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Proves soundness of `make_count` in `mod.rs` at commit f5bb719 (outdated[1]).

1 Vetting history

- Pull Request #513

2 Pseudocode

Preconditions

- `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 `VectorDomain`
- `TO` (output type) is a type with trait `Number`. `Number` further implies `TO` has the trait bounds:
  - `CheckNull` so that `TO` is a valid atomic type for `AllDomain`
  - `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`

Implementation

```
def make_count():
    input_domain = VectorDomain(AllDomain(TIA))
    output_domain = AllDomain(TO)

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

    input_metric = SymmetricDistance()
    output_metric = AbsoluteDistance(TO)

```

[1] See new changes with `git diff f5bb719..453bce686 rust/src/transformations/count/mod.rs`

```
15     stability_map = new_stability_map_from_constant(TO.one())
16
17     return Transformation(
18         input_domain, output_domain, function,
19         input_metric, output_metric, stability_map)
```

Postcondition

`make_count` raises an exception or returns a valid `Transformation`.

3 Proof

Theorem 3.1. For every setting of the input parameters `(TIA, TO)` to `make_count` such that the given preconditions hold, `make_count` 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`, `function`(v) is in `output_domain` or raises a data-independent runtime exception.

2. (Domain-metric compatibility). The domain `input_domain` matches one of the possible domains listed in the definition of `input_metric`, and likewise `output_domain` matches one of the possible domains listed in the definition of `output_metric`.

3. (Stability guarantee). For every pair of elements u, v 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 u, v are `d_in`-close under `input_metric` and `stability_map(d_in)` $\leq$ `d_out`, then `function`(u), `function`(v) are `d_out`-close under `output_metric`.

Proof. **(Part 1 – appropriate output domain).** The `output_domain` is `AllDomain(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 7) returns a non-null value of type `TO`. □

Proof. **(Part 2 – domain-metric compatibility).** Our `input_metric` of `SymmetricDistance` is compatible with any domain of the form `VectorDomain(inner_domain)`, and our `input_domain` of `VectorDomain(AllDomain(TIA))` is of this form. Therefore our `input_domain` and `input_metric` are compatible.

Our `output_metric` of `AbsoluteDistance` is compatible with any domain of the form `AllDomain(T)` where `T` has the trait `InfSub`, and our `output_domain` of `AllDomain(TO)` is of this form and `TO` has the necessary trait. Therefore our `input_domain` and `input_metric` are compatible. □

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

Lemma 3.2. $|$`function`$(u) -$ `function`$(v)| \leq |$`len(u)` $-$ `len(v)`$|$, where `len` is an alias for `input_domain.size`.

Proof. In line 7, we know the argument to `TO.exact_int_cast` is non-negative and integral. Therefore, by the definition of `ExactIntCast`, the invocation of `TO.exact_int_cast` can only fail if the argument is greater than `TO.MAX_CONSECUTIVE`. In this case, the value is replaced with `TO.MAX_CONSECUTIVE`. Therefore, `function`$(u) = min($`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 3.3. For vector v with each element $\ell \in v$ drawn from domain $\mathcal{X}$, $\texttt{len(v)} = \sum_{z \in \mathcal{X}} h_v(z)$.

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<AllDomain<TIA»`, we depend on the correctness of the implementation Conditioned on the correctness of the implementation of `CollectionDomain` for `VectorDomain<AllDomain<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 3 – 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)` $\leq$ `d_out`. These assumptions are used to establish the following inequality:

$$
\begin{aligned}
|\texttt{function}(u) - \texttt{function}(v)| &\leq |\texttt{len(u)} - \texttt{len(v)}| && \text{by 3.2} \\
&= |\sum_{z \in \mathcal{X}} h_{\texttt{u}}(z) - \sum_{z \in \mathcal{X}} h_{\texttt{v}}(z)| && \text{by 3.3} \\
&= |\sum_{z \in \mathcal{X}} (h_{\texttt{u}}(z) - h_{\texttt{v}}(z))\,| && \text{by algebra} \\
&\leq \sum_{z \in \mathcal{X}} |h_{\texttt{u}}(z) - h_{\texttt{v}}(z)| && \text{by triangle inequality} \\
&= d_{Sym}(u, v) && \text{by SymmetricDistance} \\
&\leq \texttt{d\_in} && \text{by the first assumption} \\
&\leq \texttt{TO.inf\_cast(d\_in)} && \text{by InfCast} \\
&\leq \texttt{TO.one().inf\_mul(TO.inf\_cast(d\_in))} && \text{by InfMul} \\
&= \texttt{stability\_map(d\_in)} && \text{by pseudocode line 15} \\
&\leq \texttt{d\_out} && \text{by the second assumption}
\end{aligned}
$$

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