

# fn find\_min\_covering

Michael Shoemate

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

Proves soundness of **find\_min\_covering** in **mod.rs** at commit **f5bb719** (outdated<sup>1</sup>).  
**find\_min\_covering** attempts to return the smallest covering from **sets** that spans **must\_cover**.

## 1 Hoare Triple

### Precondition

#### Compiler-verified

- Argument **must\_cover** is of type **BTreeSet<T>**
- Argument **sets** is of type **HashMap<BTreeSet<T>, u32>**
- Generic **T** is some type that implements **Hash** and has **total Ord**, so that it can be used in a B-tree set.

#### Human-verified

None

### Pseudocode

```
1 def find_min_covering(  
2     must_cover: set[T], sets: list[set[T], u32]  
3 ) -> list[tuple[set[T], u32]] | None:  
4  
5     covered = list() #  
6  
7     while must_cover: #  
8  
9         def score(pair):  
10             by, weight = pair  
11             return len(by & must_cover), -len(by), -weight  
12  
13         best_match = max(sets.items(), key=score)  
14  
15         if best_match is None or best_match[0].isdisjoint(must_cover):  
16             return None  
17         best_set, weight = best_match  
18  
19         must_cover -= best_set #  
20         covered[best_set] = weight  
21  
22     return covered
```

---

<sup>1</sup>See new changes with `git diff f5bb719..0c6a4fd rust/src/domains/polars/frame/mod.rs`

## Postcondition

Return a subset of `sets` whose intersection contains `must_cover`, or `None`.

## 2 Proofs

*Proof.* All that needs to be proven is that the return set covers `must_cover`. While the algorithm makes a best effort to minimize the cardinality of the cover, nothing about optimality of the algorithm has been proven.

The algorithm initializes with an empty cover on [5](#) and continues to run until all elements have been added to the cover (see [7](#)). If there are no remaining sets that intersect with `must_cover`, then the algorithm terminates without a cover, which is a valid output.

Otherwise, on `state`, a new set is added to the cover and those elements in `must_cover` are removed. The algorithm only terminates once all members of `must_cover` have had sets that include them added to `covered`.

Therefore `covered` returns a subset of `sets` whose intersection contains `must_cover`, or `None`.

□