See new changes with `git diff f5bb719..7efe2b26 rust/src/transformations/make_stable_lazyframe/truncate/mod.rs`

```
                    context=Context.RowByRow,
                ),
                PartitionDistance(middle_metric[0]),
                key,
            )

    output_domain = middle_domain.clone()
    for truncation in truncations: #
        output_domain = truncate_domain(output_domain, truncation)

    def function(plan: DslPlan) -> DslPlan:
        for truncation in truncations:
            match truncation:
                case Truncation.Filter(predicate):
                    plan = DslPlan.Filter(
                        input=plan,
                        predicate=predicate,
                    )

                case Truncation.GroupBy(keys, aggs):
                    plan = DslPlan.GroupBy(
                        input=plan,
                        keys=keys,
                        aggs=aggs,
                        apply=None,
                        maintain_order=False,
                        options=GroupbyOptions.default(),
                    )
        return plan

    def stability_map(id_bounds: Bounds) -> Bounds:
        #
        total_num_ids = id_bounds.get_bound({}).per_group

        # each truncation is used to derive row bounds
        new_bounds = []
        for truncation_bound in truncation_bounds: #
            # each truncation is used to derive row bounds
            new_bounds.append(
                truncate_id_bound( #
                    id_bounds.get_bound(truncation_bound.by), #
                    truncation_bound,
                    total_num_ids,
                )
            )
        return Bounds(new_bounds)

    t_truncate = Transformation.new(
        middle_domain,
        output_domain,
        Function.new(function),
        middle_metric,
        FrameDistance(SymmetricDistance),
        StabilityMap.new_fallible(stability_map),
    )
    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)` $\leq$ `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. □

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

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

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

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

where g denotes the group when partitioned by `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`,

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

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)` $\leq$ `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_metric`. □