fn make_expr_strptime

Michael Shoemate

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

Proves soundness of `make_expr_strptime` in `mod.rs` at commit f5bb719 (outdated[1]).
`make_expr_strptime` returns a Transformation that parses strings in a dataframe column into temporal types.

1 Hoare Triple

Precondition

Compiler-verified

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

Human-verified

None

Pseudocode

```
def make_expr_strptime(
    input_domain: WildExprDomain, input_metric: M, expr: Expr
) -> Transformation:
    match expr:  #
        case Expr.Function(
            inputs,
            function=FunctionExpr.StringExpr(
                StringFunction.Strptime(to_type, strptime_options)
            ),
        ):
            pass
        case _:
            raise ValueError("expected str.strptime expression")
    input, ambiguous = inputs
```

[1] See new changes with `git diff f5bb719..0c6a4fd rust/src/transformations/make_stable_expr/namespace_str/expr_strptime/mod.rs`

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

    if strptime_options.format is None:  #
        raise ValueError("format must be specified")

    if to_type == DataType.Time and not strptime_options.exact:
        raise ValueError("non-exact not implemented for Time data type")

    # never raise on error
    strptime_options.strict = False  #

    try:  #
        ambig_value = literal_value_of(ambiguous)
    except Exception:
        ambig_value = None
    ambiguous = lit(ambig_value if ambig_value in {"earliest" "latest"} else "null")

    output_domain = middle_domain.clone()
    series_domain = output_domain.column

    # check input and output types
    if series_domain.dtype() != DataType.String:  #
        raise ValueError("str.strptime input dtype must be String")

    if to_type not in {DataType.Time, DataType.Datetime, DataType.Date}:  #
        raise ValueError("str.strptime output dtype must be Time, Datetime or Date")

    # in Rust, this assigns to the series domain in output_domain
    series_domain.set_dtype(to_type)  #
    series_domain.nullable = True

    def function(expr: Expr) -> Expr:
        return expr.str.strptime(to_type, strptime_options, ambiguous)

    return t_prior >> Transformation.new(  #
        middle_domain,
        output_domain,
        Function.then_expr(function),
        middle_metric,
        middle_metric,
        StabilityMap.new(lambda d_in: d_in),
    )
```

Postcondition

Theorem 1.1. For every setting of the input parameters `(input_domain, input_metric, expr, M)` to `make_expr_strptime` such that the given preconditions hold, `make_expr_strptime` 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 4, `expr` is matched as if it were a strptime expression, or otherwise rejects the expression.

All preconditions for `make_stable` on line 16 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 51 is a valid transformation.

Proof. **Data-Independent Errors** Line 38 ensures that the input data is always a string type and line 41 ensures that the output data is always a temporal type, which are necessary preconditions for the strptime function in Polars. However, even if these checks were ignored when they would fail, then all possible choices of input dataset would fail, resulting in data-independent errors.

Line 26 disables raising an error when conversion fails. Conversion cannot be ambiguous because line 19 requires the format to be specified, but even if parsing were to be ambiguous, line 28 configures the parsing of ambiguous values to not raise.

Therefore, all errors that may be encountered while parsing result in null values, so no error can be raised by the strptime transformation. □

Proof. **Appropriate Output Domain** Line 45 constructs a series domain whose elements are of type `to_type` or null. Each string in the input data may either parse into `to_type` or fail and by 26 and 28 evaluate to null.

Therefore the output domain is the same as the input domain, but where the active column domain's data type is nullable values of type `to_type`. □

Proof. **Stability Guarantee** strptime can be considered a 1-stable row-by-row transformation, as the date parsing is applied independently to each and every row. See `make_row_by_row` for a proof of the stability of row-by-row functions. □

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