fn make_clamp

Sílvia Casacuberta

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

Proves soundness of `fn make_clamp` in mod.rs at commit 0db9c6036 (outdated[1]).

Vetting History

- Pull Request #512

1 Hoare Triple

Precondition

To ensure the correctness of the output, we require the following preconditions:

- Type `TA` must have trait `TotalOrd`.

- Type `M` must have trait `DatasetMetric`.

Pseudocode

```python
def make_clamp(
    input_domain: VectorDomain[AtomDomain[TA]],
    input_metric: M,
    bounds: (TA, TA)
):
    input_domain.element_domain.assert_non_null()

    # clone to make it explicit that we are not mutating the input domain
    output_row_domain = input_domain.element_domain.clone()
    output_row_domain.bounds = Bounds.new_closed(bounds)

    def clamper(value: TA) -> TA:
        return value.total_clamp(bounds[0], bounds[1])

    return make_row_by_row_fallible(
        input_domain,
        input_metric,
        output_row_domain,
        clamper
    )
```

---

[1]See new changes with `git diff 0db9c6036..8840a59b rust/src/transformations/clamp/mod.rs`

Postconditions

For every setting of the input parameters (`input_domain, input_metric, bounds`) to `make_clamp` such that the given preconditions hold, `make_clamp` 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 v in `input_domain`, $\text{function}(v)$ is in `output_domain` or raises a data-independent runtime exception.

2. (Stability guarantee). For every pair of elements u, v in `input_domain` and for every pair $(\texttt{d\_in}, \texttt{d\_out})$, where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_metric`, if u, v are `d_in`-close under `input_metric` and $\text{stability\_map}(\texttt{d\_in}) \le \texttt{d\_out}$, then $\text{function}(u), \text{function}(v)$ are `d_out`-close under `output_metric`.

2   Proof

Lemma 2.1. The invocation of `make_row_by_row_fallible` (line 15) satisfies its preconditions.

Proof. The preconditions of `make_clamp` and pseudocode definition (line 5) ensure that the type preconditions of `make_row_by_row_fallible` are satisfied. The remaining preconditions of `make_row_by_row_fallible` are:

- `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`.

The first precondition is satisfied by the definition of `clamper` (line 12) in the pseudocode.

For the second precondition, assume the input is a member of `input_domain`'s row domain. Therefore, by 6, the input is non-null. In addition, since `Bounds.new_closed` did not raise an exception, then by the definition of `Bounds.new_closed`, the bounds are non-null. Thus, by the definition of `TotalOrd`, the preconditions of `total_clamp` are satisfied, so the output is within the bounds. Therefore, the output is a member of `output_row_domain`. □

We now prove the postcondition of `make_clamp`.

Proof. By 2, the preconditions of `make_row_by_row_fallible` are satisfied. Thus, by the definition of `make_row_by_row_fallible`, the output is a valid transformation. □