

fn make_expr_datetime_component

Michael Shoemate

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

Proves soundness of `make_expr_datetime_component` in `mod.rs` at commit `f5bb719` (outdated¹).
`make_expr_datetime_component` returns a Transformation that extracts a component from a temporal data type.

1 Hoare Triple

Precondition

Compiler-verified

- Argument `input_domain` of type `ExprDomain`
- Argument `input_metric` of type `M`
- Generic `M` implements `OuterMetric`
 - `OuterMetric` defines an associated type `InnerMetric` that must implement `DatasetMetric`
- `(ExprDomain, M)` implements `MetricSpace`
- `Expr` implements `StableExpr<M, M>`

Caller-verified

None

Pseudocode

```
1 def make_expr_datetime_component(  
2     input_domain: ExprDomain,  
3     input_metric: M,  
4     expr: Expr,  
5 ) -> Transformation:  
6     match expr: #  
7         case Expr.Function(input=inputs, function=FunctionExpr.TemporalExpr(  
8             temporal_function)):  
9             pass  
10            case _:  
11                raise ValueError("expected datetime component expression")  
12            to_dtype, _ = match_datetime_component(temporal_function) #  
13
```

¹See new changes with `git diff f5bb719..46e01e0 rust/src/transformations/make_stable_expr/namespace_dt/expr_datetime_component/`
`rs`

```

14 # raises an error if there is not exactly one input
15 input, = inputs #
16
17 t_prior = input.make_stable(input_domain, input_metric) #
18 middle_domain, middle_metric = t_prior.output_space()
19
20 in_dtype = middle_domain.column.dtype
21 if in_dtype not in {DataType.Time, DataType.Datetime, DataType.Date}: #
22     raise ValueError("expected a temporal input type")
23
24 output_domain = middle_domain.clone() #
25 output_domain.column.set_dtype(to_dtype) #
26
27 def function(expr: Expr) -> Expr:
28     return Expr.Function(
29         input=[expr],
30         function=FunctionExpr.TemporalExpr(temporal_function),
31         options=FunctionOptions(
32             collect_groups=ApplyOptions.ElementWise,
33         ),
34     )
35
36 return t_prior >> Transformation.new( #
37     middle_domain,
38     output_domain,
39     Function.then_expr(function),
40     middle_metric,
41     middle_metric,
42     StabilityMap.new(lambda d_in: d_in),
43 )

```

Postcondition

Theorem 1.1. For every setting of the input parameters (`input_domain`, `input_metric`, `M`) to `make_expr_datetime_component` such that the given preconditions hold, `make_expr_datetime_component` 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`, `invoke(x)` and `invoke(x')` either both return the same error or neither return an error.
2. (Appropriate output domain). For every member x in `input_domain`, `function(x)` is in `output_domain` or raises a data-independent runtime error.
3. (Stability guarantee). For every pair of members x and 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 error, and `stability_map(d_in) = d_out`, then `function(x), function(x')` are d_{out} -close under `output_metric`.

2 Proof

Starting from line 6, `expr` is matched as if it were a temporal expression, or otherwise rejects the expression. All preconditions for `make_datetime_component` on line 12 are satisfied by the compiler, therefore by the postcondition `to_dtype` and `max_num_partitions` are the output data type and upper bound on the number of unique values in the output.

All preconditions for `make_stable` on line 17 are compiler-verified, therefore by the postcondition `t_prior` is a valid transformation. To prove that the output is a valid transformation, we must first prove that the transformation on line 36 is a valid transformation.

Proof of Data-Independent Errors. If the input data type does not include the component, then all possible choices of input dataset would fail, resulting in data-independent errors.

If the input data type does include the component, then Polars will never raise an error. \square

Proof of Output Domain. The output domain is the same as the input domain, but the active series has a new data type and margin metadata needs to be updated. Line 25 updates the series domain so that non-null elements are of type `to_type`, which by the postcondition of `match_datetime_component` on line 12 represents the type of outputs when `temporal_function` is applied to the data.

Other domain descriptors, like the nullity and name, remain unchanged. New null values cannot be introduced, because retrieval of a time component fails (in a data-independent way) if the component is not present in the type. The output domain now accounts for all transformations made to data in the input domain. \square

Proof of Stability. Datetime component retrieval is a 1-stable row-by-row transformation, as the component retrieval is applied independently to each and every row. See `make_row_by_row` for a proof of the stability of row-by-row functions. \square

Since it has been shown that both `t_prior` and the component transformation are valid transformations, then the preconditions for `make_chain_tt` are met (invoked via the right-shift operator shorthand), and therefore the output is a valid transformation.