

# 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Proves soundness of **make\_row\_by\_row\_fallible** in **mod.rs** at commit **f5bb719** (outdated<sup>1</sup>).

**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
```

<sup>1</sup>See new changes with `git diff f5bb719..80c1d6f1 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 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) ≤ 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 `stability_map(d_in) ≤ d_out`.

$$\begin{aligned}
d_M(\text{function}(u), \text{function}(v)) &= d_M([f(u_1), f(u_2), \dots], [f(v_1), f(v_2), \dots]) && \text{since } D0 \text{ is a } \text{DatasetDomain} \\
&\leq d_M([u_1, u_2, \dots], [v_1, v_2, \dots]) && \text{by } 2.1 \\
&= d_M(u, v) && \text{since } DI \text{ is a } \text{DatasetDomain} \\
&= d_{in} && \text{by the first assumption} \\
&\leq T0.\text{inf\_cast}(d_{in}) && \text{by } \text{InfCast} \\
&\leq T0.\text{one}().\text{inf\_mul}(T0.\text{inf\_cast}(d_{in})) && \text{by } \text{InfMul} \\
&= \text{stability\_map}(d_{in}) && \text{by pseudocode line } 16 \\
&\leq d_{out} && \text{by the second assumption}
\end{aligned}$$

It is shown that  $\text{function}(u), \text{function}(v)$  are  $d_{out}$ -close under  $\text{output\_metric}$ . □