Rule 0.27 ⟨*Use Mappings Instead of Arrays for Data Lists*⟩

```
[...]
contract A {
  T[] arr;
  [...]
  function add(T val) {
    require(arr.length
      < max_uint256);
    arr.push(val);
  }
  function get(uint idx) {
    return arr[idx];
  }
  [...]
}
```

=

```
[...]
contract A' {
  mapping(uint => T) map;
  uint size;
  [...]
  function add(T val) {
    require(size
      < max_uint256);
    map[size] = val;
    size++;
  }
  function get(uint idx) {
    return map[idx];
  }
  [...]
}
```

where

arr is a dynamic array of type $T[]$ in contract A;

map is a mapping from **uint** to T in contract A';

size is a counter tracking the number of elements in the mapping;

T is the element type of the array and mapping values;

val is a value of type T being added;

idx is an index used to access elements;

$\max_{\mathbf{uint256}}$ represents $2^{256} - 1$.

provided

The contract does not require iterating over all elements frequently;

Element access is primarily done by index/key rather than sequential iteration;

The mapping provides sufficient functionality for the use case;

A separate *size* counter is maintained to track the number of elements;

Array operations like **push** are replaced with direct mapping assignments and size increments;

Bounds checking uses *size* instead of *arr.length*;

Both implementations include overflow protection to prevent wraparound in A and arithmetic overflow in A';

The overflow check can be implemented using **require**, custom errors, or any equivalent validation mechanism that reverts on overflow.

Invariant:

Let s_i and s'_i be the initial state of A and A', respectively.

Let s_f and s'_f be the state reached by A and A', respectively, after $A.f()$ and $A'.f()$ are executed from s_i and s'_i, respectively.

Then, the coupling invariant is

$$\forall\, s_i, s'_i\ .\ (s_i = s'_i) \rightarrow (s_f = s'_f)$$