

# fn make\_count

Sílvia Casacuberta, Grace Tian, Connor Wagaman

Proves soundness of `make_count` in `mod.rs` at commit `f5bb719` (outdated<sup>1</sup>).

`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`.

## 1 Hoare Triple

### Precondition

#### Compiler-verified

- Generic TIA (atomic input type) is a type with trait `Primitive`.
- Generic T0 (output type) is a type with trait `Number`.
- Argument `input_domain` is of type `VectorDomain<AtomDomain<TIA>>`.
- Argument `input_metric` is of type `SymmetricDistance`.

#### Caller-verified

None

### Pseudocode

```
1 def make_count(  
2     input_domain: VectorDomain[AtomDomain[TIA]],  
3     input_metric: SymmetricDistance  
4 ):  
5     output_domain = AtomDomain.default(T0) #  
6  
7     def function(arg: Vec[TIA]) -> T0: #  
8         size = arg.len() #  
9         try: #  
10            return T0.exact_int_cast(size) #  
11        except FailedCast:  
12            return T0.MAX_CONSECUTIVE #  
13  
14     output_metric = AbsoluteDistance(T0)  
15  
16     stability_map = StabilityMap.new_from_constant(T0.one()) #  
17  
18     return Transformation(  
19         input_domain, output_domain, function,  
20         input_metric, output_metric, stability_map)
```

---

<sup>1</sup>See new changes with `git diff f5bb719..113a1afb rust/src/transformations/count/mod.rs`

## Postcondition

**Theorem 1.1.** For every setting of the input parameters (`input_domain`, `input_metric`, `TIA`, `T0`) 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(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 9) returns a non-null value of type `T0`.  $\square$

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

**Lemma 2.1.**  $|\text{function}(x) - \text{function}(x')| \leq |\text{len}(x) - \text{len}(x')|$ , where `len` is an alias for `input_domain.size`.

*Proof.* As `arg` has type `Vec<TIA>`, it supports the Rust standard library function `len` that returns the number of elements in the `arg` as type `usize` on line 8. By the definition of `ExactIntCast`, the invocation of `T0.exact_int_cast` on line 10 can only fail if the argument is greater than `T0.MAX_CONSECUTIVE`. In this case, the value is replaced with `T0.MAX_CONSECUTIVE`. Therefore,  $\text{function}(x) = \min(\text{len}(x), c)$ , where  $c = \text{T0.MAX\_CONSECUTIVE}$ . We use this equality to prove the lemma:

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

$\square$

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

*Proof.* Every element  $\ell \in x$  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  $h_x(z)$  will return the number of occurrences of value  $z$  in vector  $x$ . Therefore,  $\sum_{z \in \mathcal{X}} h_x(z)$  is the sum of the number of occurrences of each unique value; this is equivalent to the total number of items in the vector.

By the postcondition of `Vec.len` in the Rust standard library, the variable `size` is of type `usize` containing the number of elements in `arg`. Therefore,  $\sum_{z \in \mathcal{X}} h_x(z)$  is equivalent to `size`.  $\square$

*Proof. (Part 2 – stability map).* Take any two elements  $x, x'$  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  $x, x'$  are `d_in`-close under `input_metric` and that `stability_map(d_in) ≤ d_out`. These assumptions are used to establish the following inequality:

$$\begin{aligned}
|\text{function}(x) - \text{function}(x')| &\leq |\text{len}(x) - \text{len}(x')| && \text{by 2.1} \\
&= \left| \sum_{z \in \mathcal{X}} h_x(z) - \sum_{z \in \mathcal{X}} h_{x'}(z) \right| && \text{by 2.2} \\
&= \left| \sum_{z \in \mathcal{X}} (h_x(z) - h_{x'}(z)) \right| && \text{by algebra} \\
&\leq \sum_{z \in \mathcal{X}} |h_x(z) - h_{x'}(z)| && \text{by triangle inequality} \\
&= d_{Sym}(x, x') && \text{by SymmetricDistance} \\
&\leq d\_in && \text{by the first assumption} \\
&\leq T0.\text{inf\_cast}(d\_in) && \text{by InfCast} \\
&\leq T0.\text{one}().\text{inf\_mul}(T0.\text{inf\_cast}(d\_in)) && \text{by InfMul} \\
&= \text{stability\_map}(d\_in) && \text{by line 16, see StabilityMap} \\
&\leq d\_out && \text{by the second assumption}
\end{aligned}$$

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