fn match_truncation_predicate

Michael Shoemate

February 1, 2026

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

Proves soundness of `match_truncation_predicate` in `mod.rs` at commit `f5bb719` (outdated[1]).

1 Hoare Triple

Compiler Verified

Types matching pseudocode.

Precondition

Compiler Verified

Types matching pseudocode.

Precondition

None

Function

```
def match_truncation_predicate(
    predicate: Expr, identifier: Expr
) -> Optional[Vec[Bound]]:
    if (
        isinstance(predicate, FunctionExpr)
        and predicate.function == BooleanFunction.AllHorizontal
    ):
        # propagate errors
        bounds = [
            match_truncation_predicate(expr, identifier) for expr in predicate.input
        ]

        # propagate nones
        if not all(bounds): #
            return None

        # appears to differ from Rust, but is equivalent
        # because options don't need to be flattened in Python
        return bounds
```

[1] See new changes with `git diff f5bb719..a197d7e rust/src/transformations/make_stable_lazyframe/truncate/matching/mod.rs`

```
20
21      elif isinstance(predicate, BinaryExpr) and predicate.op == Operator.And:
22          left = match_truncation_predicate(predicate.left, identifier)
23          right = match_truncation_predicate(predicate.right, identifier)
24          if left is None or right is None:  #
25              return None
26          return left + right
27
28      elif isinstance(predicate, BinaryExpr):
29          left, right = predicate.left, predicate.right
30          if predicate.op == Operator.Lt:
31              over, threshold, offset = left, right, 0
32          elif predicate.op == Operator.LtEq:
33              over, threshold, offset = left, right, 1
34          elif predicate.op == Operator.Gt:
35              over, threshold, offset = right, left, 0
36          elif predicate.op == Operator.GtEq:
37              over, threshold, offset = right, left, 1
38          else:
39              return None
40
41          if not isinstance(over, Expr.Window): #
42              return None
43
44          threshold_value = literal_value_of(threshold, u32)
45          if threshold_value is None:
46              raise ValueError(
47                  f"literal value for truncation threshold ({threshold}) must be representable
     as a u32"
48              )
49
50          # account for distinction between gt and ge
51          threshold_value = threshold_value.inf_add(offset) #
52
53          num_groups = match_num_groups_predicate(
54              over.function, over.partition_by, identifier, threshold_value
55          )
56          per_group = match_per_group_predicate(
57              over.function, over.partition_by, identifier, threshold_value
58          )
59
60          if num_groups is None and per_group is None: #
61              raise ValueError(
62                  f"expected a predicate that limits per_group contributions (via int_range)
    or num_groups contributions (via rank). Found {over.function}"
63              )
64
65          return [num_groups or per_group] #
```

Postcondition

Theorem 1.1 (Postcondition)**.** For a given filter predicate and identifier expression, returns an error if the predicate contains a mis-specified truncation, none if the predicate is not a truncation, otherwise the per-identifier bounds on user contribution.

Proof. Truncation predicates are rooted by one of three expressions:

1. `AllHorizontal`, which is a vector-valued "and" expression. If all elements of the vector are truncation predicates, as checked on line 14, their intersection is also a truncation.
2. `BinaryExpr` where the operator is "and". If both operands are truncation predicates, as checked on line 24, their intersection is also a truncation.
3. `BinaryExpr` where the operator is a comparison. The logic for this case is more involved.

In the comparison case, the algorithm first matches through the comparison operator to identify what should be the "over" and "threshold" expressions. Since window expressions are not valid row-by-row functions, they are unambiguously truncation expressions. Therefore, if the "over" expression is a window expression, as checked on line 41, then any further failures to match the truncation predicates can now be raised as errors.

`threshold_value` on line 51 is resolved to the literal u32 upper bound on contributions.

By the postconditions of `match_num_groups_predicate`, and `match_per_group_predicate`, `num_groups` and `per_group` are optional bounds on the number of groups and row contributions per-group, respectively. If both are not defined, then the predicate is not a truncation, and an error is raised on line 60.

Otherwise, the matched bound is returned on line 65, satisfying the postcondition. □