Manual Verification

Coverage

Manual Verification would cover the auction contracts, defined by these files in onchain/src/Vulcan

```
├── Compile.hs
├── Config.hs
├── Onchain
│   ├── Auction
│   │   ├── StateMP
│   │   │   ├── Common.hs
│   │   │   ├── Private.hs
│   │   │   └── Public.hs
│   │   └── Validator
│   │       ├── Common.hs
│   │       ├── Helpers.hs
│   │       ├── Private.hs
│   │       └── Public.hs
│   ├── FinSet
│   │   ├── MP
│   │   │   ├── Common.hs
│   │   │   ├── Helpers.hs
│   │   │   ├── Separator.hs
│   │   │   └── Standard.hs
│   │   └── Validator.hs
│   ├── Market
│   │   ├── MP.hs
│   │   └── Validator.hs
│   └── NFT
│       └── MP.hs
├── SpecialUTxO
│   ├── Types.hs
│   └── Utils.hs
├── Types
│   ├── Auction.hs
│   ├── FinSet.hs
│   ├── Market.hs
│   └── State.hs
├── Utils
│   ├── List.hs
│   ├── Patterns.hs
│   ├── Prelude.hs
│   ├── Scripts.hs
│   └── Value.hs
└── Utils.hs
```

As well as 3 standalone auxiliary contracts. We expect these to look something like this. This part of the audit will start later, after we finish developing these contracts.

```
├── Onchain
│   ├── Collections
│   │   ├── BulkMint.hs
│   │   ├── DirectTransfer.hs
│   │   ├── MerkleTree.hs
│   │   ├── SequentialMint.hs
│   │   ├── Utils.hs
```

Reauditing

After the initial report, CertiK will re-audit fixed code and add it to the report. It's an interactive process to get to a point where all findings are remediated or Ikigai is comfortable. This is part of the audit.

Formal Verification

Disclaimer

This document has been hastily put together and may be incomplete. It should be treated as first draft of what we'd like verified, rather than a final, complete list.

Context

FinSet is implemented essentially as linked list (of keys) with restrictions over elements and order.

A set of keys can be represented as a collection of nodes, where each node is a pair of an optional key and and optional next key:

```
data SetNode k = Node
  { key  :: Maybe k
  , next :: Maybe k
  }
```

Properties to Verify

Has a Head & Tail

- At all times, there must be exactly one node in the collection with `key` set to `Nothing` (the "head" node), and there must be exactly one node in the collection with `next` set to `Nothing` (the "tail" node). These nodes may be the same (i.e. an empty set).
- The only way to remove the head node is to first remove all other nodes, and then de-initialise the empty set, or use the RemoveAndDeinit operation directly.

Is a Linearly Ordered Set

- Linear: the `next` field of a node can never reference more than one node. The `next` fields of two distinct nodes can never reference the same node.
- Ordered: The `next` field of a node, if it is not `Nothing`, must always reference the smallest key in the collection that is larger than the `key` of the node.
- Set: The `key` field can never be identical for two nodes.

Is Reducible

- You are always able to obtain a finite set with one (non-Head) node through removal