

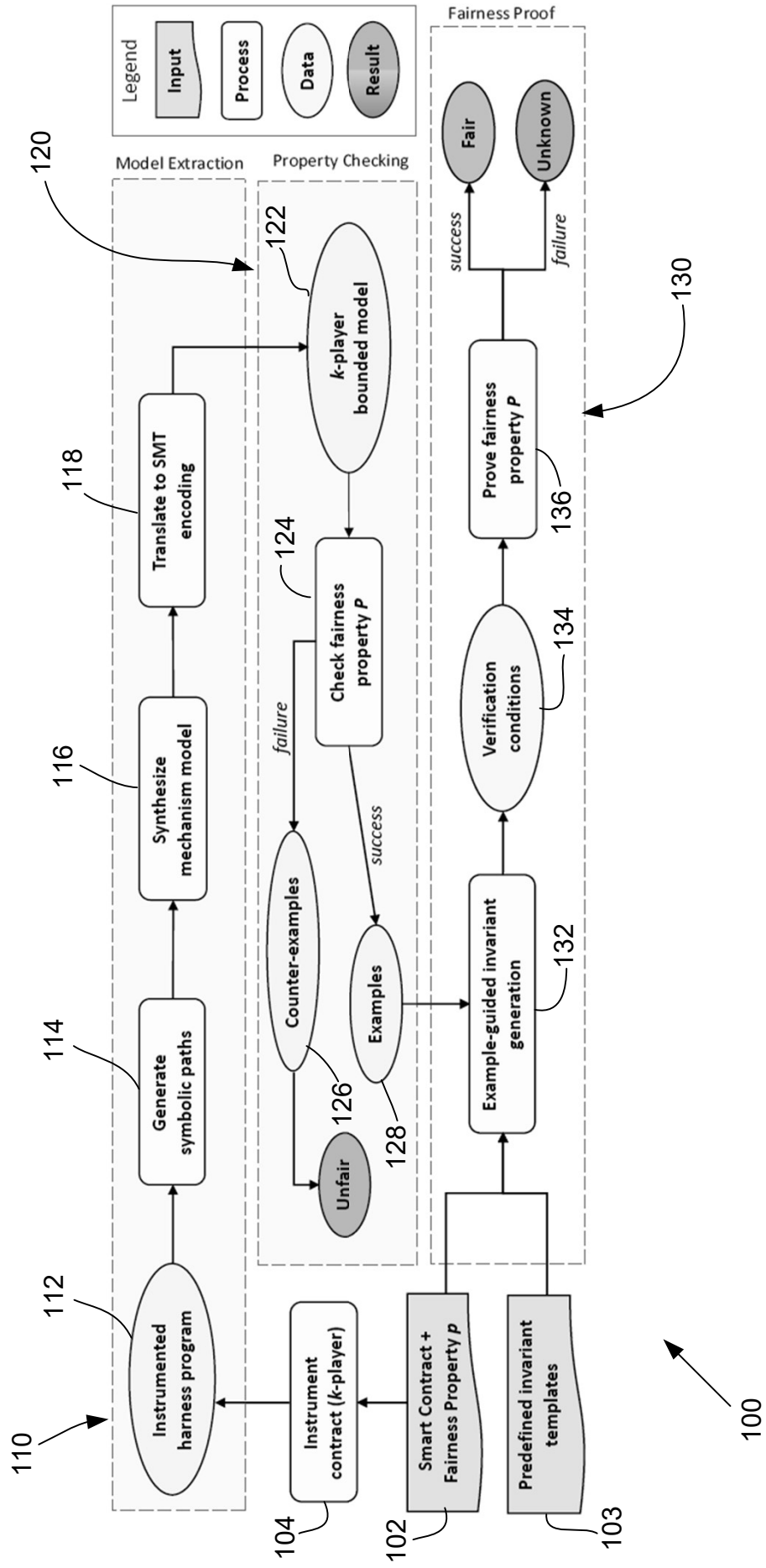


Figure 1

2/10

```
1 contract MechanismHarness {
2   // k-player bounded model
3   uint BID[k]; // symbolic values
4   uint VALUE[k]; // symbolic values
5   for (uint i=0; i<k; i++) {
6     // Example: msg.sender = i
7     require(@individual.id == toStr(i));
8     // Example: msg.value = BID[i]
9     require(@individual.bid == BID[i]);
10    // Example: bid() function inlined
11    @outcome;
12    // Example: ALLOCATE = highestBidder
13    ALLOCATE = @allocate;
14    // Example: PRICE = highestBid
15    PRICE = @price;
16    // Check loop invariant
17    // PRICE = max(BID[0..i]) ∧ ALLOCATE = arg max(BID[0..i])
18    assert(<invariant>);
19  }
20  // Check post condition
21  assert(<property>);
22 }
```

Figure 2

3/10

```
1 contract CryptoRomeAuction {
2     /** FairCon Annotations
3         @individual(msg.sender, msg.value, VALUE)
4         @allocate(highestBidder)
5         @price(highestBid)
6         @outcome(bid())
7     */
8     uint256 public highestBid = 0;
9     address payable public highestBidder;
10    mapping(address=>uint) refunds;
11    function bid() public payable{
12        uint duration = 1;
13        if (msg.value < (highestBid + duration)){
14            revert();
15        }
16        if (highestBid != 0) {
17            refunds[highestBidder] += highestBid;
18        }
19        highestBidder = msg.sender;
20        highestBid = msg.value;
21    }
22 }
```

Figure 3

4/10

```
CryptoRomeAuction := (msgsender1, msgvalue1, _)
(msgsender2, msgvalue2, _)
(msgsender3, msgvalue3, _)
assume : (not (msgvalue2 < msgvalue1 + 1)) and
          (not (msgvalue3 < msgvalue2 + 1))
allocate : argmax(msgvalue1, msgvalue2, msgvalue3)
price : max(msgvalue1, msgvalue2, msgvalue3)
```

Figure 4

5/10

```
<individual> := (id :  $\mathbb{S}$ , bid :  $\mathbb{N}$ , val :  $\mathbb{N}$ )  
  <func> := max | argmax  
    <exp> := <individual>.id | <individual>.bid  
            |  $\mathbb{N}$  | <exp> [+ -] <exp> | <func>(<exp>*)  
  <bool> := <exp> == <exp> | <exp> < <exp>  
            | not <bool> | <bool> and <bool>  
<assumption> := assume : <bool>  
  <outcome> := allocate : <exp>  
              | price : <exp>; allocate : <exp>  
  <property> := <bool> | forall : <bool>  
<mechanism> := <individual>*; <assumption>; <outcome>;  
              <property>
```

Figure 5

6/10

$$\begin{array}{c} \frac{(id_1, bid_1, val_1), \dots, (id_n, bid_n, val_n)}{N \leftarrow \{1, \dots, n\} \quad \hat{\theta} \leftarrow \{bid_1, \dots, bid_n\} \quad \{v_i(o_i, \theta_i)\} \leftarrow \{val_i\}} \text{ [Indiv]} \\[1em] \frac{\mathbf{assume} : assumption \quad \mathbf{allocate} : allocation}{d(\hat{\theta}) \leftarrow \mathbf{eval}(assumption \wedge allocation)} \text{ [Alloc]} \\[1em] \frac{\mathbf{assume} : assumption \quad \mathbf{price} : clearprice}{t(\hat{\theta}) \leftarrow \mathbf{eval}(assumption \wedge clearprice)} \text{ [Price]} \end{array}$$

Figure 6

7/10

```
1 contract EtherAuction {
2   //Anyone can bid by calling this function and supplying the
   ↪ corresponding eth
3   function bid() public payable {
4       require(auctionStarted);
5       require(now < auctionEndTime);
6       require(msg.sender != auctioneer);
7       // If sender is already the highest bidder, reject it.
8       require(highestBidder != msg.sender);
9       address _newBidder = msg.sender;
10      uint previousBid = balances[_newBidder];
11      uint _newBid = msg.value + previousBid;
12      // Each bid has to be 0.05 eth higher
13      if (_newBid == highestBid + (5 * 10 ** 16)) return;
14      // The highest bidder is now the second highest bidder
15      secondHighestBid = highestBid;
16      secondHighestBidder = highestBidder;
17      highestBid = _newBid;
18      highestBidder = _newBidder;
19      latestBidTime = now;
20      // Update the bidder's balance so they can later withdraw
   ↪ any pending balance
21      balances[_newBidder] = _newBid;
22  }
23 }
```

Figure 7

8/10

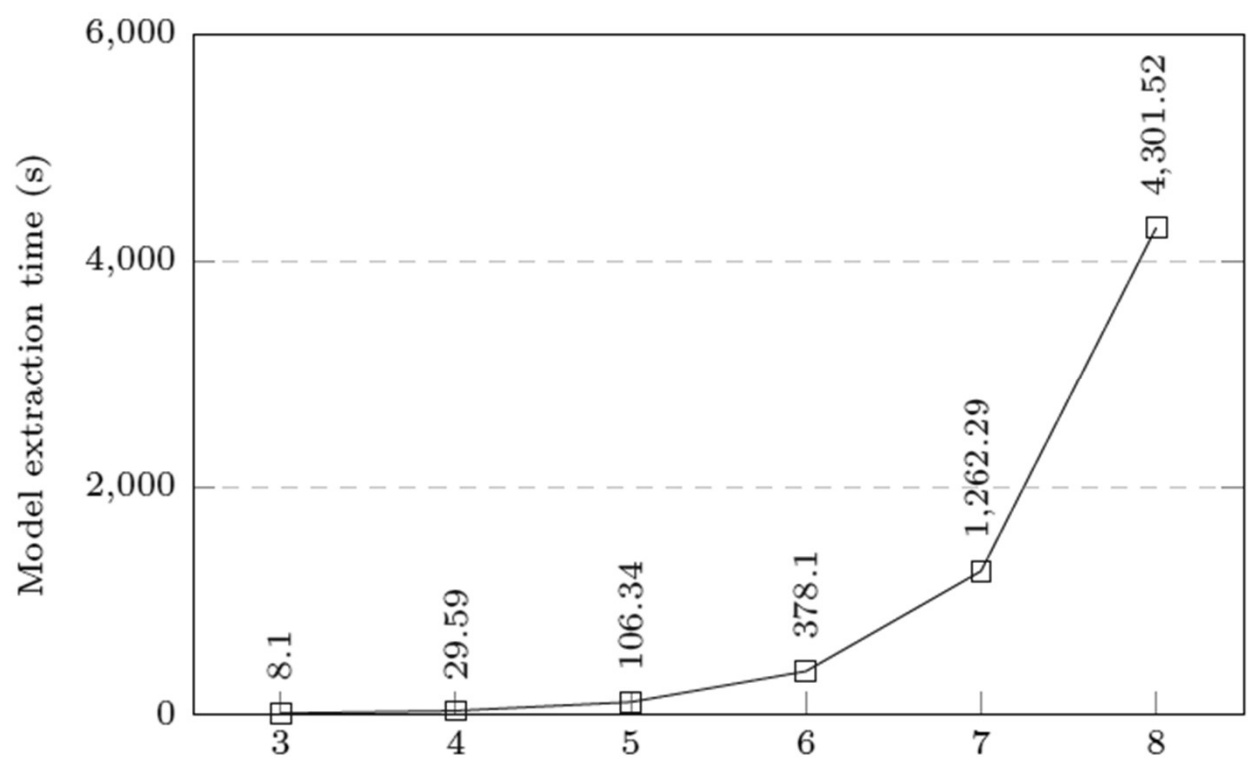


Figure 8

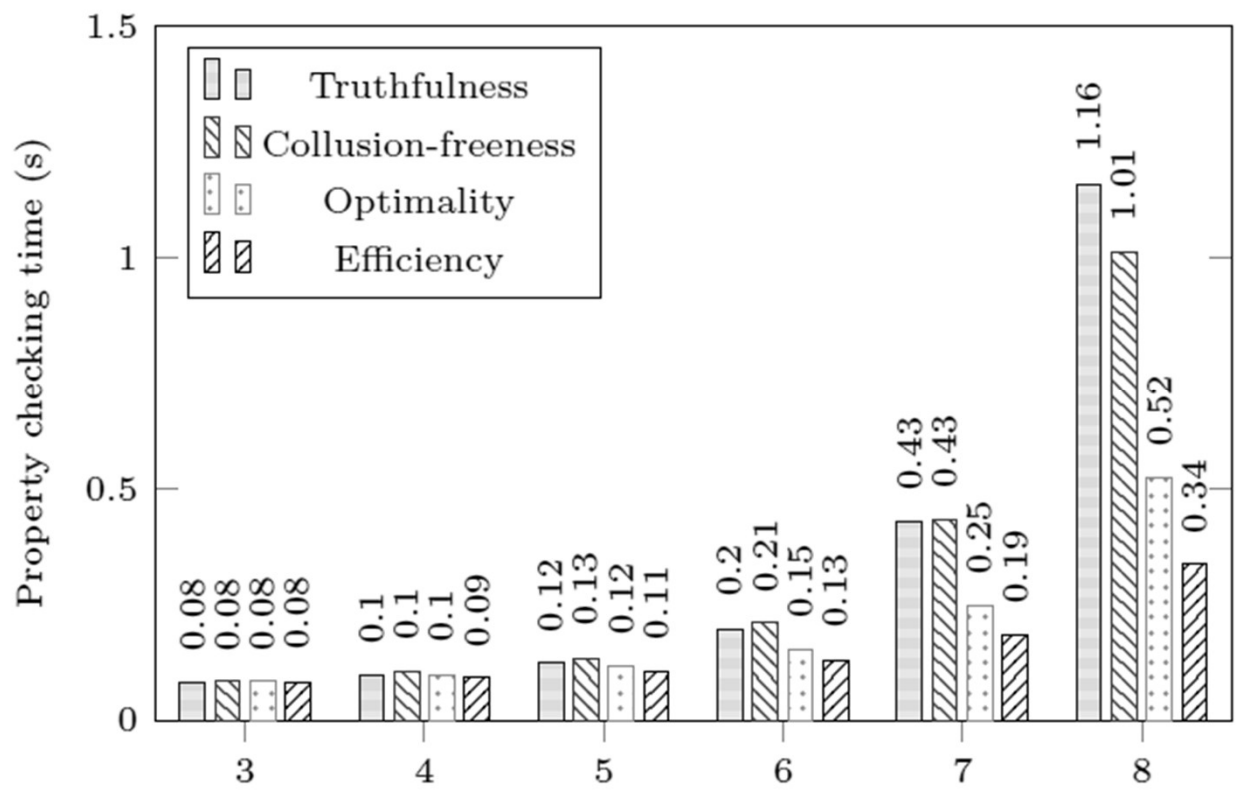


Figure 9

10/10

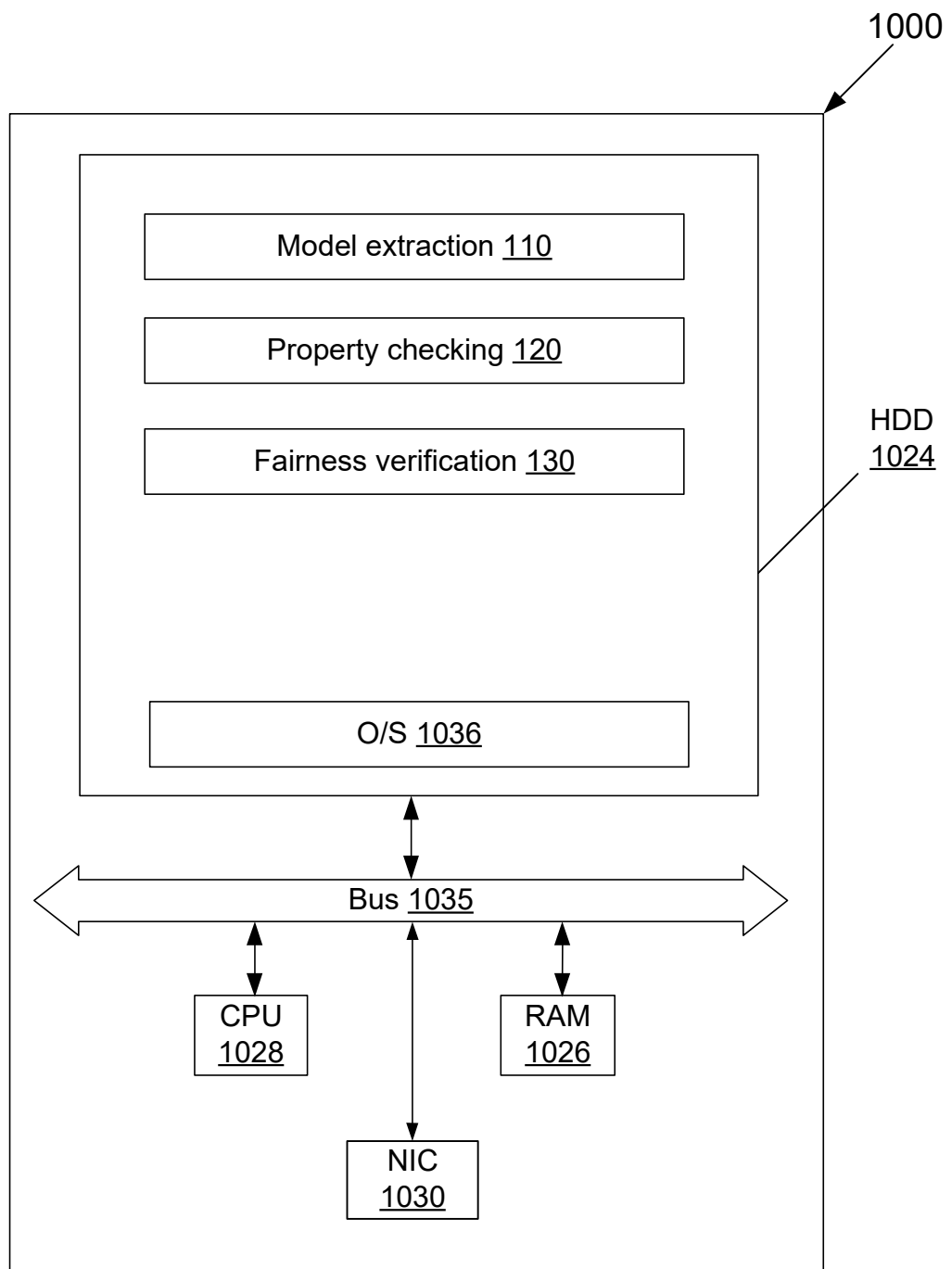


Figure 10