fn make_clamp

Sílvia Casacuberta

Proves soundness of `fn make_clamp` in `mod.rs` at commit 0db9c6036 (outdated[1]).

1 Hoare Triple

Precondition

Compiler-verified

- Argument `input_domain` is of type `VectorDomain<AtomDomain<TA>>`.
- Argument `input_metric` is of type `M`.
- Argument `bounds` is of type `(T, T)`.
- Generic `TA` must implement `Number`.
- Generic `M` must have trait `DatasetMetric`.

User-verified

None

Pseudocode

```python
def make_clamp(
    input_domain: VectorDomain[AtomDomain[TA]],
    input_metric: M,
    bounds: tuple[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..feaaeb8e rust/src/transformations/clamp/mod.rs`

Postcondition

Theorem 1.1. For every setting of the input parameters `(input_domain, input_metric, bounds, TA, M)` 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. (Data-independent runtime error). For every pair of elements x, x' in `input_domain`, `function`(x) returns an error if and only if `function`(x').

2. (Appropriate output domain). For every element x in `input_domain`, `function`(x) is in `output_domain` or raises a data-independent runtime exception.

3. (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`) $\leq$ `d_out`, then `function`(x), `function`(x') 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. 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.

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:

- Errors from `row_function` are data-independent.

- `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`.

By the definition of `ProductOrd.total_clamp`, `total_clamp` won't raise because `arg` and both bounds are not null, and because the lower bound is less than the upper bound by the postcondition of `Bounds.new_closed` on line 10. Since `total_clamp` is the only potential source of errors, and it cannot throw an error, there are no runtime errors in `row_function`, satisfying the first remaining precondition.

The second remaining precondition is satisfied by the definition of `clamper` (line 12) in the pseudocode. The function uses `ProductOrd.total_clamp`, and by its definition there are no side-effects.

For the last remaining precondition, by the postcondition/proof definition of `ProductOrd.total_clamp`, the outcome is within the bounds, which satisfies the additional descriptor bound in the output domain. Therefore, the output is a member of `output_row_domain`. □

We now prove theorem 1.1.

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. □