fn make_row_by_row_fallible

Michael Shoemate

Proves soundness of make_row_by_row_fallible in mod.rs at commit f5bb719 (outdated[1]).

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

Compiler-verified

- Generic DI (input domain) is a type with trait RowByRowDomain<DO>.
- Generic DO (output domain) is a type with trait DatasetDomain. DatasetDomain is used to define the type of the row domain.
- Generic 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 (DO, M).
- Argument input_domain is of type DI
- Argument input_metric is of type M
- Argument output_row_domain is of type DI::ElementDomain, as defined by DatasetDomain.
- Argument row_function is an immutable thread-safe function taking in a value of the carrier type of the element domain associated with input_domain, and returning a value of the carrier type of output_row_domain or an error.

User-verified

- row_function has no side-effects.
- If the input to row_function is a member of input_domain's element domain, then the output is a member of output_row_domain, or a data-independent error.

[1] See new changes with git diff f5bb719..52d86bbb rust/src/transformations/manipulation/mod.rs

Pseudocode

```
def make_row_by_row_fallible(
    input_domain: DI,
    input_metric: M,
    output_row_domain: DO_RowDomain,
    # a function from input domain's row type to output domain's row type
    row_function: Callable([[DI_RowDomain_Carrier], DO_RowDomain_Carrier])
) -> Transformation:

    # where .translate is defined by the RowByRowDomain trait
    output_domain = input_domain.translate(output_row_domain)

    def function(data: DI_Carrier) -> DO_Carrier:
        # where .apply_rows is defined by the RowByRowDomain trait
        return DI.apply_rows(data, row_function)

    stability_map = new_stability_map_from_constant(1) #

    return Transformation(
        input_domain, output_domain, function,
        input_metric, input_metric, stability_map)
```

Postcondition

Theorem 1.1. For every setting of the input parameters `(input_domain, input_metric, output_row_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`) $\le$ `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. □

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 x, x' of input arguments in the input domain, and any choice `M` for which `DatasetMetric` is implemented for, $d_M([f(x_1), f(x_2), ...], [f(x'_1), f(x'_2), ...]) \le d_M([x_1, x_2, ...], [x'_1, x'_2, ...])$.

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 $x_i = x'_i$, then $f(x_i) = f(x'_i)$. That is, the row function cannot increase the distance between corresponding rows in any adjacent dataset. On the other hand, it is possible for $f(x_i) = f(x'_i)$, even if $x_i \ne x'_i$. For example, if f is a constant function, then

$f(x_i) = f(x'_i)$ for all i. Therefore, by any of the metrics that `DatasetMetric` is implemented for, f can only make datasets more similar. ☐

Proof. **(Part 2 – stability map).** Take any two elements x, x' 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`) $\leq$ `d_out`.

$$
\begin{aligned}
d_M(\texttt{function}(x), \texttt{function}(x')) &= d_M([f(x_1), f(x_2), ...], [f(x'_1), f(x'_2), ...]) && \text{since } \texttt{DO} \text{ is a } \texttt{DatasetDomain} \\
&\leq d_M([x_1, x_2, ...], [x'_1, x'_2, ...]) && \texttt{by } 2.1 \\
&= d_M(x, x') && \text{since } \texttt{DI} \text{ is a } \texttt{DatasetDomain} \\
&= \texttt{d\_in} && \text{by the first assumption} \\
&\leq \texttt{TO.inf\_cast(d\_in)} && \text{by } \texttt{InfCast} \\
&\leq \texttt{TO.one().inf\_mul(TO.inf\_cast(d\_in))} && \text{by } \texttt{InfMul} \\
&= \texttt{stability\_map(d\_in)} && \text{by pseudocode line } 16 \\
&\leq \texttt{d\_out} && \text{by the second assumption}
\end{aligned}
$$

It is shown that `function`(x), `function`(x') are `d_out`-close under `output_metric`. ☐