

fn make_row_by_row_fallible

Michael Shoemate

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

Proves soundness of `make_row_by_row_fallible` in `mod.rs` at commit `f5bb719` (outdated¹).

`make_row_by_row_fallible` returns a Transformation that applies a user-specified function to each record in the input dataset. The function is permitted to return a data-independent error.

Vetting History

- [Pull Request #562](#)

1 Hoare Triple

Precondition

- DI (input domain) is a type with trait `RowByRowDomain<D0>`. This trait provides a way to apply a map function to each record in the input dataset to retrieve a dataset that is a member of the output domain, of type D0. The trait further implies that `DatasetDomain` is also implemented for DI.
- D0 (output domain) is a type with trait `DatasetDomain`. `DatasetDomain` is used to define the type of the row domain.
- M (metric) is a type with trait `DatasetMetric`. `DatasetMetric` is used to restrict the set of valid metrics to those which measure distances between datasets.
- `MetricSpace` is implemented for (DI, M). Therefore M is a valid metric on DI.
- `MetricSpace` is implemented for (D0, M).
- `row_function` has no side-effects.
- If the input to `row_function` is a member of `input_domain`’s row domain, then the output is a member of `output_row_domain`, or a data-independent error.

Pseudocode

```
1 def make_row_by_row_fallible(  
2     input_domain: DI,  
3     input_metric: M,  
4     output_row_domain: D0,  
5     # a function from input domain's row type to output domain's row type  
6     row_function: Callable([[DI_RowDomain_Carrier], D0_RowDomain_Carrier])  
7 ) -> Transformation:  
8
```

¹See new changes with `git diff f5bb719..cabbe21 rust/src/transformations/manipulation/mod.rs`

```

9   # where .translate is defined by the RowByRowDomain trait
10  output_domain = input_domain.translate(output_row_domain)
11
12  def function(data: DI_Carrier) -> DO_Carrier:
13    # where .apply_rows is defined by the RowByRowDomain trait
14    return DI.apply_rows(data, row_function)
15
16  stability_map = new_stability_map_from_constant(1) #
17
18  return Transformation(
19    input_domain, output_domain, function,
20    input_metric, input_metric, stability_map)

```

Postcondition

Theorem 1.1. For every setting of the input parameters (`input_domain`, `input_metric`, `output_domain`, `row_function`, `DI`, `DO`, `M`) to `make_row_by_row` such that the given preconditions hold, `make_row_by_row` 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 Proofs

Proof. (Part 1 – appropriate output domain). By the definition of `RowByRowDomain`, `DI.apply_rows(data, row_function)` returns a dataset in `input_domain.translate(output_row_domain)`, if `row_function` is a mapping between `input_domain`'s row domain to `output_row_domain`. This is satisfied by the precondition on `row_function`. Thus, for all settings of input arguments, the function returns a dataset in the output domain. \square

Before proceeding with proving the validity of the stability map, we first provide a lemma.

Lemma 2.1. Let f denote the `row_function`. For any choice u, v of input arguments in the input domain, and any choice M for which `DatasetMetric` is implemented for, $d_M([f(u_1), f(u_2), \dots], [f(v_1), f(v_2), \dots]) \leq d_M([u_1, u_2, \dots], [v_1, v_2, \dots])$.

Proof. Assume WLOG that any source of randomness is fixed when f is computed on u vs v . Given this assumption, and the precondition that f has no side-effects, if $u_i = v_i$, then $f(u_i) = f(v_i)$. That is, the row function cannot increase the distance between corresponding rows in any adjacent dataset. On the other hand, it is possible for $f(u_i) = f(v_i)$, even if $u_i \neq v_i$. For example, if f is a constant function, then $f(u_i) = f(v_i)$ for all i . Therefore, by any of the metrics that `DatasetMetric` is implemented for, f can only make datasets more similar. \square

Proof. (Part 2 – stability map). Take any two elements u, v in the `input_domain` and any 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`. Assume u, v are `d_in`-close under `input_metric` and that $\text{stability_map}(\text{d_in}) \leq \text{d_out}$.

$$\begin{aligned}
d_M(\text{function}(u), \text{function}(v)) &= d_M([f(u_1), f(u_2), \dots], [f(v_1), f(v_2), \dots]) && \text{since D0 is a DatasetDomain} \\
&\leq d_M([u_1, u_2, \dots], [v_1, v_2, \dots]) && \text{by 2.1} \\
&= d_M(u, v) && \text{since DI is a DatasetDomain} \\
&= \text{d_in} && \text{by the first assumption} \\
&\leq \text{T0.inf_cast}(\text{d_in}) && \text{by InfCast} \\
&\leq \text{T0.one().inf_mul}(\text{T0.inf_cast}(\text{d_in})) && \text{by InfMul} \\
&= \text{stability_map}(\text{d_in}) && \text{by pseudocode line 16} \\
&\leq \text{d_out} && \text{by the second assumption}
\end{aligned}$$

It is shown that $\text{function}(u), \text{function}(v)$ are `d_out`-close under `output_metric`. □