

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** (outdated¹).

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 **T0**. If the length is too large to be represented exactly by **T0**, the cast saturates at the maximum value of type **T0**.

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**
- T0 (output type) is a type with trait **Number**. **Number** further implies T0 has the trait bounds:
 - **InfSub** so that the output domain is compatible with the output metric
 - **CheckNull** so that T0 is a valid atomic type for **AtomDomain**
 - **ExactIntCast** for casting a vector length index of type **usize** to T0. **ExactIntCast** further implies T0 has the trait bound:
 - * **ExactIntBounds**, which gives the **MAX_CONSECUTIVE** value of type T0
 - **One** provides a way to retrieve T0’s representation of 1
 - **DistanceConstant** to satisfy the preconditions of **new_stability_map_from_constant**

Pseudocode

```
1 def make_count(  
2     input_domain: VectorDomain[AtomDomain[TIA]],  
3     input_metric: SymmetricDistance  
4 ):  
5     output_domain = AtomDomain(T0) #  
6  
7     def function(data: Vec[TIA]) -> T0: #  
8         size = input_domain.size(data) #  
9         try: #  
10             return T0.exact_int_cast(size) #
```

¹See new changes with `git diff f5bb719..e4fb7f4 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`. \square

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

Lemma 2.1. $|\text{function}(u) - \text{function}(v)| \leq |\text{len}(u) - \text{len}(v)|$, 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, $\text{function}(u) = \min(\text{len}(u), c)$, where $c = \text{TO.MAX_CONSECUTIVE}$. We use this equality to prove the lemma:

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

\square

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

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 $\ell \in v$. 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`. \square

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) ≤ d_out`. These assumptions are used to establish the following inequality:

$$\begin{aligned}
|\text{function}(u) - \text{function}(v)| &\leq |\text{len}(u) - \text{len}(v)| && \text{by 2.1} \\
&= \left| \sum_{z \in \mathcal{X}} h_u(z) - \sum_{z \in \mathcal{X}} h_v(z) \right| && \text{by 2.2} \\
&= \left| \sum_{z \in \mathcal{X}} (h_u(z) - h_v(z)) \right| && \text{by algebra} \\
&\leq \sum_{z \in \mathcal{X}} |h_u(z) - h_v(z)| && \text{by triangle inequality} \\
&= d_{Sym}(u, v) && \text{by SymmetricDistance} \\
&\leq d_in && \text{by the first assumption} \\
&\leq T0.\text{inf_cast}(d_in) && \text{by InfCast} \\
&\leq T0.\text{one}().\text{inf_mul}(T0.\text{inf_cast}(d_in)) && \text{by InfMul} \\
&= \text{stability_map}(d_in) && \text{by pseudocode line 16} \\
&\leq d_out && \text{by the second assumption}
\end{aligned}$$

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