

# fn make\_int\_to\_bigint\_threshold

Michael Shoemate

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

Proves soundness of the implementation of `make_int_to_bigint_threshold` in `mod.rs` at commit `f5bb719` (outdated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

## 1 Hoare Triple

### Precondition

#### Compiler-Verified

- Generic TK implements trait `Hashable`
- Generic TV implements trait `Integer` and `SaturatingCast<IBig>`
- Const-generic P is of type `usize`
- Generic QI implements trait `Number`

#### User-Verified

None

### Pseudocode

```
1 def make_int_to_bigint_threshold(  
2     input_space: tuple[  
3         MapDomain[AtomDomain[TK], AtomDomain[TV]], LOPInfDistance[P, AbsoluteDistance[QI]]  
4     ],  
5 ) -> Transformation[  
6     MapDomain[AtomDomain[TK], AtomDomain[TV]],  
7     MapDomain[AtomDomain[TK], AtomDomain[IBig]],  
8     LOPInfDistance[P, AbsoluteDistance[QI]],  
9     LOPInfDistance[P, AbsoluteDistance[RBig]],  
10 ]:  
11     input_domain, input_metric = input_space  
12  
13     def stability_map(d_in):  
14         lo, lp, li = d_in  
15         lp = UBig.try_from(lp)  
16         li = UBig.try_from(li)  
17         return lo, lp, li  
18  
19     return Transformation.new(  
20         input_domain,
```

---

<sup>1</sup>See new changes with `git diff f5bb719..1b82039 rust/src/measurements/noise_threshold/nature/integer/mod.rs`

```

21     MapDomain( #
22         key_domain=input_domain.key_domain,
23         value_domain=AtomDomain.default(IBig),
24     ),
25     Function.new(lambda x: {k: IBig.from_(v) for k, v in x.items()}), #
26     input_metric,
27     LOPI.default(),
28     StabilityMap.new_fallible(stability_map),
29 )

```

## Postcondition

### Theorem 1.1.

**Theorem 1.2.** For every setting of the input parameters (`input_space`, `TK`, `TV`, `P`, `QI`) to `make_int_to_bigint_threshold` such that the given preconditions hold, `make_int_to_bigint_threshold` 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`.

*Proof.* By the definition of the function on line 25, and since `IBig.from` is infallible, the function is infallible, meaning that the function cannot raise data-dependent errors. The function also always returns a hashmap whose keys are un-changed, and values are `IBigs`, meaning the output of the function is always a member of the output domain, as defined on line 21. Finally, the function is 1-stable, because the real values of the numbers remain un-changed, meaning the distance between adjacent inputs always remains the same, satisfying the stability property.  $\square$