fn make_stable_group_by

Michael Shoemate

April 22, 2025

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

Proves soundness of `make_stable_group_by` in `mod.rs` at commit f5bb719 (outdated[1]).

1 Hoare Triple

Precondition

Caller Verified

None

Function

```
def make_stable_group_by(
    input_domain: DslPlanDomain, input_metric: FrameDistance[M], plan: DslPlan
) -> Transformation[DslPlanDomain, DslPlanDomain, FrameDistance[M], FrameDistance[M]]:
    match plan:
        case DslPlan.GroupBy(input, keys, aggs, apply, maintain_order, options):
            pass
        case _:
            raise "Expected group by in logical plan"

    if apply is not None:
        raise "apply is not currently supported"

    if maintain_order:
        raise "maintain_order is wasted compute because row ordering is protected information"

    if options != GroupbyOptions.default():
        raise "options is not currently supported"

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

    expr_domain = WildExprDomain(
        columns=middle_domain.series_domains,
        context=Context.RowByRow,
    )

    for key in keys:  #
        key.make_stable(expr_domain, PartitionDistance(middle_metric[0]))
```

[1] See new changes with `git diff f5bb719..12e5590c rust/src/transformations/make_stable_lazyframe/group_by/mod.rs`

```
    for agg in aggs:  #
        check_infallible(agg, Resize.Allow)

    if middle_metric[0].identifier().is_some():  #
        raise "identifier is not currently supported"

    # prepare output domain series
    output_schema = middle_domain.simulate_schema(  #
        lambda lf: lf.group_by(keys).agg(aggs)
    )
    series_domains = [SeriesDomain.new_from_field(f) for f in output_schema]

    # prepare output domain margins
    h_keys = list(keys)

    def without_invariant(m: Margin) -> Margin:
        m.invariant = None
        return m

    margins = [
        without_invariant(m) for m in middle_domain.margins if m.by.is_subset(h_keys)
    ]

    output_domain = FrameDomain.new_with_margins(series_domains, margins)

    def stability_map(d_in: Bounds) -> Bounds:
        #
        contributed_rows = d_in.get_bound(set()).per_group
        #
        contributed_groups = d_in.get_bound(h_keys).num_groups

        influenced_groups = option_min(contributed_rows, contributed_groups)
        if influenced_groups.is_none():
            return "an upper bound on the number of contributed rows or groups is required"

        if per_group is not None: #
            per_group = per_group.inf_mul(2)

        bound = Bound(by=set(), per_group=per_group, num_groups=None)
        return Bounds([bound]) #

    t_group_agg = Transformation.new(
        middle_domain,
        output_domain,
        lambda plan: DslPlan.GroupBy(
            input=plan,
            keys=keys,
            aggs=aggs,
            apply=None,
            maintain_order=False,
            options=options,
        ),
        middle_metric,
        middle_metric,
        StabilityMap.new_fallible(stability_map),
    )

    return t_prior >> t_group_agg
```

Postcondition

Theorem 1.1. For every setting of the input parameters `(input_domain, input_metric, plan)` to `make_stable_group_by` such that the given preconditions hold, `make_stable_group_by` 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 27 the grouping keys are stable row-by-row transformations of the data. By line 30 the aggregates are infallible. Therefore the function does not raise data-dependent errors.

By the postcondition of `DslPlan.simulate_schema`, `series_domains` follows the expected schema of members in the output domain. Notice that this is a very conservative output domain, as no data descriptors are preserved except for the schema itself. On the other hand, this comes with the benefit that aggregations are black-boxes, allowing for any infallible group-wise data processing.

For the same reason, the only margins preserved are those that are a subset of the grouping keys. Among these margins, invariants are discarded. A more careful proof may be able to preserve invariants, but this is not necessary for the soundness of the transformation.

It has been shown that for every element x in `input_domain`, `function`(x) is in `output_domain` or raises a data-independent runtime exception. □

The stability guarantee doesn't attempt to claim the broadest set of possible bounds on output distances, rather it only claims a simple bound that might be useful for the user. This can be extended in the future, but is sufficient for select queries.

Stability guarantee. We first simplify the problem on line 33 by only considering datasets that differ by rows, not by identifiers.

The only bound derived is when there are no grouping keys.

We first retrieve optional upper bounds on the number of rows and groups an individual may contribute on lines 56 and 58. Both of these bounds directly correspond to the number of rows an individual may influence in the resulting dataset.

$$\begin{aligned} &\max_{x \sim x'} d_{\text{FrameDistance<M>}}(\texttt{function}(x), \texttt{function}(x')) && (1) \\ &\leq \text{option\_min}(\texttt{contributed\_rows}, \texttt{contributed\_groups}) \cdot 2 && (2) \end{aligned}$$

Since each influenced row accounts for one addition and one removal, the distance is twice the number of influenced rows. This is reflected on line 65. The resulting bound is returned on line 69, satisfying the postcondition. □