fn make_count

Sílvia Casacuberta, Grace Tian, Connor Wagaman

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

Proves soundness of `make_count` in `mod.rs` at commit f5bb719 (outdated[1]).

`make_count` returns a Transformation that computes a count of the number of records in a vector. The length of the vector, of type `usize`, is exactly casted to a user specified output type `TO`. If the length is too large to be represented exactly by `TO`, the cast saturates at the maximum value of type `TO`.

Vetting History

- Pull Request #513

1 Hoare Triple

Precondition

- `TIA` (atomic input type) is a type with trait `Primitive`. `Primitive` implies `TIA` has the trait bound:
  - `CheckNull` so that `TIA` is a valid atomic type for `AtomDomain`
- `TO` (output type) is a type with trait `Number`. `Number` further implies `TO` has the trait bounds:
  - `InfSub` so that the output domain is compatible with the output metric
  - `CheckNull` so that `TO` is a valid atomic type for `AtomDomain`
  - `ExactIntCast` for casting a vector length index of type `usize` to `TO`. `ExactIntCast` further implies `TO` has the trait bound:
    * `ExactIntBounds`, which gives the `MAX_CONSECUTIVE` value of type `TO`
  - `One` provides a way to retrieve `TO`'s representation of 1
  - `DistanceConstant` to satisfy the preconditions of `new_stability_map_from_constant`

Pseudocode

```python
def make_count(
    input_domain: VectorDomain[AtomDomain[TIA]],
    input_metric: SymmetricDistance
):
    output_domain = AtomDomain(TO) #

    def function(data: Vec[TIA]) -> TO: #
        size = input_domain.size(data) #
        try: #
            return TO.exact_int_cast(size) #
```

[1] See new changes with `git diff f5bb719..bffbbf2 rust/src/transformations/count/mod.rs`

```
11         except FailedCast:
12             return TO.MAX_CONSECUTIVE #
13 
14     output_metric = AbsoluteDistance(TO)
15 
16     stability_map = new_stability_map_from_constant(TO.one()) #
17 
18     return Transformation(
19         input_domain, output_domain, function,
20         input_metric, output_metric, stability_map)
```

Postcondition

Theorem 1.1. For every setting of the input parameters `(TIA, TO)` to `make_count` such that the given preconditions hold, `make_count` 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).** The `output_domain` is `AtomDomain(TO)`, so it is sufficient to show that `function` always returns non-null values of type `TO`. By the definition of the `ExactIntCast` trait, `TO.exact_int_cast` always returns a non-null value of type `TO` or raises an exception. If an exception is raised, the function returns `TO.MAXIMUM_CONSECUTIVE`, which is also a non-null value of type `TO`. Thus, in all cases, the function (from line 9) returns a non-null value of type `TO`. □

Before proceeding with proving the validity of the stability map, we provide a couple lemmas.

Lemma 2.1. $|\texttt{function}(u) - \texttt{function}(v)| \leq |\texttt{len(u)} - \texttt{len(v)}|$, where `len` is an alias for `input_domain.size`.

Proof. By `CollectionDomain`, we know `size` on line 8 is of type `usize`, so it is non-negative and integral. Therefore, by the definition of `ExactIntCast`, the invocation of `TO.exact_int_cast` on line 10 can only fail if the argument is greater than `TO.MAX_CONSECUTIVE`. In this case, the value is replaced with `TO.MAX_CONSECUTIVE`. Therefore, $\texttt{function}(u) = min(\texttt{len(u)}, c)$, where $c = $ `TO.MAX_CONSECUTIVE`. We use this equality to prove the lemma:

$$\begin{aligned} |\texttt{function}(u) - \texttt{function}(v)| &= |min(\texttt{len(u)}, c) - min(\texttt{len(v)}, c)| \\ &\leq |\texttt{len(u)} - \texttt{len(v)}| && \text{since clamping is stable} \end{aligned}$$

□

Lemma 2.2. For vector v with each element $\ell \in v$ drawn from domain $\mathcal{X}$, `len(v)` $= \sum_{z \in \mathcal{X}} h_v(z)$.

Proof. Every element $\ell \in v$ is drawn from domain $\mathcal{X}$, so summing over all $z \in \mathcal{X}$ will sum over every element $\ell \in x$. Recall that the definition of `SymmetricDistance` states that $h_v(z)$ will return the number of occurrences of value z in vector v. Therefore, $\sum_{z \in \mathcal{X}} h_v(z)$ is the sum of the number of occurrences of each unique value; this is equivalent to the total number of items in the vector.

Since `CollectionDomain` is implemented for `VectorDomain<AtomDomain<TIA»`, we depend on the correctness of the implementation Conditioned on the correctness of the implementation of `CollectionDomain` for `VectorDomain<AtomDomain<TIA»`, the variable `size` is of type `usize` containing the number of elements in `arg`. Therefore, $\sum_{z \in \mathcal{X}} h_v(z)$ is equivalent to `size`. □

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)` $\le$ `d_out`. These assumptions are used to establish the following inequality:

$$\begin{aligned}
|\texttt{function}(u) - \texttt{function}(v)| &\le |\texttt{len(u)} - \texttt{len(v)}| && \text{by 2.1} \\
&= |\sum_{z \in \mathcal{X}} h_{\texttt{u}}(z) - \sum_{z \in \mathcal{X}} h_{\texttt{v}}(z)| && \text{by 2.2} \\
&= |\sum_{z \in \mathcal{X}} (h_{\texttt{u}}(z) - h_{\texttt{v}}(z))\,| && \text{by algebra} \\
&\le \sum_{z \in \mathcal{X}} |h_{\texttt{u}}(z) - h_{\texttt{v}}(z)| && \text{by triangle inequality} \\
&= d_{Sym}(u, v) && \text{by SymmetricDistance} \\
&\le \texttt{d_in} && \text{by the first assumption} \\
&\le \texttt{TO.inf_cast}(\texttt{d_in}) && \text{by InfCast} \\
&\le \texttt{TO.one().inf_mul(TO.inf_cast(d_in))} && \text{by InfMul} \\
&= \texttt{stability_map}(\texttt{d_in}) && \text{by pseudocode line 16} \\
&\le \texttt{d_out} && \text{by the second assumption}
\end{aligned}$$

It is shown that `function`(u), `function`(v) are `d_out`-close under `output_metric`. □