

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     output_domain = AtomDomain(T0)
4
5     def function(data: Vec[TIA]) -> T0:
6         size = input_domain.size(data)
7         try:
8             return T0.exact_int_cast(size)
9         except FailedCast:
10            return T0.MAX_CONSECUTIVE
```

¹See new changes with `git diff f5bb719..9dd7ab54 rust/src/transformations/count/mod.rs`

```

11
12     input_metric = SymmetricDistance()
13     output_metric = AbsoluteDistance(T0)
14
15     stability_map = new_stability_map_from_constant(T0.one())
16
17     return Transformation(
18         input_domain, output_domain, function,
19         input_metric, output_metric, stability_map)

```

Postcondition

For every setting of the input parameters (TIA, T0) 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) ≤ d_out`, then `function(u), function(v)` are d_{out} -close under `output_metric`.

2 Proofs

Proof. (Part 1 – appropriate output domain). The `output_domain` is `AtomDomain(T0)`, so it is sufficient to show that `function` always returns non-null values of type `T0`. By the definition of the `ExactIntCast` trait, `T0.exact_int_cast` always returns a non-null value of type `T0` or raises an exception. If an exception is raised, the function returns `T0.MAXIMUM_CONSECUTIVE`, which is also a non-null value of type `T0`. Thus, in all cases, the function (from line 7) returns a non-null value of type `T0`. \square

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(AtomDomain(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 `AtomDomain(T)` where `T` has the trait `InfSub`, and our `output_domain` of `AtomDomain(T0)` is of this form and `T0` has the necessary trait. Therefore our `input_domain` and `input_metric` are compatible. \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 6 is of type `usize`, so it is non-negative and integral. Therefore, by the definition of `ExactIntCast`, the invocation of `T0.exact_int_cast` on line 8 can only fail if the argument is greater than `T0.MAX_CONSECUTIVE`. In this case, the value is replaced with `T0.MAX_CONSECUTIVE`. Therefore, $\text{function}(u) = \min(\text{len}(u), c)$, where $c = \text{T0.MAX_CONSECUTIVE}$. We use this equality to prove the lemma:

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

□

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**. □

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 $\text{stability_map}(d_in) \leq d_out$. These assumptions are used to establish the following inequality:

$$\begin{aligned}
|\text{function}(u) - \text{function}(v)| &\leq |\text{len}(u) - \text{len}(v)| && \text{by 2.1} \\
&= \left| \sum_{z \in \mathcal{X}} h_u(z) - \sum_{z \in \mathcal{X}} h_v(z) \right| && \text{by 2.2} \\
&= \left| \sum_{z \in \mathcal{X}} (h_u(z) - h_v(z)) \right| && \text{by algebra} \\
&\leq \sum_{z \in \mathcal{X}} |h_u(z) - h_v(z)| && \text{by triangle inequality} \\
&= d_{\text{Sym}}(u, v) && \text{by SymmetricDistance} \\
&\leq d_in && \text{by the first assumption} \\
&\leq T0.\text{inf_cast}(d_in) && \text{by InfCast} \\
&\leq T0.\text{one}().\text{inf_mul}(T0.\text{inf_cast}(d_in)) && \text{by InfMul} \\
&= \text{stability_map}(d_in) && \text{by pseudocode line 15} \\
&\leq d_out && \text{by the second assumption}
\end{aligned}$$

It is shown that $\text{function}(u), \text{function}(v)$ are **d_out**-close under **output_metric**. □