`fn compute_score`

Michael Shoemate

February 20, 2025

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

Proves soundness of `compute_score` in `mod.rs` at commit f5bb719 (outdated[1]). `compute_score` returns a score for each candidate passed in, where the score is the distance between the candidate and the ideal alpha-quantile.

Vetting History

- Pull Request #456

1 Hoare Triple

Precondition

- `TIA` (input atom type) is a type with trait `PartialOrd`.
- `x` is non-null
- `candidates` is strictly increasing
- `alpha_numer / alpha_denom` is in $[0, 1]$
- `size_limit * alpha_denom` does not overflow

Function

```
def compute_score(
    x: list[TIA],
    candidates: list[TIA],
    alpha_num: usize,
    alpha_den: usize,
    size_limit: usize
) -> list[usize]:

    x = list(sorted(x))

    num_lt = [0] * len(candidates)
    num_eq = [0] * len(candidates)

    count_lt_eq_recursive(
        num_lt, # mutated in-place
        num_eq, # mutated in-place
```

[1] See new changes with `git diff f5bb719..0fff9eba rust/src/transformations/quantile_score_candidates/mod.rs`

```
        edges=candidates,
        x=x,
        x_start_idx=0)

    def score(lt, eq):
        return abs_diff(
            alpha_den * min(lt, size_limit),
            alpha_num * min(len(x) - eq, size_limit))

    return [score(lt, eq) for lt, eq in zip(num_lt, num_eq)]
```

Postcondition

Each element in the return value corresponds to the score of the candidate at the same index in `candidates`:

$$\text{compute\_score}(X, c, \alpha_{num}, \alpha_{den}, l) = |\alpha_{den} \cdot \min(\#(X < c), l), \alpha_{num} \cdot \min(|X| - \#(X = c), l)| \tag{1}$$

2 Proof

By the preconditions on `compute_count`, and by sorting x, the preconditions on `count_lt_eq_recursive` are satisfied.

By the definition of `count_lt_eq_recursive`, `num_lt` contains the number of values in x less than c, for each candidate, and similarly for `num_eq`.

Then the score is evaluated for each candidate in a loop. The scoring function cannot overflow or return null because the risk of overflow is protected by the preconditions. The function is also completely non-negative due to the use of `abs_diff`.