VETSC: Towards Automated Safety Vetting of Smart Contracts in Decentralized Applications

Yue Duan
Illinois Institute of Technology

Smart Contract

Physical contracts

Trusted 3rd party

Slow speed

High cost

Smart Contract

Spontaneous

Fully automated

High speed

Secure

Decentralized Applications

```
function bidOnAuction(uint _id) public payable {
    uint256 ethAmountSent = msg.value;

    // owner cannot bid on his/her own merchandise
    Auction memory myAuction = auctions[_id];
    if(myAuction.owner == msg.sender) revert();

    // check whether auction has expired
    if(block.timestamp > myAuction.deadline) revert();

    // check whether previous bids exist
    uint bidsLength = accepted[_id].length;
    uint256 tempAmount = myAuction.startPrice;
    Bid memory lastBid;
    if(bidsLength > 0) {
        lastBid = accepted[_id][bidsLength-1];
        tempAmount = lastBid.amount;
    }

    // check if bid price is greater than the current highest
    if(ethAmountSent < tempAmount) revert();

    // refund the last bidder
    if(bidsLength > 0)
        if(!lastBid.from.send(lastBid.amount)) revert();

    // add the new bid to auction state
    Bid memory newBid;
    newBid.from = msg.sender;
    newBid.amount = ethAmountSent;
    accepted[_id].push(newBid);
    emit BidSuccess(msg.sender, _id);
}
```

Owner can cancel at ANYTIME. Safety risk!

```
function cancelAuction(uint _id) public isOwner(_id) {
    Auction memory myAuction = auctions[_id];
    uint bidsLength = accepted[_id].length;

    // refund the last bid, if prior bids exist
    if(bidsLength > 0) {
        Bid memory lastBid = accepted[_id][bidsLength - 1];
        if(!lastBid.from.send(lastBid.amount)) revert();
    }
    auctions[_id].active = false;
    emit AuctionCanceled(msg.sender, _id);
}
```

Smart Contract Safety Vetting

- Challenge:
  Understanding function and variable semantics

terminate a bidding process

```
35 function cancelAuction(uint _id) public isOwner(_id) {
36     Auction memory myAuction = auctions[_id];
37     uint bidsLength = accepted[_id].length;
38
39     // refund the last bid, if prior bids exist
40     if(bidsLength > 0) {
41         Bid memory lastBid = accepted[_id][bidsLength -
        1];
42         if(!lastBid.from.send(lastBid.amount)) throw;
43     }
44     auctions[_id].active = false;
45     emit AuctionCanceled(msg.sender, _id);
46 }
```

prior accepted bids

auction state

Smart Contract Safety Vetting

- Ex

unreliable
terious and error-prone
unavailable

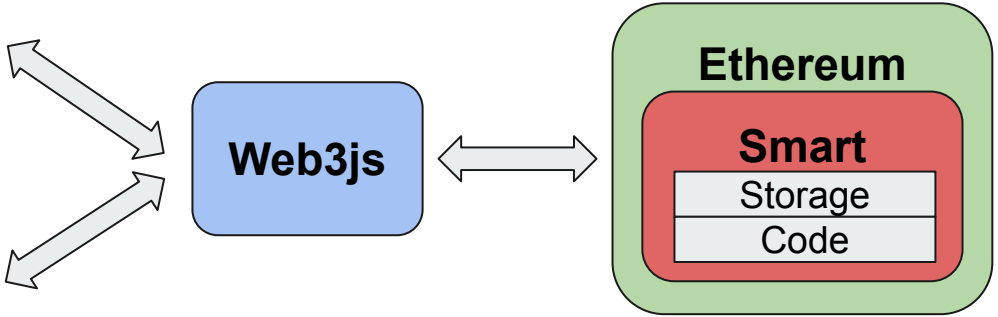
  - M
    exception m and cy, etc.)
  - Semantic i s extracted manually
    - from variable names, function names, etc

Decentralized Applications

GUI Front-end

middleware

backend

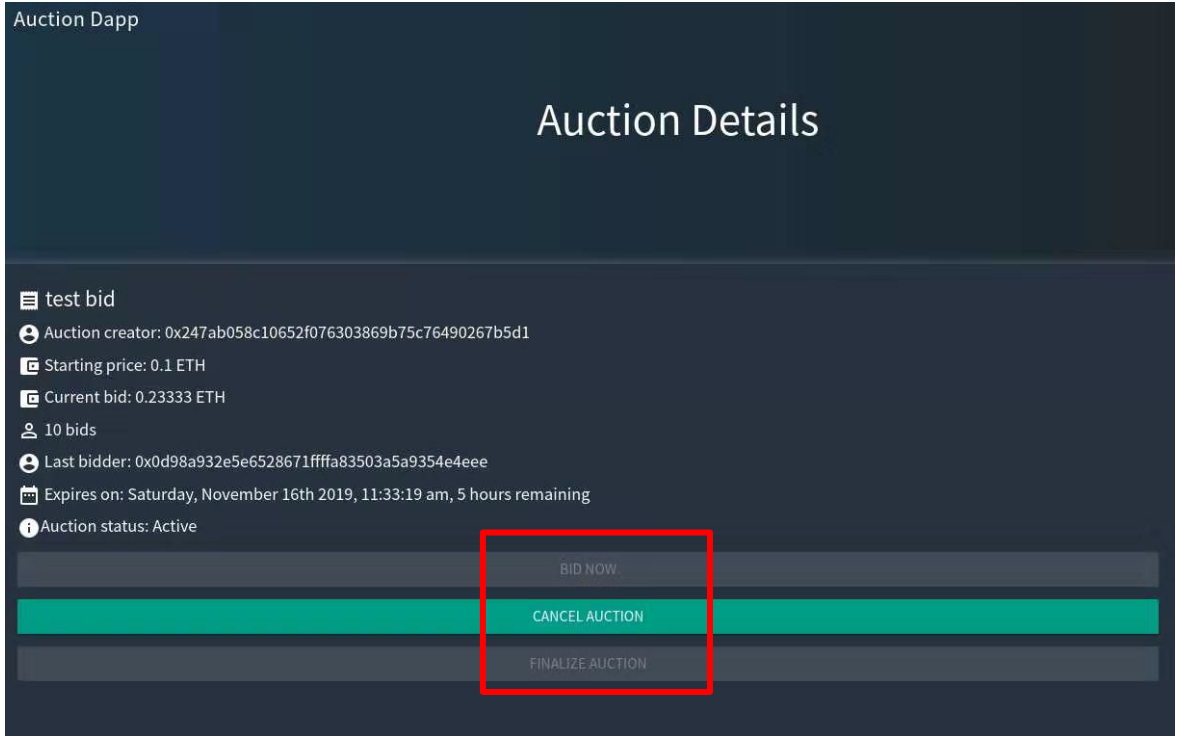


Our Insights

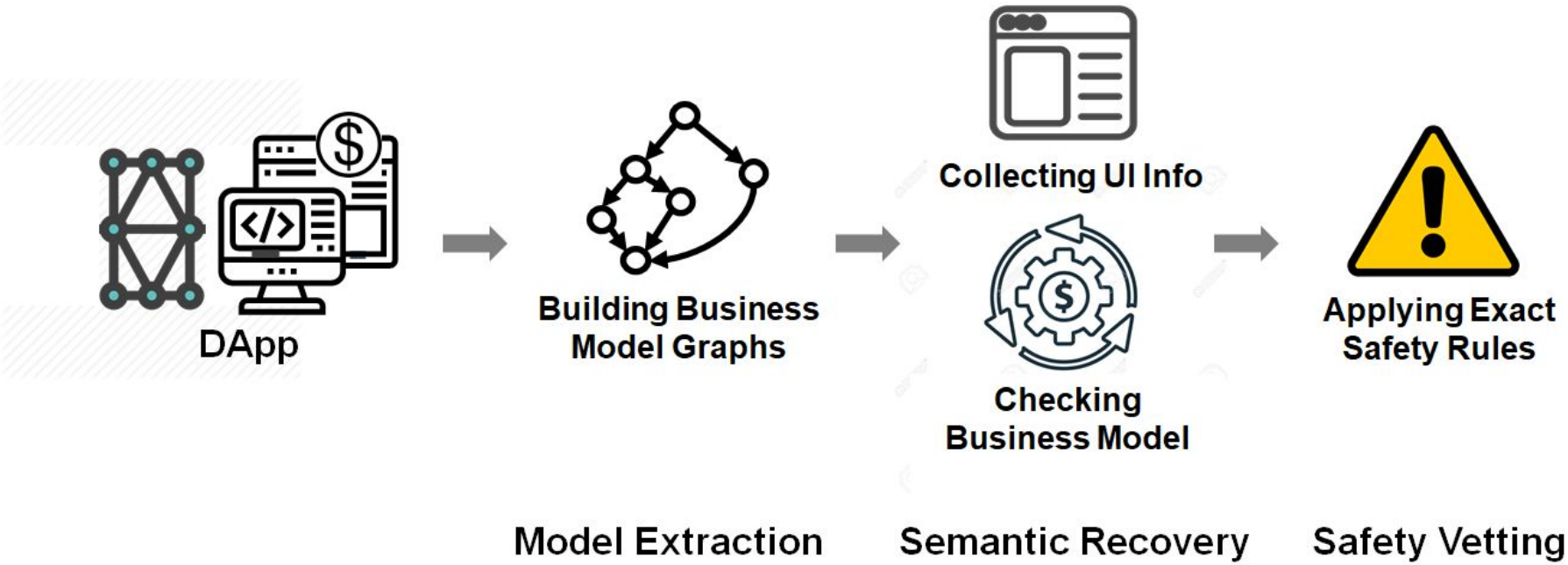
- Limited types of business logics
  - auction
  - voting
  - trading
  - gambling
  - wallet
  - crowdsale

- UI contains semantics info

Semantic Inference



VetSC Overview



Business Model Graph

- To represent high-level business logic
  - key factors
    - transaction properties
      - msg.sender
      - msg.value
      - block.timestamp
      - etc

```
function bidOnAuction(uint _id) public payable {
    uint256 ethAmountSent = msg.value;

    // owner cannot bid on his/her own merchandise
    Auction memory myAuction = auctions[_id];
    if(myAuction.owner == msg.sender) revert();

    // check whether auction has expired
    if(block.timestamp > myAuction.deadline) revert();

    // check whether previous bids exist
    uint bidsLength = accepted[_id].length;
    uint256 tempAmount = myAuction.startPrice;
    Bid memory lastBid;
    if(bidsLength > 0) {
        lastBid = accepted[_id][bidsLength-1];
        tempAmount = lastBid.amount;
    }

    // check if bid price is greater than the current highest
    if(ethAmountSent < tempAmount) revert();

    // refund the last bidder
    if(bidsLength > 0)
        if(!lastBid.from.send(lastBid.amount)) revert();

    // add the new bid to auction state
    Bid memory newBid;
    newBid.from = msg.sender;
    newBid.amount = ethAmountSent;
    accepted[_id].push(newBid);
    emit BidSuccess(msg.sender, _id);
}
```

Business Model Graph

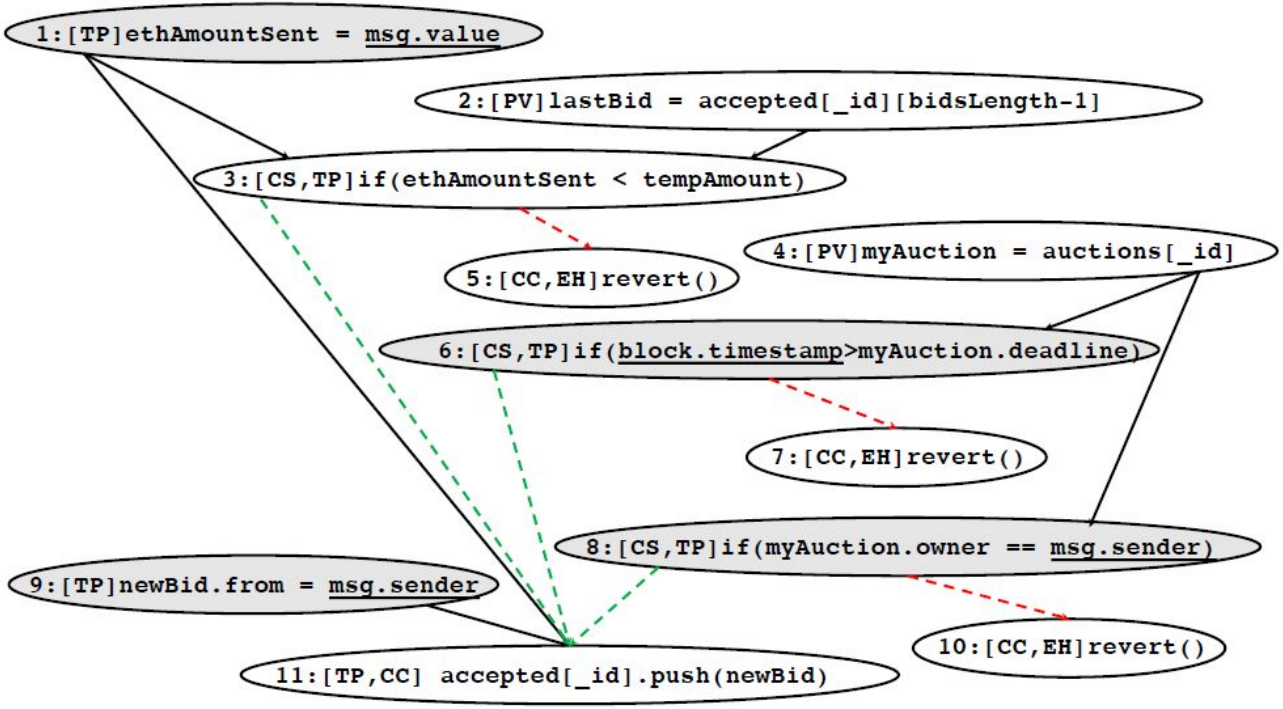
- To represent high-level business logic
  - key factors
    - transaction properties
    - global variables
    - dataflow
    - condition check
    - cryptocurrency transfer

```
1  function bidOnAuction(uint _id) public payable {
2      uint256 ethAmountSent = msg.value;
3
4      // owner cannot bid on his/her own merchandise
5      Auction memory myAuction = auctions[_id];
6      if(myAuction.owner == msg.sender) revert();
7
8      // check whether auction has expired
9      if(block.timestamp > myAuction.deadline) revert();
10
11     // check whether previous bids exist
12     uint bidsLength = accepted[_id].length;
13     uint256 tempAmount = myAuction.startPrice;
14     Bid memory lastBid;
15     if(bidsLength > 0) {
16         lastBid = accepted[_id][bidsLength-1];
17         tempAmount = lastBid.amount;
18     }
19
20     // check if bid price is greater than the current
       highest
21     if(ethAmountSent < tempAmount) revert();
22
23     // refund the last bidder
24     if(bidsLength > 0)
25         if(!lastBid.from.send(lastBid.amount)) revert();
26
27     // add the new bid to auction state
28     Bid memory newBid;
29     newBid.from = msg.sender;
30     newBid.amount = ethAmountSent;
31     accepted[_id].push(newBid);
32     emit BidSuccess(msg.sender, _id);
33 }
```

Business Model Graph

- A Business Model Graph is:
  - directed graph G = (V, E, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$) over statements $\boldsymbol{\Sigma}$ and relations R
  - V: statements in $\boldsymbol{\Sigma}$
  - E: causal dependencies between statements
  - $\boldsymbol{\alpha}$: V -> $\boldsymbol{\Sigma}$
    - labeling function that associates nodes with labels
  - $\boldsymbol{\beta}$: V -> R
    - labeling function that associates edges with labels

Business Model Graph

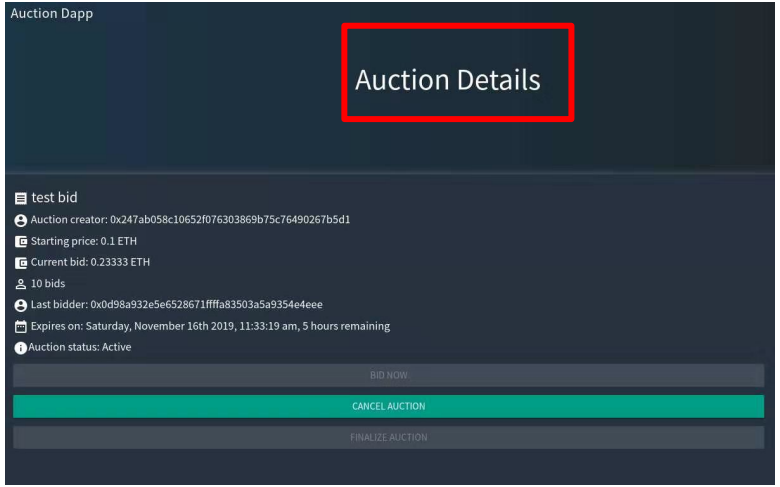


Business Model Graph

- Memory Aliasing
  - two global memory regions
    - storage
    - memory

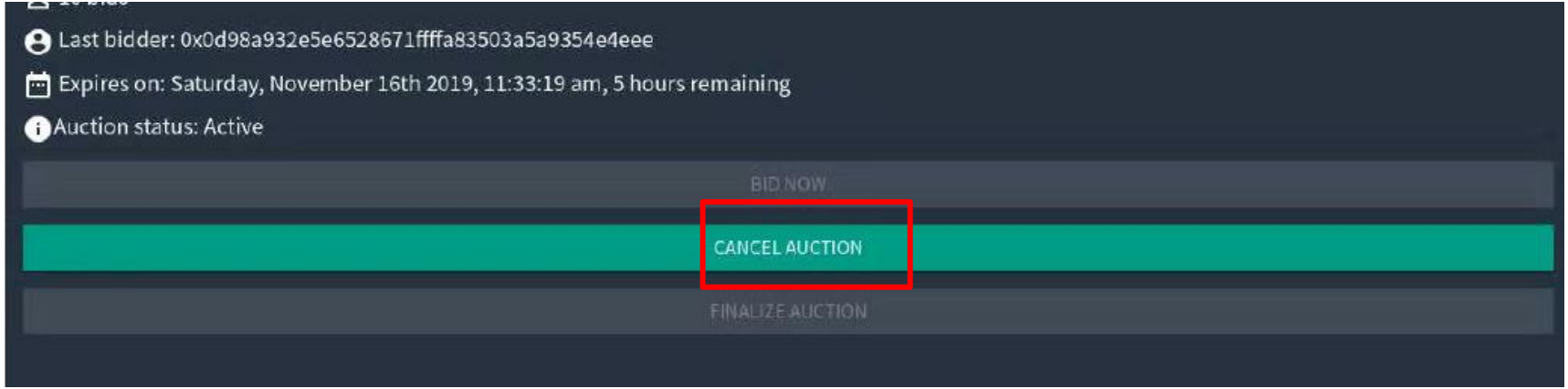
Semantics Recovery

- Three levels of semantics recovery
  - smart contract level
    - HTML parsing ⇒ NLP technique ⇒ semantics



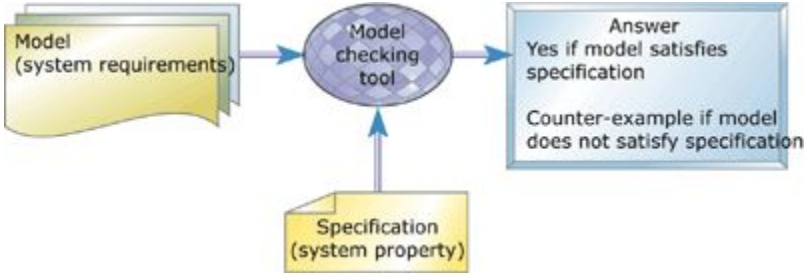
Semantics Recovery

- Three levels of semantics recovery
  - function level
    - GUI text ⇒ NLP technique ⇒ semantics

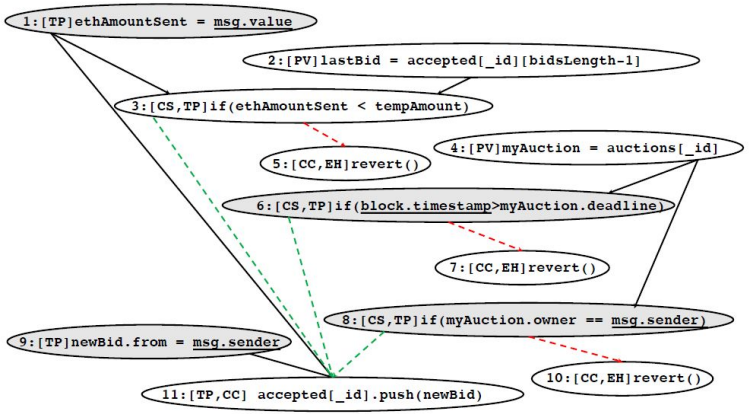


Semantics Recovery

- Three levels of semantics recovery
  - variable level
    - model checking
      - pre-defined specs
      - extracted models



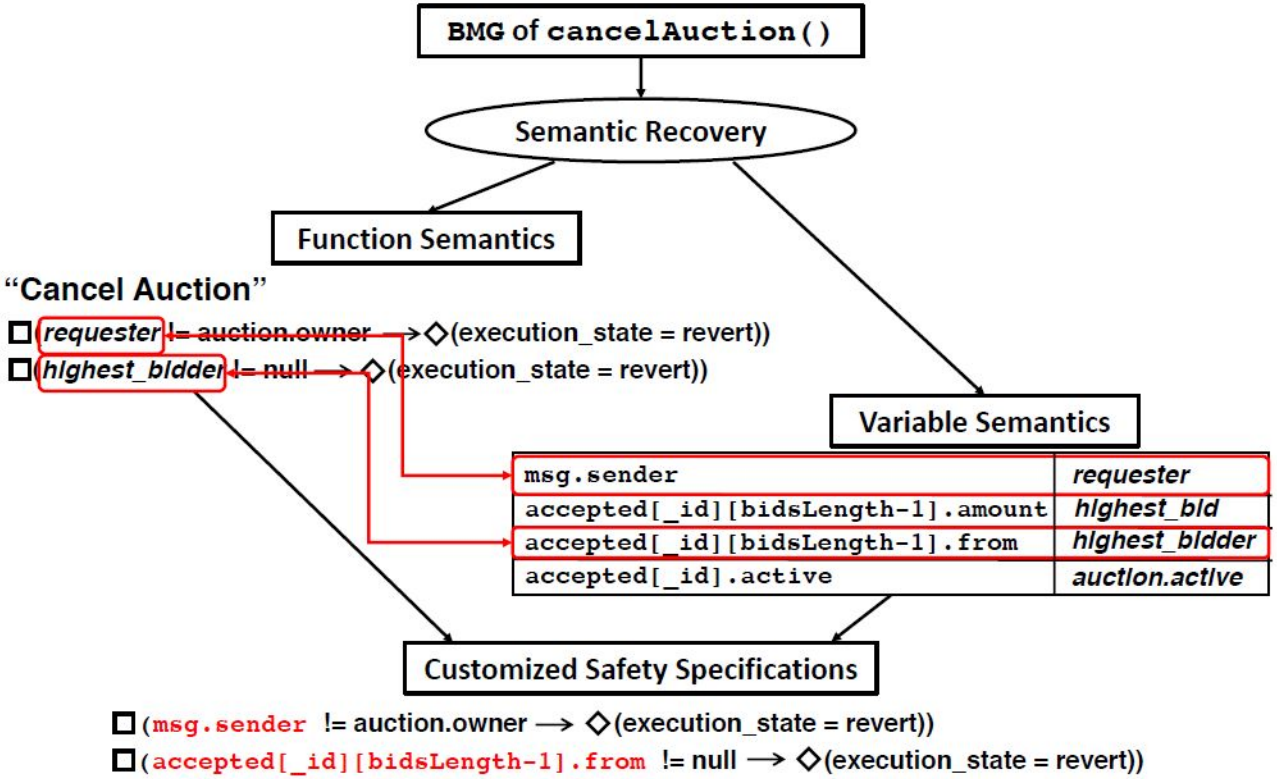
Semantics Recovery



NuSMV Model

Function	Spec Type	Formal Spec
Bidding	Essential#1	$\Box(current\_bid > highest\_bid \rightarrow \Diamond(highest\_bid := current\_bid \wedge highest\_bidder := current\_bidder))$
	Safety#1	$\Box(current\_time > deadline \rightarrow \Diamond(execution\_state := revert))$
	Safety#2	$\Box(auction.active == false \rightarrow \Diamond(execution\_state := revert))$
	Safety#3	$\Box current\_bidder == auction.owner \rightarrow \Diamond(execution\_state := revert))$
Cancel	Essential#1	$auction.active := false$
	Safety#1	$\Box(requester != auction.owner \rightarrow \Diamond(execution\_state := revert))$
	Safety#2	$\Box(highest\_bidder != null \rightarrow \Diamond(execution\_state := revert))$

Customized Spec Generation



Semantics Recovery

- Recovered semantics

Function/Domain	Smart Contract Variable	Spec Concept
`bidOnAction`	`msg.value`	*current_bid*
`bidOnAction`	`msg.sender`	*current_bidder*
`bidOnAction`	`newBid.amount`	*highest_bid*
`bidOnAction`	`newBid.from`	*highest_bidder*
`cancelAction`	`msg.sender`	*requester*
contract-wide	`accepted[_id][bidsLength-1].amount`	*highest_bid*
contract-wide	`accepted[_id][bidsLength-1].from`	*highest_bidder*
contract-wide	`accepted[_id].active`	*auction.active*

Evaluation

- Dataset
  - 24 real-world DApps
  - 465 solidity functions
- Comparison with state-of-the-art
  - VerX

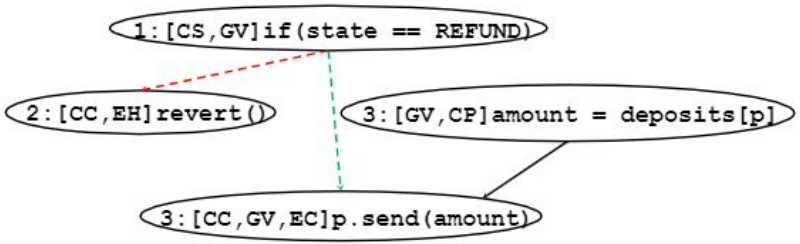
Evaluation

#	Name	Unsafe Func Name	Code Logic	Major Widget Text/Context	UI == Logic?	Safety Issue in Smart Contracts	Violated Policy	Source Analyzability
1	cryptoatoms.org	-	-	-	Yes	-	-	Yes
2	proofoflove.digital	-	-	-	Yes	-	-	Yes
3	snailking	-	-	-	Yes	-	-	Yes
4	cryptominingwar	-	-	-	Yes	-	-	Yes
5	market.start.solar	-	-	-	Yes	-	-	No (Missing Source)
6	etheroll	-	-	-	Yes	-	-	No (Inlined Bytecode)
7	cryptokitties	`bid()`	Auction-Bid	*"buy"*	Ambiguity	N/A	N/A	Yes
8	hyperdragons	-	-	-	Yes	-	-	No (Missing Source)
9	dice2.win	-	-	-	Yes	-	-	No (Inlined Bytecode)
10	all-for-one.club	`drawNow()`	Lottery-Draw	*"Draw"*	Yes	Drawing for an expired lottery	Lottery-Draw-S2	No (Inlined Bytecode)
		`play()`	Lottery-Buy	*"pay 1 ETH"*	Yes	Buying an expired ticket	Lottery-Buy-S1	No (Inlined Bytecode)
11	openberry-ac	`placeBid()`	Auction-Bid	*"place BID"*	Yes	Bidding for an expired auction	Auction-Bid-S1	Yes
		`finalizeAuction()`	Auction-Close	*"handle Finalize"*	Yes	Closing a non-expired auction	Auction-Close-S1	Yes
						Closing an active auction	Auction-Close-S2	
12	create-react-dapp	`voteForCandidate()`	Voting-Vote	*"vote Rama/Nick/Jose"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
						Double voting	Voting-Vote-S2	
13	ethereum-voting	`vote()`	Voting-Vote	*"Vote"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
14	ethereum-wallet	-	-	-	Yes	-	-	Yes
15	heiswap.exchange	-	-	-	Yes	-	-	No (Inlined Bytecode)
16	Lottery-DApp	`makeGuess()`	Lottery-Buy	*"Buy", "Lottery"*	Yes	Buying an expired ticket	Lottery-Buy-S1	Yes
		`closeGame()`	Lottery-Draw	*"Close Game", "Lottery"*	Yes	Drawing for an expired lottery	Lottery-Draw-S2	Yes
17	mastering-e-a-d	`cancelAuction()`	Auction-Cancel	*"CANCEL AUCTION"*	Yes	Seller cancel after bidding starts	Auction-Cancel-S2	Yes
18	multisender.app	-	-	-	Yes	-	-	Yes
19	note_dapp	-	-	-	Yes	-	-	Yes
20	metacoin	-	-	-	Yes	-	-	Yes
21	simple-vote	`vote()`	N/A	*"Start a vote"*	No Impl.	N/A	N/A	Yes
22	truffle-voting	`vote()`	Voting-Vote	*"Approve/Against/Abstain"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
23	Overview	`invest()`	CS-Invest	*"Buy tokens", "Crowdsale"*	Yes	Invest an expired crowdsale	CS-Invest-S2	Yes
24	Crowdsale	-	-	-	Yes	-	-	Yes

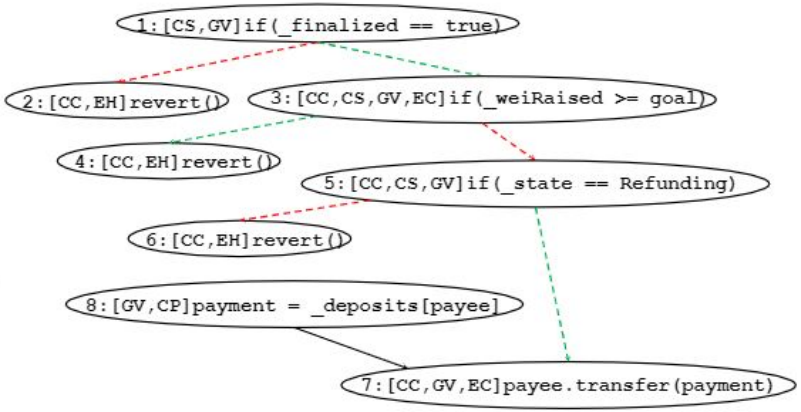
Evaluation - Source Analyzability

```
function appendInt(buffer memory buf, uint data, uint len)
    internal constant returns(buffer memory) {
    ...
    assembly {
        let bufptr := mload(buf)
        let buflen := mload(bufptr)
        let dest := add(add(bufptr, buflen), len)
        mstore(dest, or(and(mload(dest), not(mask)), data)
    )
        mstore(bufptr, add(buflen, len))
    }
    return buf;
}
```

Evaluation- Comparison with VerX



(c) BMG of `refund()` in *Overview* app



(d) BMG of `refund()` in *Crowdsale* app

Analysis Step	VETSC	Verx
(a) High-level Specs	One-Time Manual Effort	None
(b) Semantic Recovery	Automated (UI-Guided)	Manual Effort for Each Func
(c) Specs Customization	Automated	Manual Effort for Each Func
(d) Safety Verification	Automated	Automated

Evaluation

- Comparison with VerX

Analysis Step	VETSC	Verx
(a) High-level Specs	One-Time Manual Effort	None
(b) Semantic Recovery	Automated (UI-Guided)	Manual Effort for Each Func
(c) Specs Customization	Automated	Manual Effort for Each Func
(d) Safety Verification	Automated	Automated

Conclusion

- Automated Safety Vetting of Smart Contracts in DApps
- Semantic inference
  - smart contract level
  - function level
  - variable level
- Evaluation on real-world DApps